Practical Considerations for Deploying a Java Active Networking Platform

Robert F. Jaeger
University of Maryland
Department of Computer Science
rfj@cs.umd.edu

Programmable Network Devices

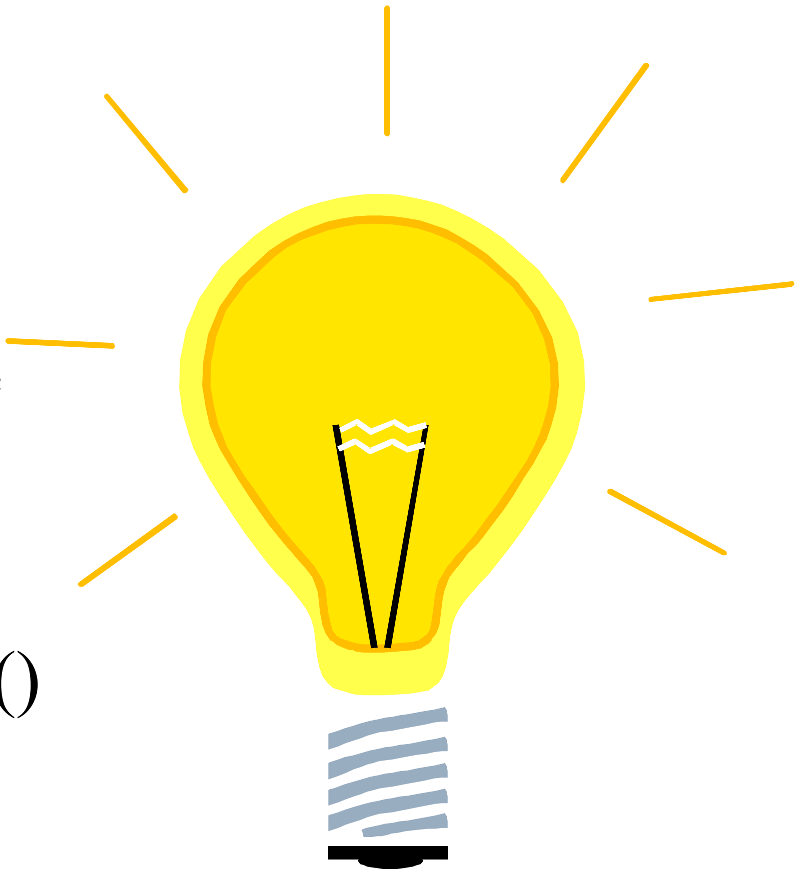
Openly Programmable devices enable new types of intelligence on the network

Agenda

- Local Computation
- New types of applications
- Programmable and Active Networks
- Network Services Architecture
- Issues
- Summary

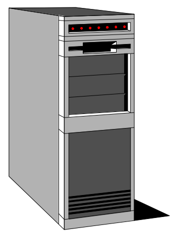
Changing the Rules of the Game

- Move **Turing Machine** onto device
  - Run non-vendor/non-bundled applications on network device



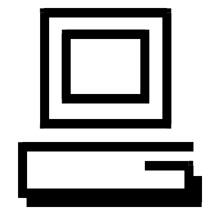
```
while (true) {
  doLocalProcessingOnDevice()
}
```

Non-vendor/Non-bundled Applications



Web Server

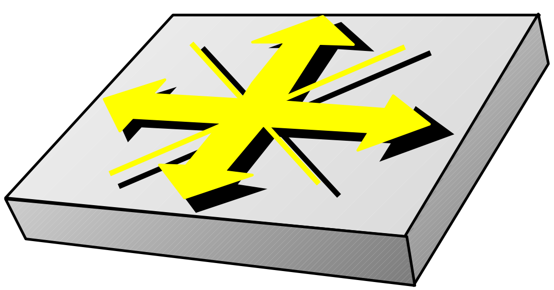
Applet



Web Browser

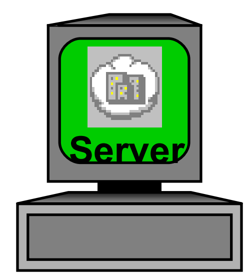
The JVM is in the Browser

Download applications for local processing



non-bundled application

Reversed Applets



The JVM is in the Device: supports non-bundled apps

The Web Changed Everything

- **Browsers**
  - Introducing JVM to browsers allowed dynamic loading of Java ***Applets*** to end stations
- **Routers**
  - Introducing JVM to routers allows dynamic loading of Java ***Services*** to routers

