Active Network Support Services Demonstration

Columbia University, University of California Berkeley, University of California Los Angeles, University of Utah

December 6, 2000
Outline

• Introduction
• Description of the demo
• Nestor (Columbia)
• Panda (UCLA)
• Janos (University of Utah)
• Ninja (UC Berkeley)
• Conclusion
Introduction

• What are active network support services?
  – Node operating systems
  – Active network management
  – Middleware to make active networks more usable
  – General clustering services applied to active networks

• Generally, they assist core active network technologies (like EEs and active applications) in doing their jobs
Two Important Characteristics of These Services

1. Intended for use by many applications
   - Requiring high degree of generality
   - And interfaces usable by wide range of applications

2. Should interoperate naturally
The Demonstration

• Two goals
  1. Show the value of the support services
  2. Demonstrate interoperation

• Basic strategy
  – Show several support services working together
  – Adding value to an application using active networks
What Will We Be Showing?

- A videoconferencing application
- Built from basic video and audio streams
- Active services allow it to operate in difficult conditions
  - Poor network links
  - Competing traffic and applications
  - Failures
- Goal is to show obvious improvement
Some Details

• Two cameras streaming live video/audio
  – In WaveVideo wavelet encoding
  – Not directly using active networks

• Crossing a wireless link
  – Resulting in unacceptable video quality

• Facing competing traffic at one node

• Eventually, one of the service nodes fails
The Demo Situation

What if a key node fails?

Competing traffic

Wireless link with insufficient bandwidth
Solving the Demo’s Problems

- Wireless link with insufficient bandwidth
- Competing traffic
- Reserve resources with Janos node OS
- Monitor conditions with Nestor
- Use Actiware VANS and Panda active network adaptations
- Use a Ninja cluster to achieve fault tolerance
- Competing traffic
- Wireless link with insufficient bandwidth

Monitor conditions with Nestor

Use Actiware VANS and Panda active network adaptations

Reserve resources with Janos node OS

Use a Ninja cluster to achieve fault tolerance

Competing traffic

Wireless link with insufficient bandwidth
What Are Nestor and Actiware Doing?

• Nestor observes wireless link characteristics
  – Reports them to Panda, when requested
  – Also displays them in real time on a system management machine

• Actiware sets up virtual active network links for Panda over wireless link
What Is Panda Doing?

- Panda intercepts four non-active data streams and makes them active
- Sets up a (simple) plan for adaptation
  - Based on information from Nestor
- Runs adaptors at near end of wireless link
  - Adaptor that drops some wavelet levels
  - Adaptor that gives more bandwidth to speaker
What Is Janos Doing?

- Makes reservations for Panda flows on intermediate node
- In the face of competing:
  - CPU hogs
  - Network hogs
  - Memory hogs
What Is Ninja Doing?

- Ninja runs Panda at one location in cluster mode
- When one cluster node running Panda fails, Ninja fails over to another node
  - In around one second
On With the Demo!

- Two live video/audio feeds are being sent through the network just described
- Note that the output sucks
- Let’s get started fixing it!
The Demo Setup

Wireless link with insufficient bandwidth

Competing traffic
Actiware and Nestor
Panda

- Middleware to bring benefits of active networks to legacy programs and other AN-unaware programs
- Panda applies active network adaptations to selected non-active streams
Adaptation of Unaware Applications

- Many existing applications don’t use active networks
- Many future applications won’t, either
- But many kinds of data streams are automatically recognizable
  - And adaptable using active networks
How Does Panda Help Unaware Applications?

- Intercept data streams at sending node
- Choose streams that Panda can handle
- Convert packets in stream to ANTS packets
- Deploy adaptors to do something helpful
- At destination, strip off ANTS stuff and deliver non-active packets
Adaptation Composition

• In complex networks, one adaptation at one place is often insufficient
• Combining multiple adaptations must be done carefully
• Requires planning to ensure adapter compatibility
  – And proper overall behavior
Panda Planning

- Two types of planning currently supported:
  - Planning at sending node
    - Sending node specifies which adapters and where
  - Hop-by-hop planning
    - Each node decides on local adapters
    - Using knowledge of previously deployed adapters
- Heuristics used in demo very primitive
- More sophisticated planning is partially implemented
What Panda Does in the Demo

- Panda captures both video and both audio streams
- Converts them to ANTS active format
- Examines Nestor-supplied information about wireless link conditions
- Chooses plan to
  - filter wavelet encoding
  - use Actiware VANs to reserve bandwidth
  - give preferential treatment to speaker’s streams
- Deploys and runs necessary adaptor
- Converts back to non-active form at destination
Team 3: Demo 2000
Janos Project

