Security Issues in Active Networking

IWAN '01 Tutorial

September 30th, 2001

Jonathan M. Smith

University of Pennsylvania

http://www.cis.upenn.edu/~jms

Tutorial Outline:

- Security Challenges
- Principles of Active Network Security
- Protection Methods
  - Language Based
  - Node Architecture Based
  - Network Based
- Applications
- Summary

1. Security Challenges

- What is *security* (versus, say *safety*)?
- Protection of Resources
- Computational Resources
- Objects/Memory Resources (namespace)
- Bandwidth Resources - QoS

Challenges: Safety & Security

- Safety: Accidents; Security: Malice
- Specification of goal (@30,000 feet!):
  Right Information to
  Right Place at
  Right Time
- Insecurity: Deviation from goal
  e.g., information to *wrong* place

Right information/Right place

- Requires identifying information units
- Requires identifying places

  e.g., locations, personnel, etc.
- Requires *security association*

  e.g., per-place *password* encrypts info.

  deny information to other places

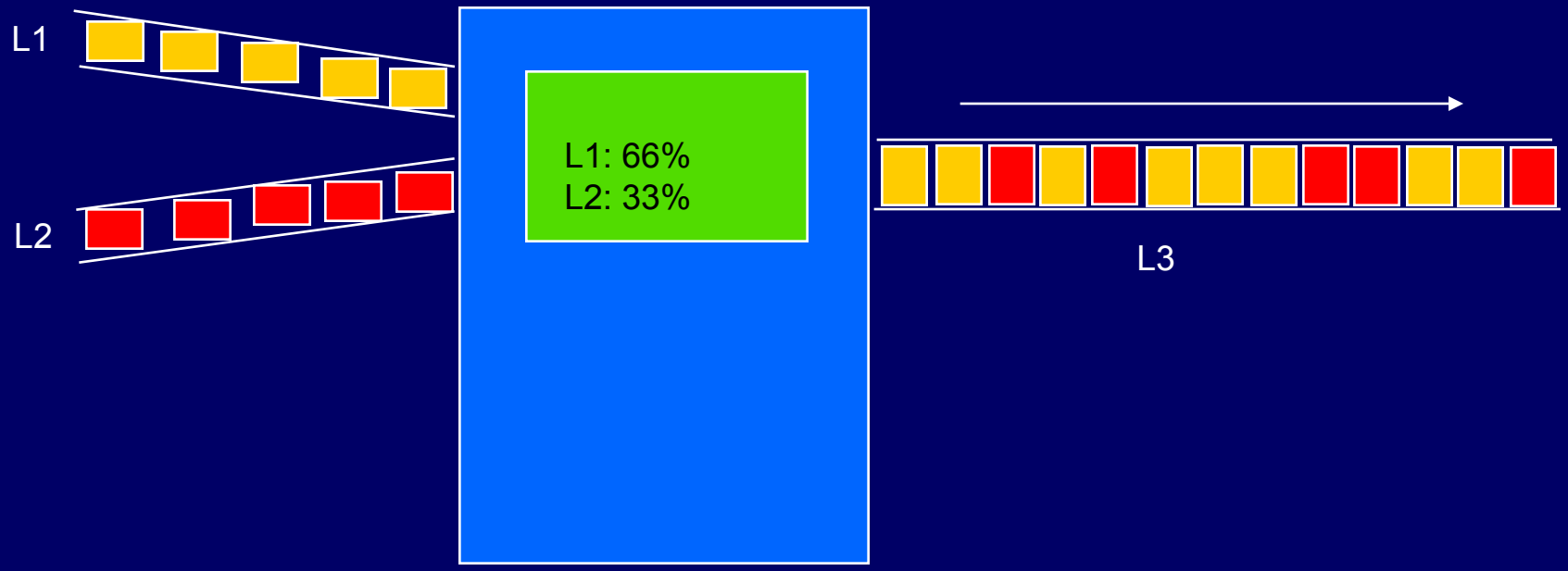
  cryptographic protocols: good progress

Right Time (the tricky one)

