

-
-
-

Active Reservation Protocol (ARP)

Lessons-Learned Session

**Bob Braden,
Bob Lindell, Steve Berson, Ted Faber, Jeff Kann
Graham Phillips, Alberto Cerpa, Ya Xu, Vivek Shenoy**

USC Information Sciences Institute

Active Nets PI Meeting, Orlando, FL
Dec 3, 2001



Active Reservation Protocols (ARP)

Original Objective: Active networking for Internet signaling and other **control-plane protocols**.

- (a) Rapid deployment using dynamic code installation.
- (b) Dynamic functional extension and customization.
- (c) Portable code to reduce protocol standardization.

Major Result:

ASP (Active Signaling Protocol) **EE**
and a variety of **Active Applications** (AAs)

ASP Execution Environment

ASP EE: Java-based environment to host the execution of AAs.

- A user-space OS “kernel” and libraries
- A “system call” interface -- PPI (Protocol Programming Interface)
- Functions:
 - Dynamic loading of AA code
 - AA process model and scheduling
 - Protection and authorization
 - Network I/O
 - Routing Control
 - (Traffic control interface)



Summarize ASP Features

- An AA may be **persistent** or transient.
- **Timer services** & soft-state repositories in all nodes.
- AAs can be loaded from code server or from prev hop.
- General UA model for launching AAs.
- Native IP as well as virtual connectivity modes.
- Inter-AA communication mechanism.

We believe these features are all important for active networking in the control plane.



Important ASP AAs

- Jrsvp -- RSVP resource reservation protocol (90%)
- AFSP -- Active Filter Signaling Protocol [Team 2 Demo 2000]
- Jrip -- RIP routing protocol
(For default forwarding in virtual topology)
- RDP -- Reliable Datagram Protocol [for code loading, ACC]
- ACC -- Active Congestion Control experiment
- PIM -- Protocol-Independent Multicast
(DJW's ANTS code, ported to ASP AA)
- BBN Sencomm Project
- EE monitoring apps (see ABone web page!)
- EE management functions ('traceroute', 'ifconfig', 'netstat',...)
- Pingpong -- student project

ASP EE Functions

See June 01 PI meeting for discussion of the ASP EE functions:

- Dynamic code loading
- AA process model
- Protection and security
- Timing Services
- User application (UA) API

I will now talk about ASP network I/O and netiod.

ASP EE: Network I/O

- ASP EE incorporates a virtual protocol stack (VNET)
 - Simple but extensible implementations of virtual layers 2-4
- PPI implements the **nodeOS channel abstraction**, with extensions.
 - > An AA opens inChannels, outChannels
- ASP EE : network I/O may be either through standard Java library calls or via **netiod** (for the ABone).
- Using netiod, supports “activated legacy” apps that need “**raw**” mode network I/O (“if/ipv4”, “if/ipv6”)