University of Utah
Flux Research Group
Java Active Network OS

- Java-oriented active network operating system
  - From AAs all the way down to the wires [JSAC 2001]
- Provides standard OS facilities
  - Separation
  - Resource control
  - Termination
- … but in a Java Virtual Machine
Java Active Network OS

- Abstractions from operating systems [HotOS’99]
  - User/kernel boundary, process model
- Mechanisms from garbage collection:
  - Distributed GC, write barriers
- Key issue: controlled sharing
  - Packet buffers
- Based on KaffeOS [Back et al, OSDI 2000]
- Comprehensive resource control
  - Physical memory, CPU, outgoing network bandwidth
Janos in the Demo

- Demonstration of Janos support for resource controls over Java code
  - CPU
  - Network bandwidth
  - Memory

- Demonstration of Java code in low-level networking
Janos in the Demo

- Janos node connects video source network and video display network
Janos in the Demo

- IPFwd application forwards packets
- Hog applications waste resources
- All apps are written to Janos Java NodeOS API
- Each application runs in its own Java process
  - Separate GC, Heap, namespace, CPU, threads, etc.
Janos Setup

Janos Java NodeOS

JavaVM

Moab NodeOS

Utah OSKit

PllI 600 / 3x 100Mbs / 128MB
Janos Setup

- **IP Fwd**: Simple IP routing of two video streams to display network
- **Net Hog**: Network bandwidth abuser
  - Consumes 90% of output link
- **Mem Hog**: Infinite memory waster
  - Java GC cleans up, restricted to 2MB
- **CPU Hog**: Endless CPU consumer
  - 200 threads in infinite loops
IP Forwarder

- Validates header checksum, decrements TTL, picks OutChan
- Written in Java:

  ```java
  if (bufHandle.computeChecksum(0, IPHeader.HEADER_LENGTH_NO_OPTIONS) != 0)
      throw new Error("Bad checksum...");
  if (!IPHeader.consumeTTL(bufHandle, 0))
      throw new Error("No time to live...");
  routeEntry = this.lookupRoute(this.iface,
      IPHeader.getDestination(bufHandle, payloadOffset));
  if ((outChan = routeEntry.getChannel()) != null)
      oc.send(bufHandle);
  ```

- Zero-copy buffer access
(Net|CPU|Mem) Hog

- Runs in own Janos domain
- Efficiently wastes just one resource
  - Net hog gets significant CPU allocation
- Each written in Java

Stats & Control

- Talks to GUI on separate NIC
Performance

• More than enough for the demo
  – 500 pps, ~500 bytes per packet
• IP Forwarder handles almost 18Kpps
  – About 40% of the C version
• Ping across forwarder in less than 1ms
Demonstration

• Janos manages CPU, memory, network usage of each domain.

• Parameters are setup such that
  – Forwarder flow gets more than enough
  – “X” hog gets a small share of “X”

• “Disabled” scheduler is simple round-robin over quantum (time slice or “packet send”)

• Memory scheduler cannot be disabled
  – Cannot revoke allocated pages
Future Work

• Performance
  – Interrupt -> Polling model for rx
  – JanosVM optimizations
  – (JIT & GC optimizations, etc.)

• Build applications to our model
  – Validate the sharing/separation

• Improve resource schedulers
  – Include latency requirements
Summary

- Janos provides resource guarantees to active network code
- Janos supports Java code for systems
  - Zero-copy buffer access
  - Full NodeOS API available (except Mem)
- Janos provides OS process model for Java applications
Available

http://www.cs.utah.edu/flux/janos/

- **NodeOS in C: Moab**
  - OSKit, Linux, FreeBSD, Solaris
- **NodeOS in Java:**
  - Bindings for: Moab, JDK, (soon) AMP
- **JanosVM**
  - Available soon
- **ANTS**
  - ANTSR available now
  - ANTS 2.0 available soon
Ninja
What Has This Demo Shown?

• Benefits of active network technologies
  – Specifically, of AN service technologies
  – Obvious benefit to a realistic service

• Ability of various active network services to interoperate beneficially

• Application of active networks to non-active applications
Demo Lessons

• Network configuration is a pain
• Wireless is a pain
  – Suggesting it’s actually a good place to look for active network opportunities
• Increasing maturity of components has actually made them useful
• Demo devils are in the details
Why Didn’t We Get Better Frame Rates?

- Multiple passes up and down through ANTS and kernels
- Wireless limitations
- Need better adaptations
  - E.g., packet aggregation
- Java runtime overheads
Credits

• Who actually did the work?
  – Kevin Eustice (UCLA)
  – Ramakrishna Gummadi (UCB)
  – Patrick Tullman (Utah)
  – Alexander Konstantinou and Gong Su (Columbia)