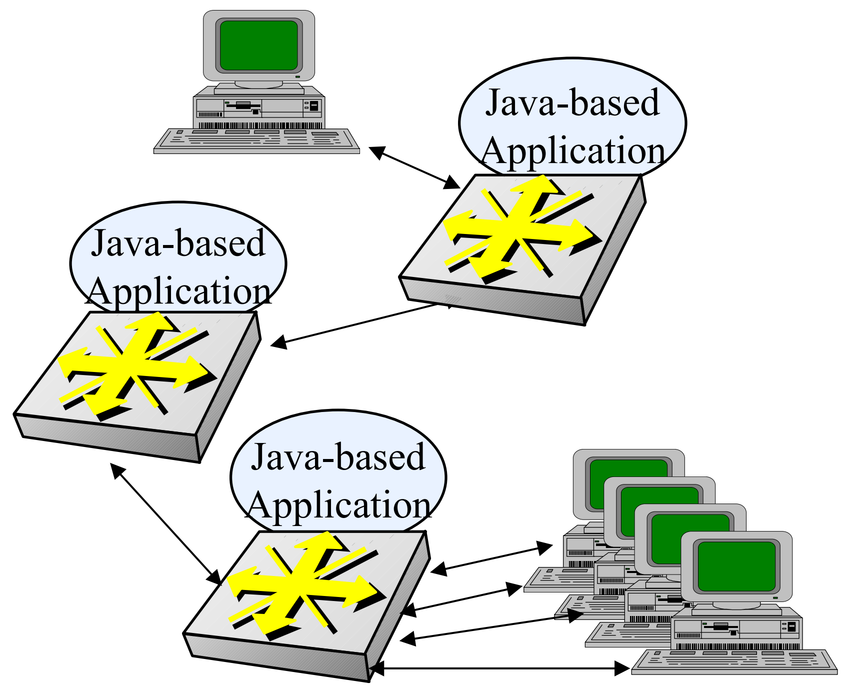
This Capability WILL Change Everything

Architecture to Augment Vendor-Provided Software

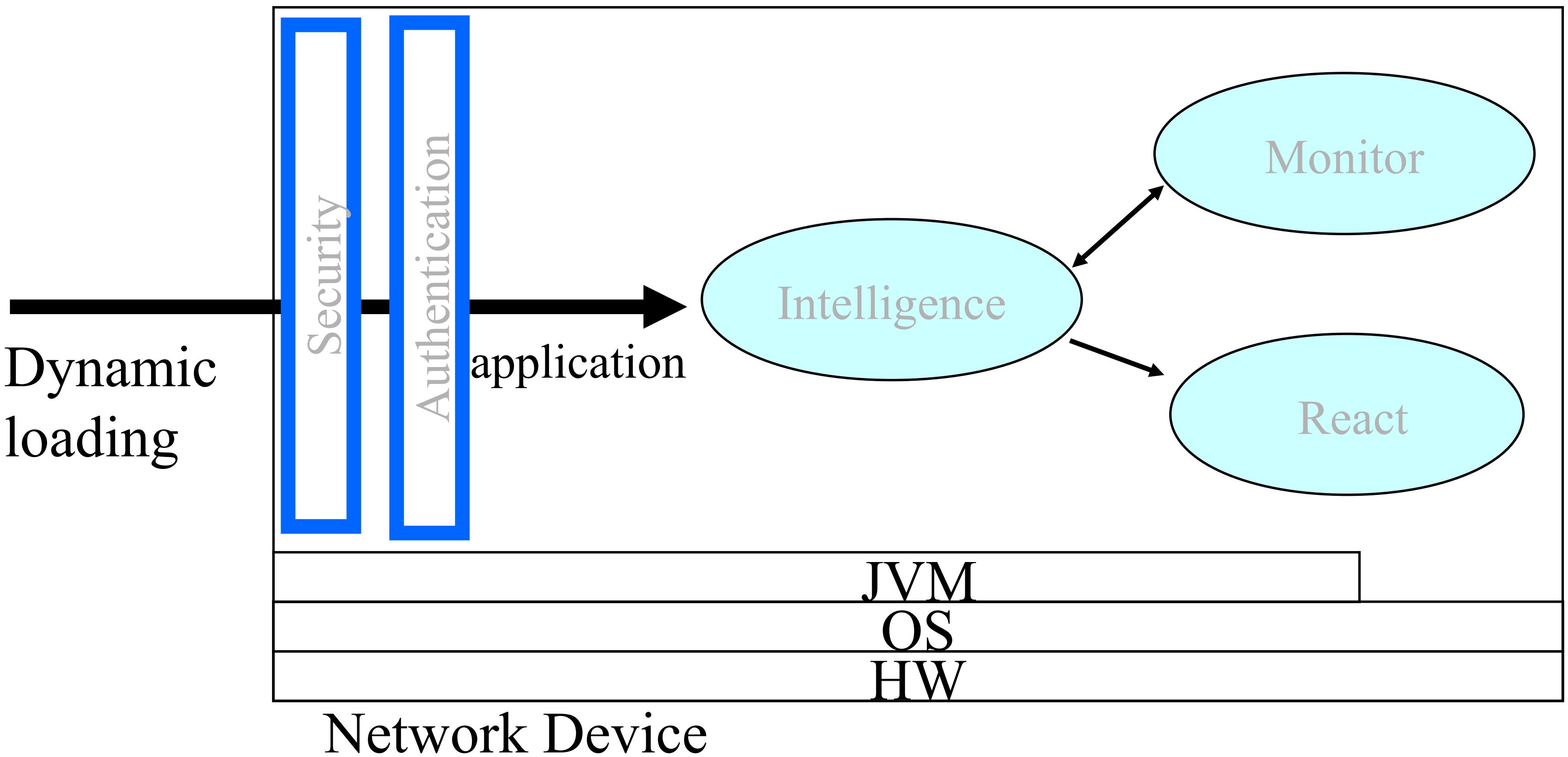
- Supports non-vendor applications
- End-user custom application development
  - Tight interaction between business applications and network devices
    - Domain experts who understand business goals
    - Innovative approaches
  - "Features on Demand"
    - download software services
    - dynamically add new capabilities

Paradigm Shift

- Supports distributed computing applications in which network devices participate
  - router to router
  - server to router



Example: Downloading Intelligence



Device-based Intelligence

- Static-vs-Dynamic Agents
  - Static
    - SNMP set/get mechanisms
    - Telnet, User Interfaces (cli, web, etc…)
  - Dynamic closed-loop interaction on nodes
    - capable of dealing with new and difficult situations
    - autonomous and rational properties
    - system monitoring & modification
    - report status and trends

Agenda

- Local Computation
- New types of Applications
- Programmable and Active Networks
- Network Services Architecture
- Issues
- Summary

New Types of Applications

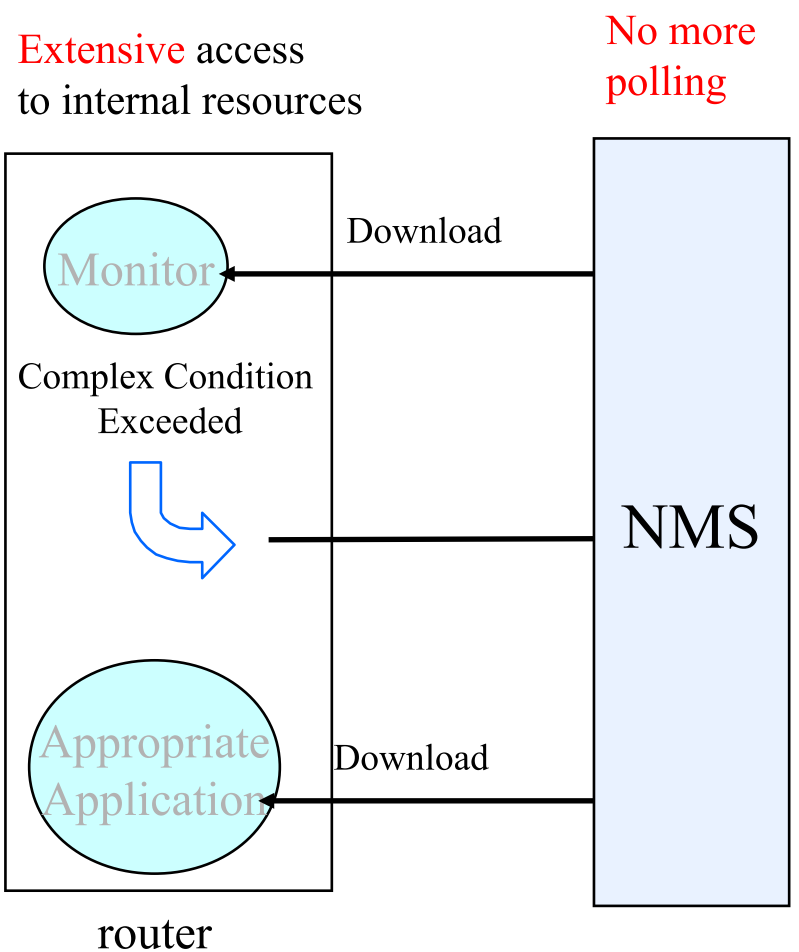
- Mobile Agents
- Local Intelligence for NMS
- Application layer collaboration among routers
- Distributed computing involving network devices and servers
- E-commerce

Mobile Agents

- Intrusion Detection - Hacker Chaser
- Traceroute for Layer 2
- Mobile Connectivity Mapper

Local Intelligence for NMS: Diagnostic Agents

- Download Intelligent Agent monitor from NMS to the device.
- Wait for threshold.
  - Might be complex conditions
  - Trend analysis
- Send “condition exceeded” event to NMS.
- Automatic download appropriate application
- Application takes action.



