Interoperability of Active Networks
Active Networking...revisited

• Dynamic deployment of programs to process particular packet subflows within the network
  – Data plane – processing data subflows e.g., adaptive recoding of video
  – Control plane – e.g., dynamic installation of flow-specific control/signalling/management algorithms
Standard Architecture

• Active Applications (AAs)
  – Fundamental unit of network programming

• Execution Environments (EEs)
  – Environment for AA execution

• Node Operating System (NodeOS)
Organization of this talk

• Part 1: Introduction to ABone Testbed
• Part 2: NodeOS Interface
• Parting thoughts
Part 1: ABONE Testbed

- Share facilities
- Extend research into realistic network environments
- Enable research collaborations
- Share tool development and software maintenance overhead
- Create a teaching plan
AN Node Architecture

- EE installed in a node by/under management control
- AAs are dynamically deployed and may be transient or persistent
- Model: 1 NodeOS, a few EEs, many AAs in each active node
- Kernel boundary not necessarily fixed
Model

ABone Unix node

Node OS

User Space

Kernel

E.g. ASP EE, ANTS EE

E.g. PLAN, SENCOMM

E.g. CANES
The ABone Architecture

- **Abone:**
  - Wide-area testbed
  - Nodes: diverse distributed OS platforms
  - Links: Internet overlays, plus dedicated links in DARPA’s CAIRN testbed
  - AA/EE/nodeOS architecture

- AN researchers remotely install and manage EEs on locally administered nodes

- Client site accesses central site for registry.
Architecture Topologies

• Creating and using a Virtual EE topology
  – Allocate nodes
  – Build/allocate accounts on these nodes
  – Generate and install configuration files on Nodes
  – Start the EEs on the nodes
  – Monitor topology
  – Launch an AA to run the experiment
ABone Software Components

- Each ABone node has:
  - ABCd (Anetd): Remote EE management daemon
    - Load and launch an EE (Java or C) in a specific file subspace
    - Terminate, restart, configure and monitor EE
  - Netiod: Network I/O daemon
    - Runs as root for kernel filtering
    - Provides uniform interface across Unix platforms
- Client side
  - ABCd (Anetd) client and ABoneShell interface
- Central site
  - Web-based registry program
Interoperability

• A small number of EE are implemented universally among all nodes
• EEs differ across nodes depending on OS
• AAs are implemented to use at least one of these EEs
• This way,
  – A distributed set of OS specific EEs
  – AAs that run using at least one of them
  – Services are hence distributed among a number of nodes and a high utilization of network processing is possible
Part 2: NodeOS Interface

- A multilayer model with 3 layers
  - AA, EE, NodeOS
- Or a multilayer model with 2 layers
  - AA, NodeOS

<table>
<thead>
<tr>
<th>Top</th>
<th>AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Middle</td>
<td>EE</td>
</tr>
<tr>
<td>Lowest</td>
<td>NodeOS  Programming model like POSIX</td>
</tr>
</tbody>
</table>
NodeOS Interface

• Option 1:
  – Multiple languages can be supported
  – Any single language can be ported to many node types

• Option 2:
  – A language runtime system directly on hardware, like in some JavaOS

The working group upholds option 1 and justifies separation into two layers
Architecture Components
Broad Goals for EE and OS

• NodeOS:
  – Multiplex node’s resources among various packet flows

• EE:
  – Offer AA a sufficiently rich, high-level programming environment
Elaboration of Goals

• For NodeOS interface
  – Primary role:
    • Support packet forwarding
  – Secondary role:
    • Arbitrary computations on select packets

• So,
  – Packet processing, accounting for resource usage and admission control are done on a per flow basis
  – Different granularities for packet flows
    • Port-port, host-host, per application
    => Interface cannot prescribe single direction of flow
Elaboration of Goals

• Account for specific capabilities provided to each EE and hence AA
• Packets requiring minimal processing should incur minimal overhead
  – For e.g. Non-active IP
• Ability for EE to extend OS
• Use of standardized facilities for specific requirements
  – POSIX
Abstractions

