A Unified Interface for Experimentation at the Edge

*Initial Thoughts*

Srikanth Sundaresan

ICSI
Measurements from the edge are critically important

• Broadband is a critical resource
  – Not a luxury anymore

• View from the outside just as important as the view from inside

• The edge is as complex as the core
  – If not more – problems are devilishly difficult to pinpoint, let alone solve
... Which explains why there are so many platforms
Do we need so many platforms?

<table>
<thead>
<tr>
<th></th>
<th>BISmark</th>
<th>Ark</th>
<th>SamKnows</th>
<th>RIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous active</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Passive</td>
<td>Y</td>
<td>N</td>
<td>Y/N</td>
<td>N</td>
</tr>
<tr>
<td>Scope of experiments</td>
<td>High</td>
<td>Higher (better CPU/storage)</td>
<td>Medium(resource constraints)</td>
<td>Low (only use tools compiled in)</td>
</tr>
<tr>
<td>Heavy duty exp</td>
<td>?</td>
<td>Y</td>
<td>N</td>
<td>Y/N</td>
</tr>
<tr>
<td>Local storage</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Scale</td>
<td>~</td>
<td>~</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Each platform is unique, valuable in its own right.
As a researcher, what would one choose?

• Considering experiment that can potentially run on all platforms:
  – Scale
  – Ease of deployability

• Experiment deployability is important
  – Else platform will never be used outside of niche group
How easily are experiments deployable in current platforms?

• BISmark – not difficult (?)
  – Comfortable with openwrt
  – Ash, C, lua
  – Short turn-around times (weeks)

• Ark ?

• SamKnows
  – Ash, C
  – Long turn-around times (months)
Can we write experiments once and deploy everywhere?

• It’s complicated
  – Technically possible
    • Standard cross-compilation techniques
    • A few hardware/other quirks (interface names, etc)
    • Some effort in integrating with existing experimentation method
  – More difficult in practice
    • Memory / CPU / bandwidth / time constraints
    • Need to make code general-purpose
Case study: Porting WtF to SamKnows

• Where’s the Fault
  – Tool that localizes throughput bottlenecks to access link or wireless gateway
  – Collects passive pcaps
  – Proof-of-concept code in Ash + custom small C modules
  – Extensively tested on BISmark
    • 65+ homes, 2 months
  – FCC got interested in June 2013
Timeline of porting WtF to SamKnows

• Summer 2013
  – Realized that Netgear 3500 has Broadcom chipset, which reduces functionality
  – 2/3 of nodes which has Atheros chipset is deployed off-path
• Fall 2013
  – Proof-of-concept code that works flawlessly in BISmark but fails miserably in SamKnows
• Spring 2014
  – Ported WtF as a lightweight, predominantly C-based program
• Summer 2014
  – Early testing + adding features
  – Testing on 100 nodes (still larger than entire BISmark deployment)
• Late summer 2014
  – Initial deployment
  – … which got postponed due to FCC MBA measurements cycle
• Fall 2014
  – Deployment!
  – Wholesale crash of 30-40% of nodes within 36 hrs
  – Experiment pulled (we did get some really interesting data though!)
A unified experiment development platform

• Is there a standard development platform we can agree upon and enforce?
  – C/C++ with Shell/Lua

• Some “basic” constraints / good habits
  – Memory, CPU, storage, network utilization
  – Real people may be using the network!!

Can we impose tight constraints and maintain usefulness of platforms?
Keep management small and separate

• Experiment vetting
  – Does it meet ToS of platform?
  – Security (hard!)
  – Resource utilization (hard!)
  – But likely only needs to be done on one platform

• Constraints should be managed by experiment
  – Hardware
    • A wireless component that works on BISmark should fail gracefully on Ark
  – Keep resource utilization minimal
Basic assumptions

• Simple packaging system
  – Expecting users to figure out packaging for every single platform is expecting too much
    • Openwrt makefiles are not pretty
  – Give us a pointer to the code repo, we’ll generate the package

• Package management system
  – Pick nodes
  – If the code repo updates, the deployments update
BISmark and Ark

• Probably easiest to integrate (externally, not internally)
  – Similar (yet different) vision, platform
• The researcher needs to provide
  – Code that is platform-agnostic
• The platform provider needs to provide
  – Platform-specific package management
  – Integration into experiment universe (crontab)
  – Nodes
  – Data pipeline
So what should the platforms provide?

• Maintain an open, easy-to-use development toolchain (easy)
  – Keep platform-specific build management separate from code

• Sync data
  – Can be offline

• Provide list of constraints (easy)
  – Memory, CPU, network usage, time

• Enforce constraints (hard)
  – Sandboxing: very difficult, if not impossible to vet experiments or deploy without losing sleep
Practical first step

• Run basic experiments on each others’ platforms
  – Bismark-active: periodically measures latency, throughput, packet loss, jitter
  – Something light-ish from Ark?

BISmark → bismARK*

* This might not happen