Application Layer Collaboration Among Routers and Servers

- Multicast Caching
- Web Caching
- Server farm load balancing
  - server state monitored
  - rerouting based on congestion/load
- Auctioning Applications

E-Commerce Example

Matching Customers with Suppliers
- comparing price/capability options
- ISP QoS capabilities & availability

Business logic based operation changes
- Resize forwarding queues
- Modify congestion control algorithm
- Adjust Packet Scheduling
- Change routing table

Agenda

- Local Computation
- New types of applications
- Programmable and Active Networks
- Architecture
- Issues
- Summary

Programmable Networks

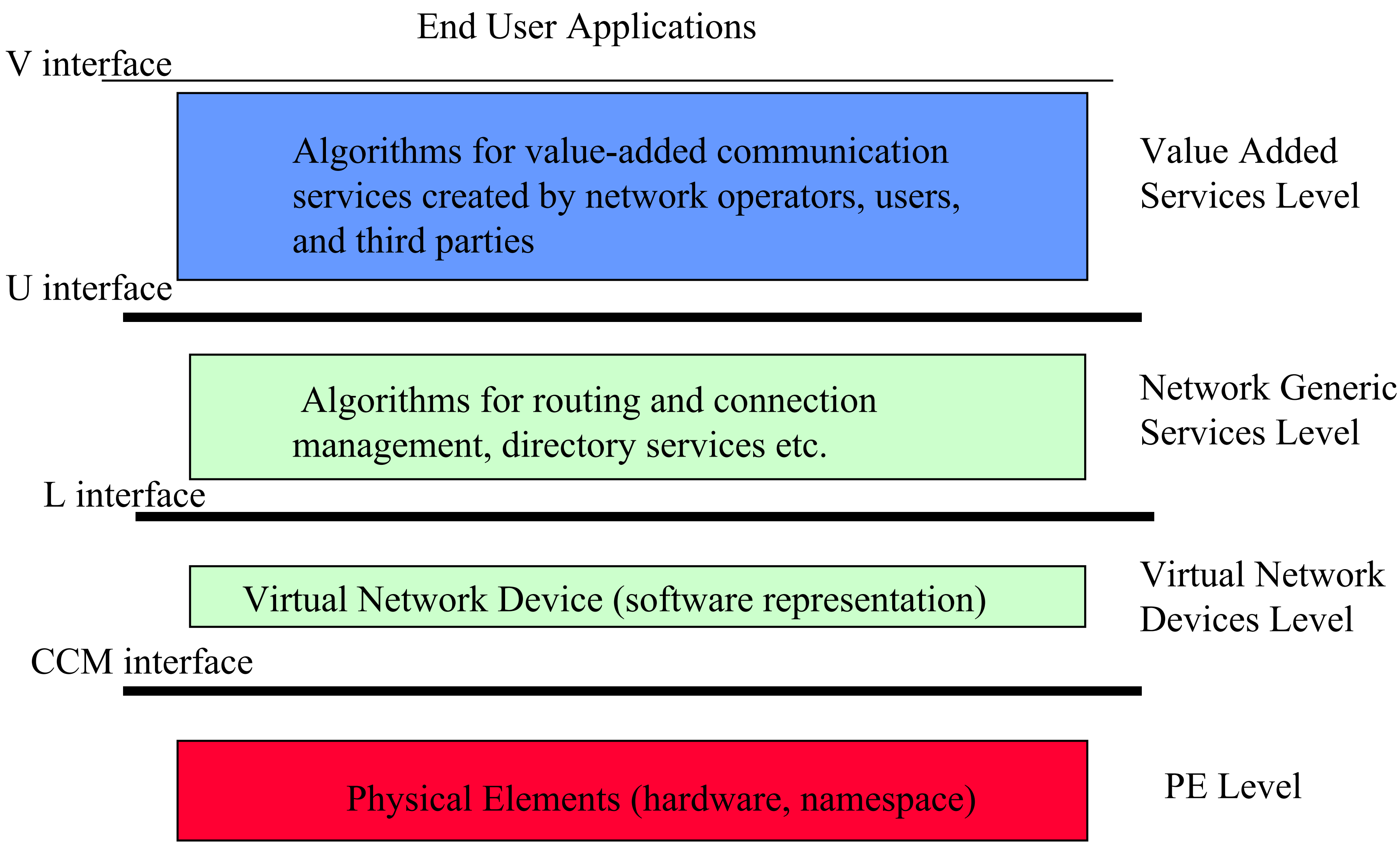
- IEEE P1520 Working Group
- Benefits of Standard Network APIs
  - separation of service business/vendor business
    - ISP resources visible for controlled modification
    - 3rd party signaling vendors
  - faster standardization
  - extensibility
  - richer semantics
    - e.g. dynamic binding

http://www.ieee-pin.org

Programmable Networks

- IETF - vs- IEEE P1520
  - IEFT - Internet standardized algorithms and protocol semantics
  - P1520 standardized programming interfaces
- MPLS Example
  - Create IDL that captures the programmability requirements of IP routers/switches from MPLS algorithm perspective
  - Common interface definitions would be used by RSVP, LDP, or traffic engineering

The P1520 Reference Model



Active Networking

"The active network provides a platform on which network services can be experimented with, developed, and deployed"

http://www.darpa.mil/ito/research/anets/index.html

Active Network Objectives

- Minimize amount of global agreement
  - Do not require global agreement to support dynamic modification of the network
- Support fast-path processing optimization
- Scale to very large global active networks
- Provide mechanisms to ensure security and robustness of nodes and of the network
- Provide mechanisms to support different QoS/CoS

Active Network Architecture

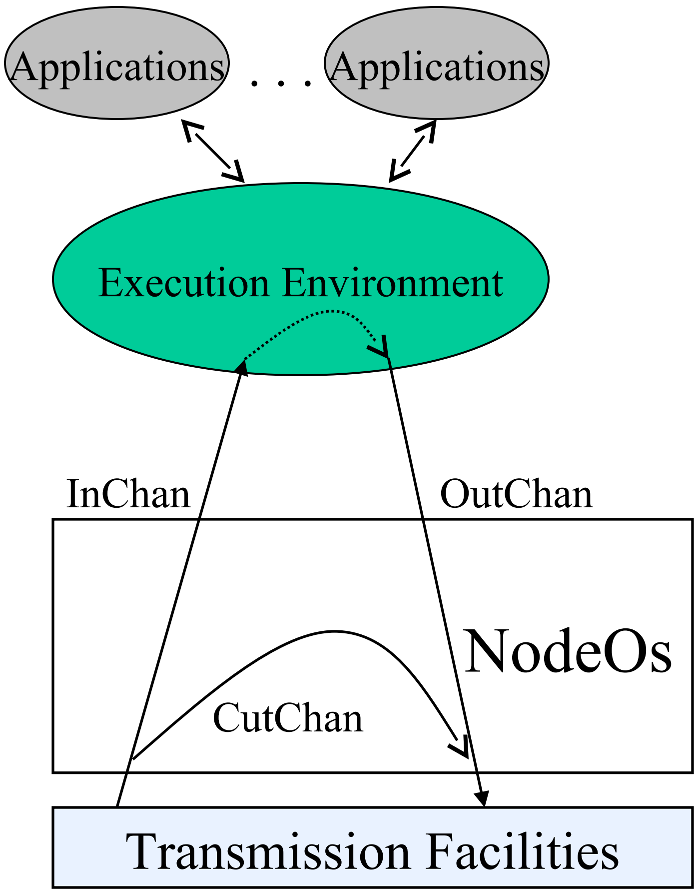
- **NodeOS** - manages resources for the node
- **Execution Environment** -
  - provides an API to applications   or
  - a shell interface through which end-to-end nework services can be accessed.
- **Active Applications -** implementation of network services which utilize the local computation and access to router resources.