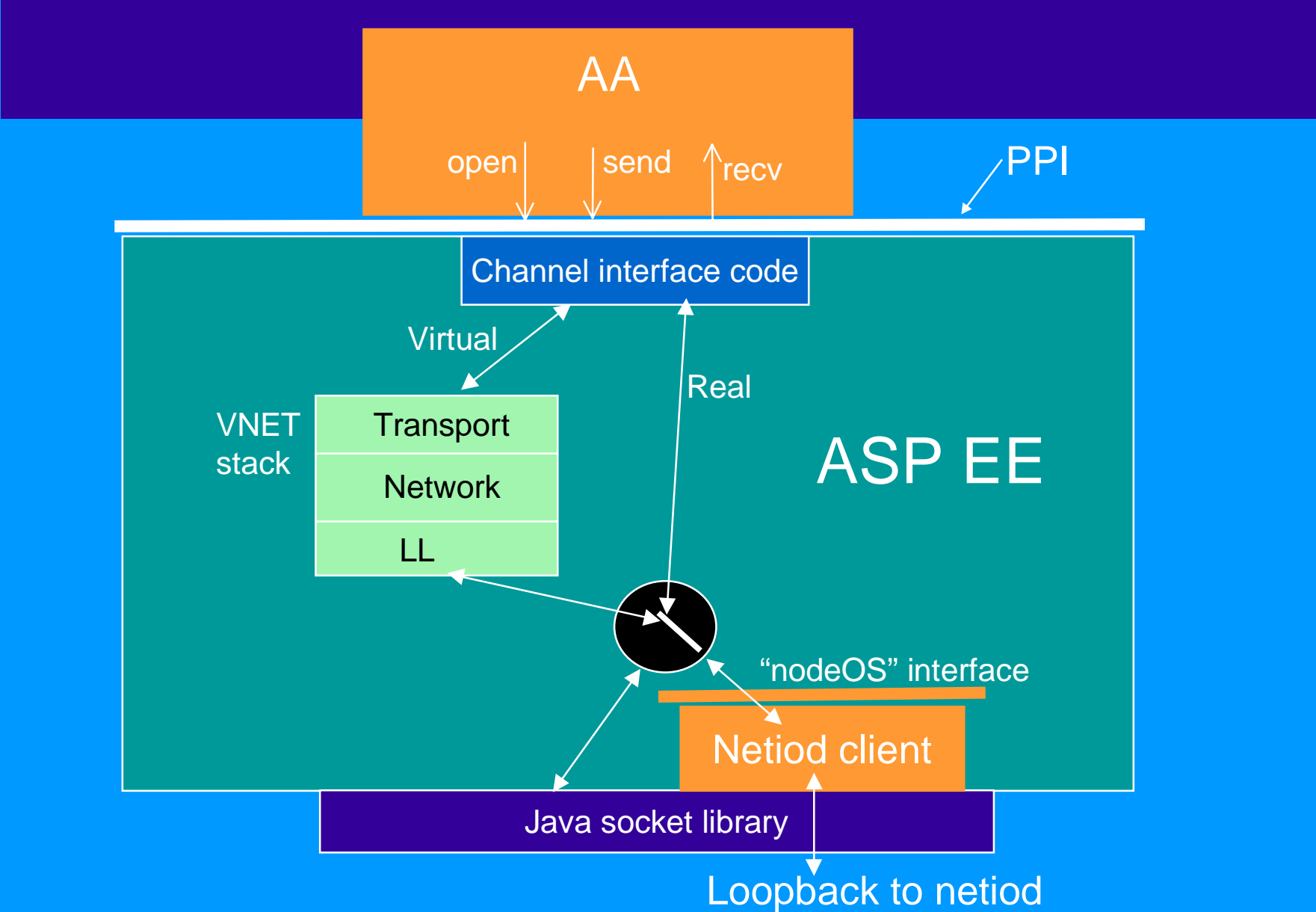
ASP EE Channel Extensions

- Virtual protocol stack --> e.g., protocol spec “vif / vn / vt”
- “asp” protocol layer --> e.g., “vif / vn / vt / asp”
- ***receivePacket()*** upcall passes actual values of wildcarded fields (***Attributes*** object ~ “AddressSpec”)
- Connection-oriented channels (TCP, RDP, UA API)
 - Added ***ChannelException*** upcall for EOF, etc.
- Inter-AA communication: “ipc” channels

Extended Channel Abstraction

- Supports (real) IPv6 stack
- Supports an API to a User Application (UA)
- “Header” support
 - Slightly stretches channel abstraction
 - Inchannel for “if/ipv4h” or “if/ipv6h” means:
perform IPvx input processing BUT pass header to AA.
(Pragmatic: build AA to translate IPv6<->IPv4 headers
before we have “if/” support in netiod)
- Interface -> Logical Interface Number (LIN)
 - Single space including all logical, virtual, & physical interfaces of native IPv4/IPv6, as well as virtual topology interfaces.

-
-
-



-
-
-
-
-
-
-
-

ASP EE Channel Abstraction

- Address Spec represented in PPI by *Attribute* object.
- *Attribute* object can be passed in *send()* call, to set/override parameters set at channel open time. Functionality of Unix *sendto()*.
- Address spec fields: added TTL
- Generic DemultiplexKey supported
 - Addition: matching operator can be `eql` `neq` `leq` `lss`