- 5 Primary abstractions
  - Thread Pool: For computation
  - Memory Pool: For temporary storage
  - Channels: For communication
  - Files: For permanent storage
  - Domain: For aggregating control and scheduling of the other four abstractions
Domain

• Role:
  – Accounting
  – Admission Control
  – Scheduling

• Domain has,
  – Thread pool (translates to CPU cycles)
    • for computation by EE
  – Memory pool (temporary storage)
    • I/O buffers that queue messages on a domain’s channel
  – Channels: inChan and OutChan
Domain

- Encapsulates resources used by both NodeOS and EE for a packet flow
- Could be created like processes: in context of another
  - Hierarchy with NodeOS as root
Domain Hierarchy

• Allows for easy domain termination
• Domain can be terminated
  – By itself
  – Parent domain
• Resources go back to NodeOS
• Parent uses a handler and clears a dying child’s resources

Hierarchy does nothing to each domain’s requirements.
Thread

- Abstraction for computation by a domain
- A thread pool is assigned to each domain during domain creation
  - Information maintained:
    - Max number of threads in a pool
    - Scheduler used
    - and so on
Thread pools

• Threads run end-end
  – Enables packet forwarding
  – Threads cuts across NodeOS into EE as domain does

• Special system threads are available
  – For e.g. Global garbage collection
  – Use POSIX style constructs like conditional signal, waits
Thread Pools

• “Data driven”
  – Threads in the pool are data driven entities with no need for explicit identities
  – Termination, creation are NodeOS specific
  – Only activation is performed for each before assigning to a pool for a domain

• Pools only for accounting purposes
  – Easy reclamation of a terminating domain’s pool of threads
Memory Pools

- Primary abstraction for soft state storage
- Packet buffers
- Holds EE specific state
- Share memory between domains
- Domain to memory pool is a many-one mapping

Enable EE to manage memory themselves
Memory Pool

• Sharing data between domains means
  – Shared data has to be present even after one of the domains terminate
  – All data references should be checked before domain termination

• Pool simplify this process by
  – Not reclaiming shared memory pages
Memory Resources

• Assignment is EE’s job and not NodeOS
  – Performed during domain creation

• Resource consumption is watched by NodeOS and EE are provided a grace time for cleanup over-utilization
  – “Callback” function for each memory pool
    • Invoked by NodeOS to access EE using this pool to clean up
    • Domain is terminated if cleanup is not performed in a timely fashion

Memory pools are implemented independently and not in a hierarchical manner.
Pools are only for accounting purposes.
Their potential for security and protected domains are unexplored.
Channels

- Primary abstraction for communication flow
- For inChan domain specifies
  - Arriving packets
  - Buffer pool to queue packets
  - A function to handle the packets
    - The handler is used to execute this packet in the context of the domain’s thread pool
- For OutChan domain specifies
  - Where packets are to be delivered
  - How much link bandwidth the channel is allowed to consume (guaranteed to get) as present in [17]
Channels

- Cut-through channels
  - Receive and transmit packets
  - EE calls a convenience function with all arguments used by inChan and outChan
Channels

• Packets are resolved by using addressing information and a demux key
• “Anchored”
  – inChan for incoming flow
  – outChan for outgoing flow
• “Cut-through”
  – No packet processing and only forwarding
Revisiting Design Goals

• Domain encapsulates resources for a flow

• Channel specifies
  – Packets belonging to flow and
  – Function applied to flow

• Cut-through channels directly forward packets
Other Abstractions

• File:
  – Loosely follow POSIX 1003.1
  – Hierarchical name space

• Name space:
  – Distinct view of a persistent file system at a directory chosen at configuration time

• Event:
  – Domain can schedule an asynchronous event in the future
Other Abstractions

• Heap:
  – Memory management

• Packet:
  – Encapsulate data that flow thro channel

• Time:
  – EE get time calls