Node Operating System

- Latest Specification (June 15, 1999)
- Abstractions
  – Channels
  – Memory Pools
  – Thread Pools
  – Files
  – Flows

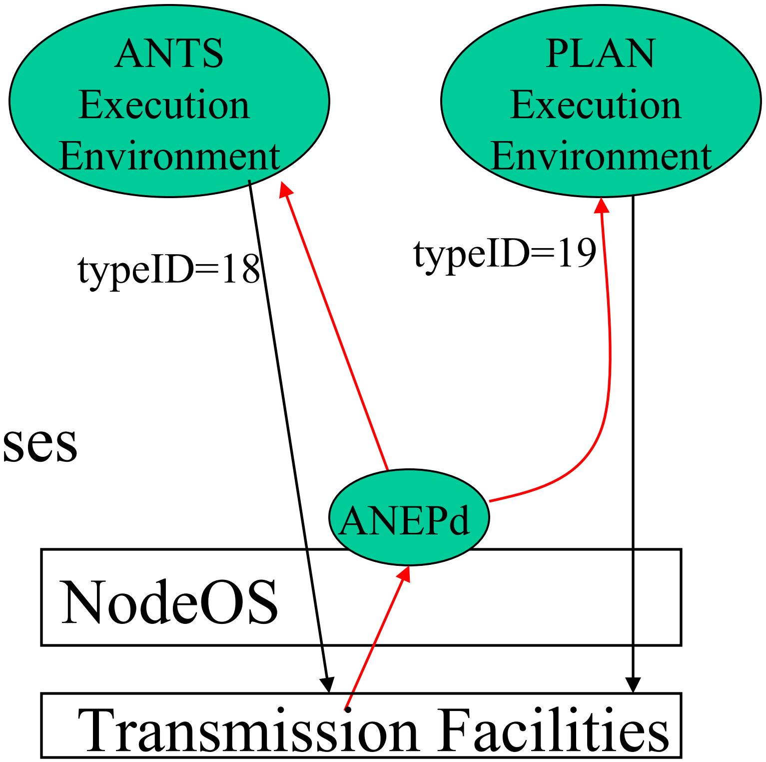
NodeOS - Channels

- Flows create channels to send, receive and forward packets
  – InChan - receives packet from network to EE
  – OutChan - puts packets onto the network from EE
  – CutChan - bypasses the Execution Environment
- Bandwidth Limitation
- Buffer Pool -- queued pkts



Active Network Encapsulation Protocol

- Routes AN packets to EEs
- ANEP_PORT = udp 3322
- TypeID identifies EE
- Tag Length Values (TLVs)
  - specify source/dest IP addresses
  - port numbers
  - Payload



NodeOS - Memory Pools

- Combines memory for one or more flows
- Shared by threads within flows
- mmap-style interface to page allocation
- flow in which thread runs charged for resource
- EE notified when flow exceeds limits
- Flow (and associated threads) terminated upon violation

NodeOS - Thread Pools

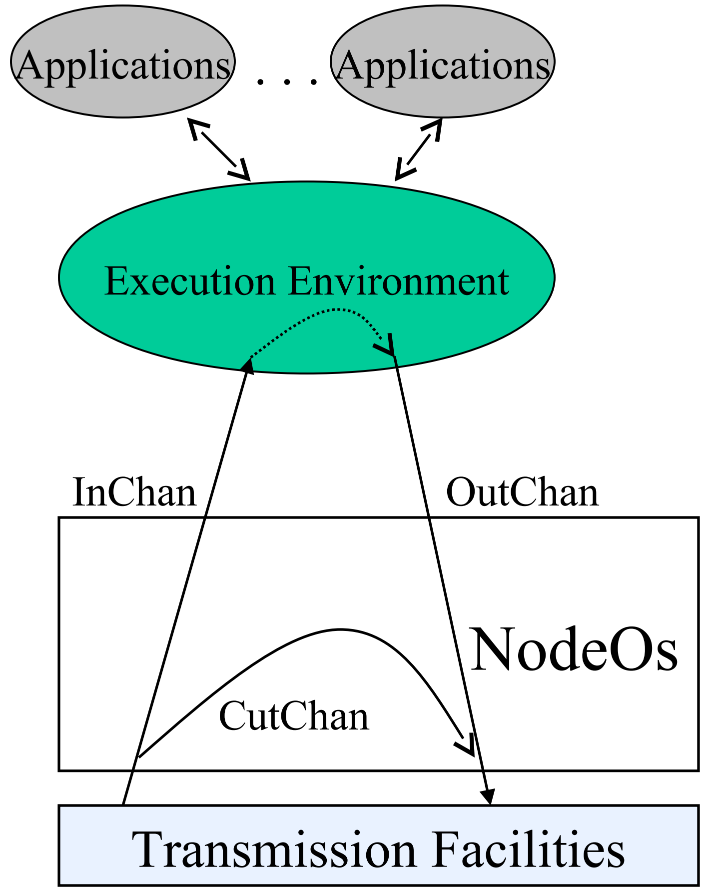
- Computational Abstraction
  - Number of threads in Pool
  - Scheduler to be used (round robin, …)
  - Max execution time between yields
  - Per thread stack size
- No explicit operation for creation/ termination -- activated by events
- Termination of flow if thread misbehaves

NodeOS - File

- Not Manditory
- Provides Persistent Storage
- EE specific view of filesystem
  - via namespace(AN/ANTS; AN/PANTS)
- Shared Memory for inter-EE communication