ASP EE Channel Abstraction

- Example ProtocolSpecs supported by PPI:
 - “vif /vn/vt” Datagrams via virtual protocol stack
 - “vif/vn/vt/asp” Same, ASP EE header stripped/added by EE
 - “vif7/vn/vt/asp” Same, Logical Interface Number 7
 - “vif/vn/rdp” RDP stream in virtual protocol stack
 - “if/ipv4/udp” UDP datagrams over IPv4
 - “if/ipv4” Raw IP
 - “if/ipv6/udp” UDP datagrams over IPv6
 - “if/ipv6h” Raw IPv6 packets with IPv6 header included
 - “ipc” Inter-AA communication

Netiod: Protocol Specs

- Virtual stack
 - Datagrams: vif[n] / vn/vt [/ asp]
 - Connections: vif[n] / vn / rdp [/ asp]
 - Virtual link layer: vif
- Real stack
 - Datagrams: if[n] / ipv4 / udp [/ asp]
if [n] / ipv6 / udp [/ asp]
 - Connections: if[n] / ipv4 / tcp [/ asp]
if[n] / ipv4 / udp / rcp [/ asp]
 - Network layer (“raw mode”): if[n] / ipv4[h] [/ asp]
if[n] / ipv6[h]

ASP EE Status

- Version 1.5: released last week (surprise!)
 - Many improvements in channel-based network I/O
 - Code reorganized.
 - Tuned -- performance improved by several factors of 2.
 - IPv6 support added
 - Demux key support added
 - Support for IPv4h, IPv6h added
 - Easier configuration
 - One file for all nodes in topology (like ANTS)
 - Configuration tools, including ANTS \leftrightarrow ASP conversion
 - A few new experimental AAs

Lessons

- Java may not be ideal programming language for AN.
(You heard it here first...)
- We have had serious portability and performance problems with Java.
 - Sun has been addressing these problems, but not fast enough. We had to spend a lot more time on these issues than we wanted.
 - Portability:
 - Not release-independent and not very system-independent
 - Often found ourselves searching the space of JVMs to find one that works on a given OS platform.
 - In practice, things built using Java seem to break a lot.

Java Lessons...

- Performance:
 - Garbage collection is very nice, but a performance problem.
 - JITs and very careful coding and benchmarking can produce code that is nearly as good as C++ ... But that is hard and costly. Using JIT is almost sure recipe for obscure failures.
- Dynamic code loading is easy, dynamic unloading is hard.
- Whatever happened to proof-carrying code and all that good stuff?

Lessons

- Systems integration is Hell. (surprise)
 - A Bone integration of the ASP EE has been painful, but this was probably unavoidable.
 - (I have whinged about this before...)



Mistakes (?)

We put a lot of effort into areas that we thought would be important for users; only time will tell whether they were worthwhile.

- Virtual networking stack
 - This was a response to early PM pressure to reinvent networking using active networking. Really, people want to use IP as basis.
- Channel abstraction in PPI
 - We did learn a lot about the channel abstraction, which hopefully will improve the nodeOS spec.
 - Channels also provide a neat packaging of the many network I/O options.



Mistakes (?) ...

- Common class byte code sharable among AAs
 - Goal: customization & extension of large, complex protocols
 - => multiple AA versions, almost identical code.
 - => Share common class byte code among AAs
 - Error! Cannot delete code; awkward for development.
 - [Partial] solution: allow either: a shared classloader for debugged code, or individual classloaders for new code.

Omissions

- Open security problems in ASP EE
 - Permission file controls AA access to privileged EE functions.
 - Credential-based authorization: not yet implemented.
 - AA code is not signed
 - Need more resource limitations on AAs
 - CPU: EE schedules ASP threads using approx RR discipline, and demotes priority of threads that should be gone.
 - Heap protection: could adopt Utah multi-heap mechanism.
 - Network I/O: need output packet scheduling to control per-AA share of network output -- something like a token bucket.

Omissions

- Functional limitations
 - No cut-through channel support
 - No support for traffic control
 - No routing socket for native IP.
 - Code loading over the network needs to be optimized.
 - Class file versioning only partially implemented.

More omissions...

- Good ideas that we did not complete:
 - Procedure for porting AAs from ANTS to ASP.
 - “Probe” mechanism for AA management.
- We did not use our AN machinery to explore design of signaling and other Internet protocols
 - Exception: AFSP for the SANDS demo.

-
-
-

References

- Web page: <http://www.isi.edu/active-signal/ARP>
- ABone: <http://www.isi.edu/abone>