Introduction

• Future networks characterized by mobile code
• Platform independence
• Wide range of computing devices
  – Desktop, embedded systems, PDAs
• Java, Inferno, …
Current VM Shortcomings

- **Manageability**
  - No central control

- **Performance**
  - Processing/memory requirements

- **Security**
  - Large Trusted Computing Base (TCB)

- **Scalability**
  - Monolithic architecture does not scale
Paper Goal

- Distributed virtual machine architecture
- Factor VM services into logical components
- Move services from client to network
- Support centralized management of critical VM functions
Paper Overview

- Introduction
- Architecture Overview
- Services
- Performance Evaluation
- Optimizations
- Related Work / Conclusions
DVM Architecture
VM Architecture

Code → Cache Check → Verifier → Security → Compiler → Optimizer → Profiler → Cache → Runtime

- Check type safety
- Check static rules
- Annotate for dynamic checks
- Translate to native format
- Transform code for performance
- Collect data on program behavior

- Check signatures
- Execute program
Distributed VM Architecture

- Distributing VM services:
  - Can’t use existing monolithic interfaces
  - Factor functionality into static & dynamic components
- Static functionality can be moved around
- Dynamic functionality executed at client
Static & Dynamic Functionality

Glue = binary code rewriting
Deployment Architecture

Java Page
http://java.foo.com

Internet

HTTP Proxy
DVM Server
- Static Services
- Rewrite Java bytecode (dynamic services)

java.foo.com
HTTP Server

Client
- Classic VM or mini VM
DVM Services
DVM Services: Verification

- **Monolithic Verifier Issues**: 
  - Different implementations
  - Security holes cannot be centrally patched
  - Memory/processor requirements (HP dropped it)

- **Phases 1, 2, 3**: check class in isolation
  - Statically at server

- **Phase 4**: verify imports
  - Dynamically at client (rewriting)
Verification Rewriting

Class Hello {
    static boolean __mainChecked = false;
    public static void main() {
        if (__mainChecked == false) {
            RTVerifier.checkField("java.lang.System", "out", "java.io.OutputStream");
            __mainChecked = true;
        }
        System.out.println("hello world");
    }
}
DVM Services: Security

- Uniformly enforce organ. security policy
- Central point of policy specification
- Impose checks in any part of code
Security Service Highlights

- Centralized policy enforcement
  - DTOS (mach) model
  - XML-based access matrix
- Control of arbitrary resource access
  - Not dependent on app. programmer checks
- During execution query security service
  - cache + invalidation protocol
- Cons: Download an enforcement manager
- Other services: remote monitoring/profiling + compilation
Performance Evaluation
Performance Evaluation

- Sun JDK 1.2
  - Vanilla v.s.
  - Removed monolithic services

- Discard 3/Average 5

- DVM 11% slower
  - Proxy parsing overhead

- Pizza (compiler) ??
Client-Side Overhead

- JDK 1.2 verifier written in C
- Still, DVM clients perform better
Security Runtime Overhead

- **Check:** wall-clock time
- **Overhead:** checked - baseline
- **Download:** global security policy overhead

<table>
<thead>
<tr>
<th>Description</th>
<th>Baseline (no check)</th>
<th>JDK (check)</th>
<th>JDK (overhead)</th>
<th>DVM (download)</th>
<th>DVM (check)</th>
<th>DVM (overhead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Property</td>
<td>0.0020</td>
<td>0.0488</td>
<td>0.0468</td>
<td>5.830</td>
<td>0.0092</td>
<td>0.0072</td>
</tr>
<tr>
<td>Open File</td>
<td>1.406</td>
<td>8.631</td>
<td>7.224</td>
<td>6.406</td>
<td>1.430</td>
<td>0.0238</td>
</tr>
<tr>
<td>Change Thread Priority</td>
<td>0.0638</td>
<td>0.0645</td>
<td>0.0007</td>
<td>5.026</td>
<td>0.0815</td>
<td>0.0177</td>
</tr>
<tr>
<td>Read File</td>
<td>0.0141</td>
<td>N/A</td>
<td>N/A</td>
<td>4.146</td>
<td>0.0368</td>
<td>0.0227</td>
</tr>
</tbody>
</table>
Proxy Overhead

- 100 random applets
- No caching
- Average download 2198ms
  - std dev 3752ms (large)
- Proxy adds 265ms (12% overhead)
DVM Proxy Scaling

- Clients 0-250
- 1 -- 1.2 sec/kB latency
- No caching
- Degradation >250
  - Memory (64MB)
Class Download Optimizations

- Class byte-code download:
  - One at a time, or as .jar file
- Not efficient for low bandwidth/high latency
- 10-30% of downloaded code never invoked
- Solution: restructure classes at server based on profiles
Conclusions

• New system architecture (DVM)
• Factor VM services
  – Centralize security services
  – Reduce client requirements
• Performance comparable to monolithic VMs
Issues?

- Comparison to application servers
- Assume trusted internal network
- Assume single point of entry (proxy)
- Clients using SSL
- Trust issues (re-signing code)
- Scalability (very large corporations)