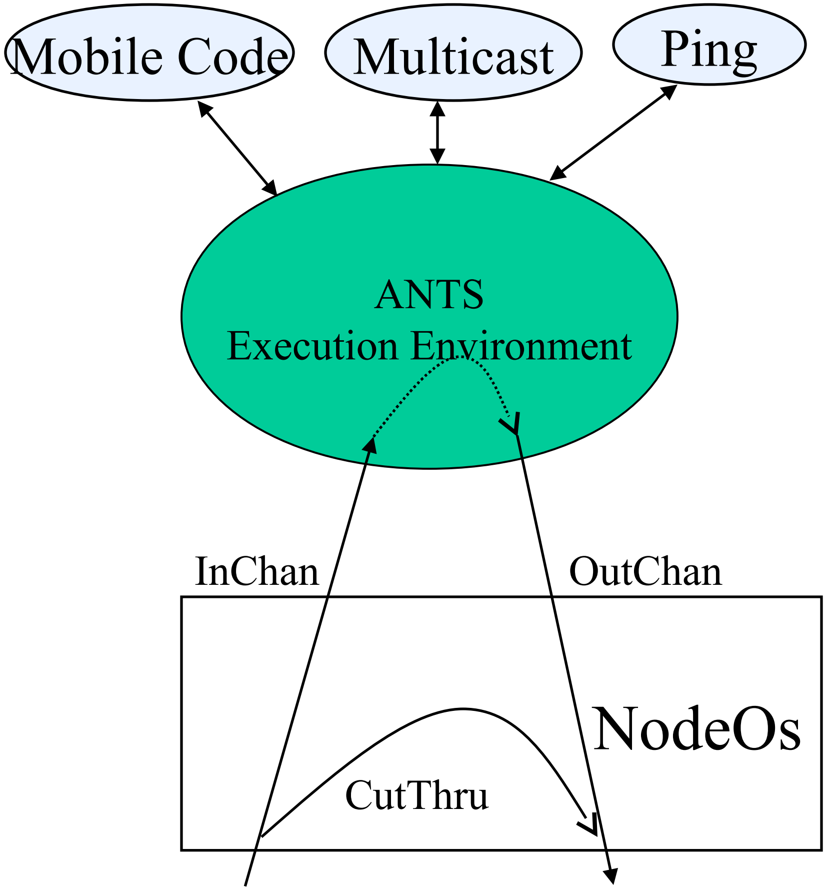
NodeOS - Flows

- Primary abstraction for accounting, admission control, and scheduling
- Flow consists of:
  - Channels
  - Memory
  - Threads
- Flow can be
  - Execution Environments
  - Active Applications



ANTS Execution Environment

- Facilitates deploying new protocols and services in network
- Toolkit for implementing an active network
  - Active Nodes
  - Network Nodes



ANTS Execution Environment

- Capsules are the unit of transfer for data and code
  - source & destination addresses
  - previous node address
  - resource limits
  - encoding and evaluation methods methods
  - Protocol/Group/Method ID access methods
- Data Capsule
  - source & destination port numbers
  - identifies active application

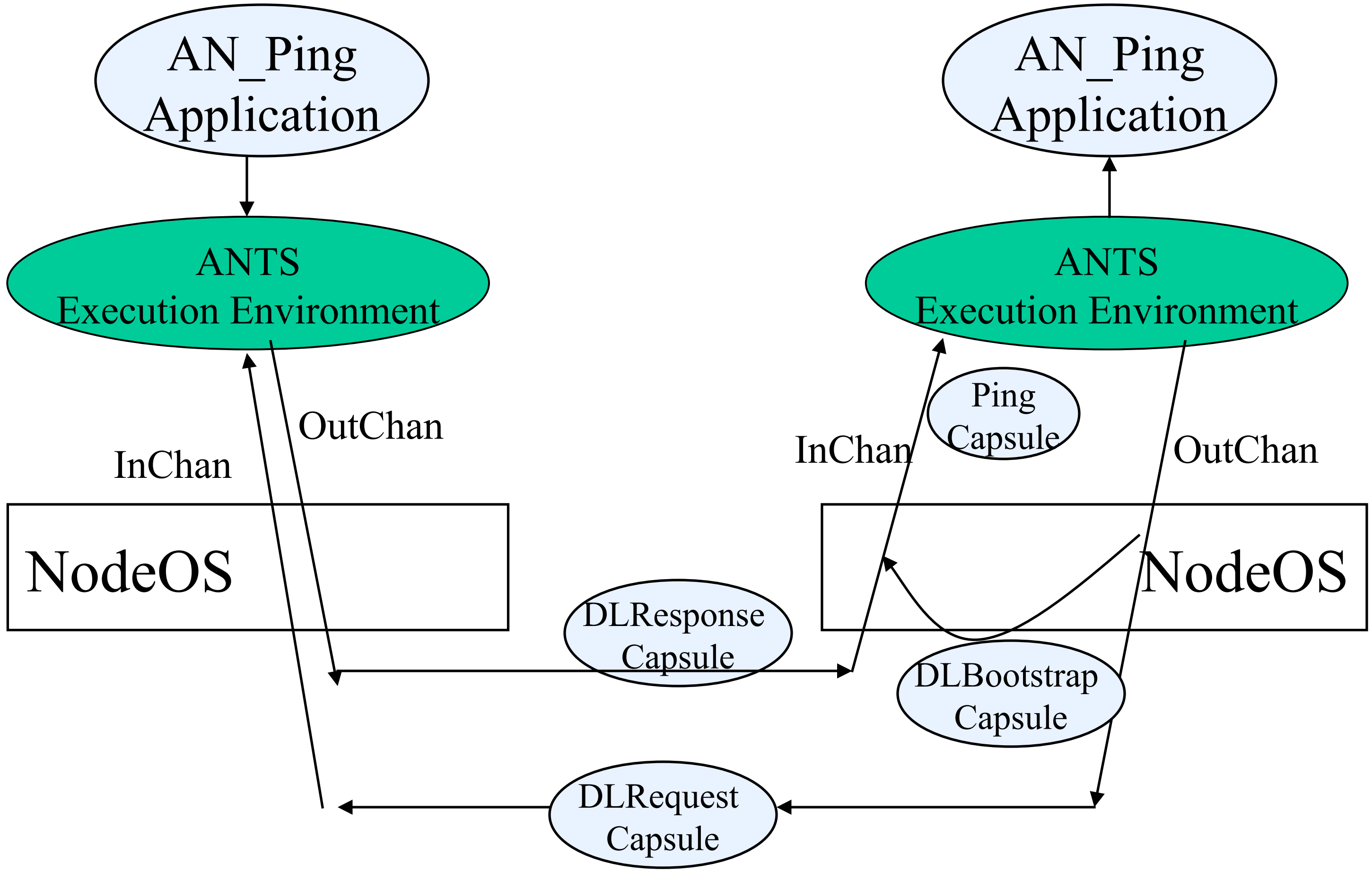
ANTS Code Distribution

- “Node” object is core of Runtime System
  - UDP Channels
  - Methods to Send/Receive Capsules
  - Supports numbers applications identified by port number
- Consists of Built-in protocols
- Accepts registration of new protocol
  - capsule code stored in code cache
  - signature (hash) computed for code

ANTS Code Distribution

- Allows Definition of additional protocols
  - Protocol
  - Code Group (transitive closure of calls)
- Dynamic Code Distribution via Capsules
  - Capsule arrives and node can’t evaluate it
    - protocol not on active node
    - must request packet from previous active node
  - DLBootstrap Capsule
  - DLRequest Capsule
  - DLResponse Capsule

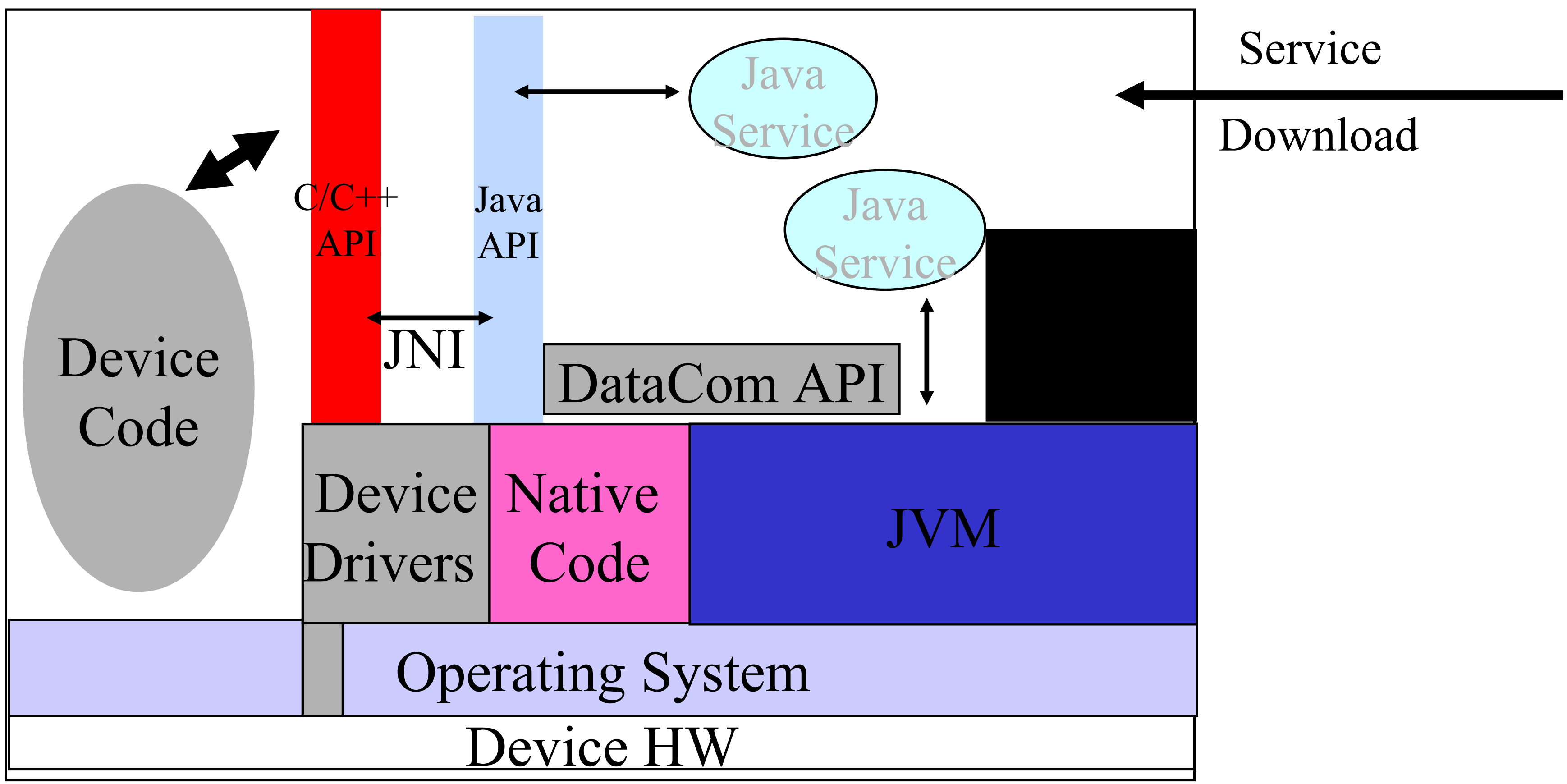
ANTS Execution Environment



Agenda

- Local Computation
- New types of applications
- Programmable and Active Networks
- Architecture
- Issues
- Summary

Open Device Architecture

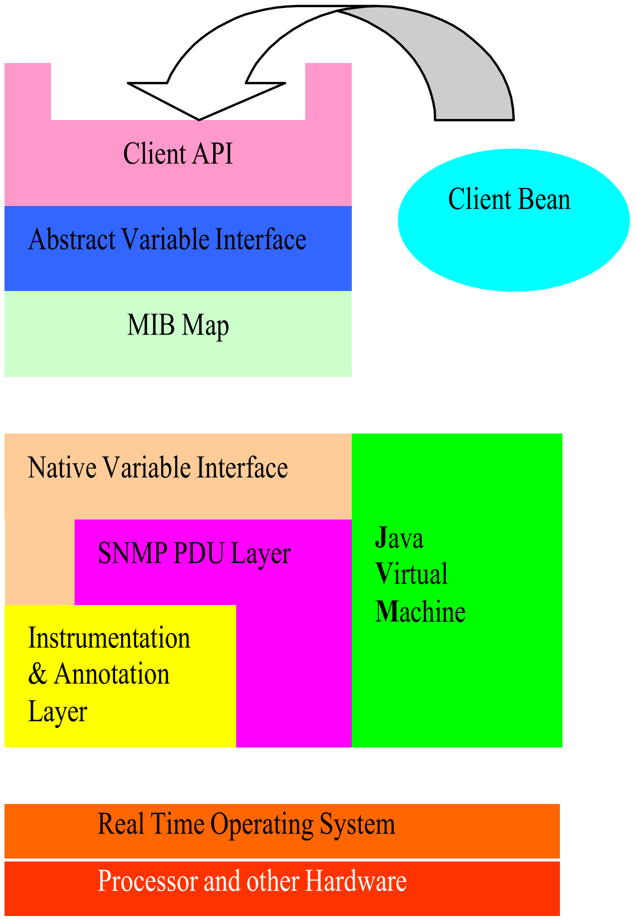


SNMP API for Network Mgmt

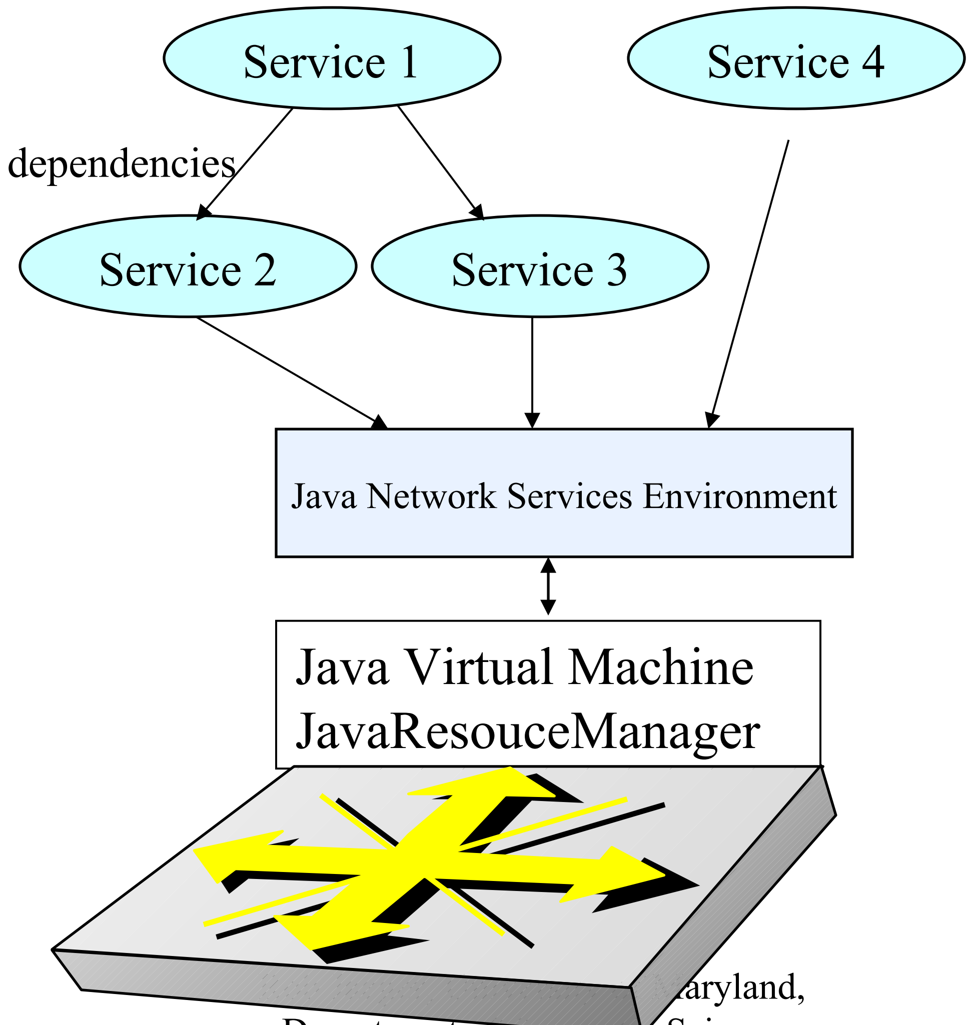
- API is generated automatically
- Device-based monitoring
  - Query MIB
  - Identify trends
- Initiate action locally
  - Report trends and/or significant events
  - Download problem specific diagnostic code
  - Take corrective action

MIB API Example

- API uses a MIB Map to dispatch requests to variable access routines
- Different parts of the MIB tree can be serviced by different mechanisms
    - An ad hoc interface to the SNMP instrumentation layer
    - A generic SNMP loopback



Java Network Services Environment



Our Prototype Java Environment

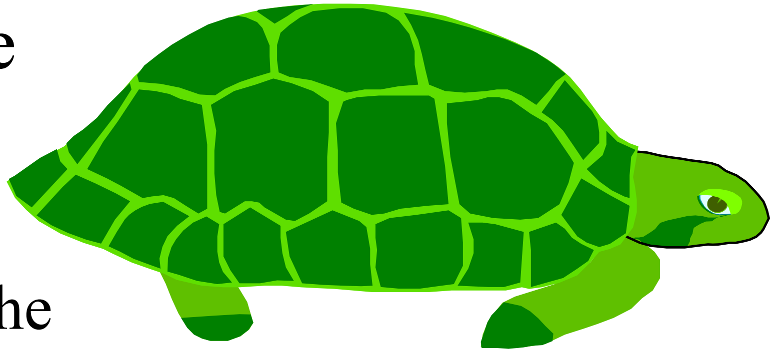
- Present RTOS with single unified task that includes:
  - Java VM (JVM)
  - Java Resource Manager (JRM)
    - thread scheduling
    - manages CPU utilization
      - JVM time-slice is managed by the JRM preemptive thread scheduler
    - internal memory manager
    - garbage collection with priority based on available memory

Why Java

- Dynamic class loading
- Reuse security mechanisms
  - Byte-code Verifier
  - Security Manager
  - Class Loader
- System stability
  - Constrain applications to the Java VMs
  - Prohibit native code applications
- Extensible, portable, & distributable services

But Java is sloooWWWW

- Not appropriate in the fast-path data forwarding plane
  - forwarding is done by ASICs
  - packet processing not affected
- Java applications run on the CPU
  - Packets destined for Java application are pushed into the control plane



Agenda

- Openness
- Local Computation
- New type of applications
- Programmable and Active Networks
- Architecture
- Issues & Questions
- Summary

Architecture Issues

Approach 1: Native Threads

- One JVM per principle
- One RTOS task per JVM
- Non-interference between Java applications
- Difficult thread-to-thread communication and sharing of data between threads
- Creates a dependency on underlying RTOS
- Multiple JVM instances consume resources

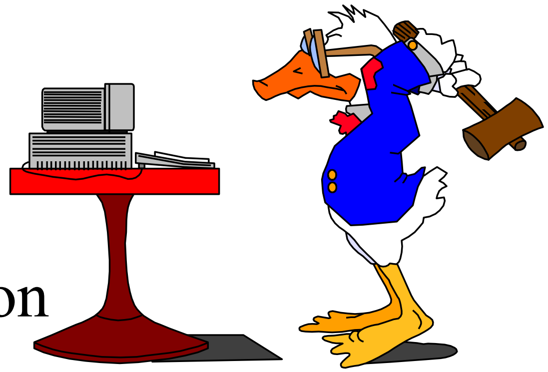
Architecture Issues

Approach 2: Single JVM - Green Threads

- Present one unified task to the RTOS
- JVM manages CPU & memory resources between competing threads;
- Propagation of component failure
- Requires modifications to the JVM
- Binding of resources to the JVM

Security Issues

- Old model: Cannot isolate core router functions
  - Dangerous Pointers (C/C++)
    - Can touch sensitive memory location
  - Risk: Memory allocations and Free
    - Allocation without freeing (leaks)
    - Free without allocation (core dump !!!!)
- Limited security in SNMP

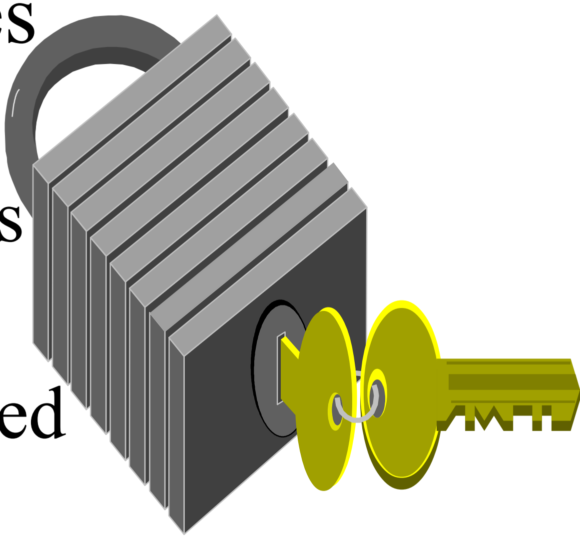


Security and Stability

- secure download of Java Applications
- safe execution environment
  - insulate core router applications from dynamically loaded applications
  - protect dynamically loaded services from one another

Strong Security in the new model

- The new concept is to securely download 3rd party code to network devices
  - Digital Signature
  - Administratively Certified Services
  - Access only to the published API
  - Verifier - only correct code is loaded
  - Class loader access list
  - No pointers that can do harm
  - No access outside the JVM space
  - JVM has run time bounds, type, and execution checking



Language Based Protection

- Type Safety
  - Reference to Objects, not random memory
  - Inappropriate accesses to memory not allowed
    - Restricts what operations code can perform on what memory locations
    - operations on objects must be valid for that object
  - dynamic access control (via reference)
  - static access control (via public, private)

Access Control [6]

```
class A {

  private  int i;
  public   int j;
  public static void method1() {
     A a1 = new A();
     A a2 = new A();
     B.method2(a1); }
}
class B
   public static void method2 (A arg) {
       arg.j++;
       arg.i++;    // illegal
   }
```

- method2 has access to public j

  but not to private i

- method2 cannot forge a reference to a2 given the a1 reference

How to Access a Class

1. Must get Class Object
   a. Class is in classpath (not secure on net)
   b. Class reference is available (visible)
   c. Have a ClassLoader Object to load Class
2. Reference to Object
3. Access control (public)

   For static methods, need just 1 and 3

Class Loaders

- Load new classes into the JVM at runtime
  - fetches code from URL or file
  - submits to JVM for verification
  - integrates code into JVM for execution
  - references to other classes causes additional class loader invocations
- Enforces protection - expose visibility and hiding
  - classes see classes loaded by same classloader
  - can use class loaders to expose classes

NameSpaces

- A namespace is
  - a set of unique names of classes loaded by a Class Loader and the binding of each name to a specific class object
  - variables, methods, & type names are all different instances in different domains

Cross Domain Communication

- Desire that Protection Domains share classes and NOT require same Class Loader
- How do we achieve this?
  - Runtime System to provide communication between components.
  - Java Network Service Environment
  - What is the policy?

Building Protection Domains

- Given multiple namespaces
  - Could use Object references for cross-domain communication:

```
class FileSystem {
    private int accessRights
    private Directory rootDirectory
    public File open(String fileName)          [6]
}
```

  - Enforce protection policies per client
  - Problems result

Protection Domains - Revocation

- Access to an object reference cannot be revoked
- Wrap object with revocable object that is a delegator to real object
  - all methods wrapped
- Programmer may forget to wrap objects referenced by wrapped object (tracking problem)

Protection Domains: Revocation[6]

```
class A {  public  int  method1(int a1, int a2); }


class AWrapper {
  private A a;
  private boolean revoked;
  public int method1(int a1, int a2) {
     if (!revoked) return a.meth1(a1, a2) ;
     else throw new RevokedException;
  }
  public void revoke() {revoked=true;}
  public AWrapper (A realA) {
     a = realA; revoked = false; }
}
```

Protection Domains: Inter-domain dependencies

- Sharing Object references between domains
- Mutable shared objects can be changed
- Malious attack:
  - pass byte array w/ legal bytecode to classloader
  - once verified, overwrite with illegal bytecode
- Should copy bytecode to classloader, not pass reference

Protection Domains: Termination

- Upon domain termination:
  - should all references obtained be released?
    - two Strings in different domains may reference the same underlying byte array
  - should object be kept alive if referenced by other domains?
    - clients could hold onto references to objects of a dead server
  - GC frees objects when NO more references!!

Protection Domains: Threads

- Method invocation for cross domain calls both execute in same thread
  - caller blocks until callee returns
    - how does caller back out gracefully?
  - untrusted domain calls stop() or suspend after calling trusted method --
    - state left unstable and blocked
  - untrusted callee can block caller that may be in critical section

Protection Domains: Accounting

- How do you account for resources obtained by a domain?
  - CPU cycles
  - Memory pages
  - Bandwidth on a channel

J Kernel Safety [6]

- Precise definition of protection domains
  - local object
  - non-local shared objects (capability objects)
- Define communication channels between protection domains
- Support revocation of capabilities
- Clean termination semantics

J-Kernel Class Loaders

- Each ClassLoader defines a namespace
  - must manage & secure namespace
  - creates stub code at run-time for cross domain communication -- use local RMI calls
    - simulate thread switching for safe method calls
    - contains a revoke method to set handle to null
  - substitutes “safe” versions of standard classes
    - e.g. file system access

J Kernel Concepts

- Capabilities:
  - handles to resources in other domains
  - client throws an exception
- Domain:
  - each domain has a namespace and threads under its control
  - shared classes
  - capabilities access is revoked upon termination

J Kernel Concepts

- Cross domain calls:
  - Invoke calls to “capability” methods
    - relies upon Java interface classes
    - extend remote (stub creation and marshalling code)
  - special calling convention
    - non capability objects are copied
    - capability objects are passed

Observations

- Provides high degree of safety for cross-domain communication
- Expensive in terms of time
  - thread switching (simulated)
  - method invocation through stub
  - copying of non-capabilities

Questions

- How do you insulate core router functionality?
- How do you securely download code?
- How do you do resource accounting?
- How do you assure resource safety?
  - fair share or priority share quotas?
    - CPU
    - Memory
    - Bandwidth

Questions

- How do you protect services from one another (trusted -vs- untrusted)?
  - stable state for critical sections
    - caller dies/is killed while trusted in critical section
  - enforce return from untrusted method
  - reject forbidden actions
- Native or Green Threads?

Agenda

- Openness
- Local Computation
- New type of applications
- Programmable and Active Networks
- Architecture
- Issues
- Summary

Summary

– Turing Machine on network devices
– *dynamic* agents vs. *static* agents
– dynamic loading
– strong security through JVM
– safety among shared components via Java Network Services Environment

Enabling Technology for the Revolution

References

[1] P.Bernadat, D. Lambright, and F. Travostino, “Towards a Resource-safe Java for Service-Guarantees in Uncooperative Environments,” IEEE Symposium on Programming Languages for Real-time Industrial Applications (PLRTIA) ‘98, Madrid, Spain, Dec. ‘98.

[2] Active Networking Node OS Working Group, NodeOS Interface Specification", June 15, 1999

[3] Active Networks Working Group, "Architectural Framework for Active Networks Version 0.9", August 31, 1999

[4] T. Lavian, R. Jaeger, "Open Programmable Architecture for Java-enable Network Devices", Stanford Hot Interconnects, August 1999.

[5] D. Wetherall et al. ANTS: A Toolkit for Building andDynamically Deploying Network Protocols. OPENARACH'98

[6] C. Hawblitzel, C. Chang, G. Czajkowski, D. Hu, T. von Eicken, “Implementing Multiple Protection Domains in Java”, 1998 USENIX Annual Technical Conference, New Orleans, LA, June 1998

[7] R. Jaeger, T. Lavian, R. Duncan, “Open Programmable Architecture for Java-enabled Network Devices”, To be presented at LANMAN ‘99, Sydney, Australia, November 1999