- Late information may be useless
- Basis of *denial of service* attacks
- Requires identifying *real* times
- Languages have no time semantics
  - gettimeofday() in C/Unix world
  - is ML better? (Dannenberg's *Arctic?)*

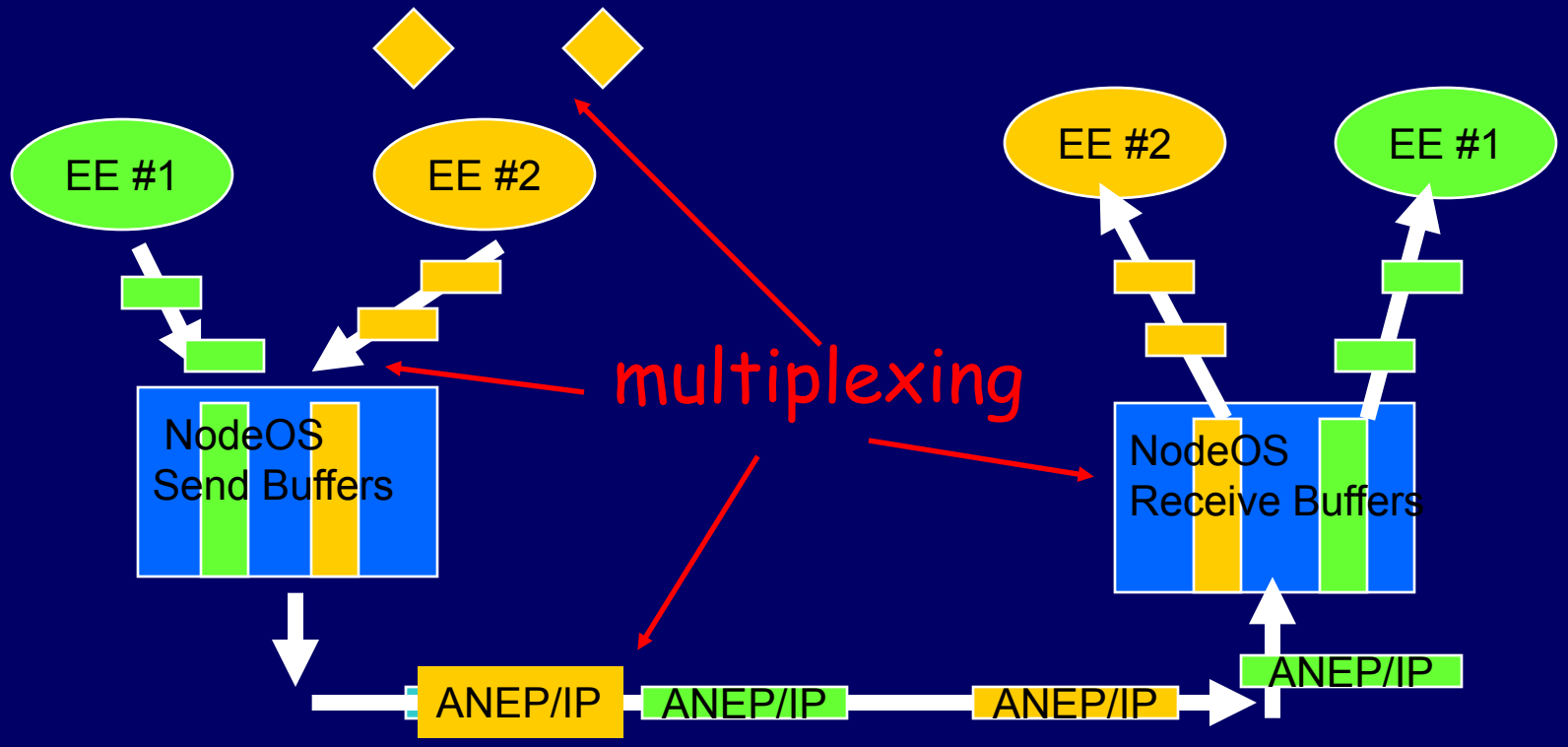
Need to control multiplexing

- E.g., assign L3 bandwidth 66%/33%



Resource Management, End-to-End

- Resource Management Challenges

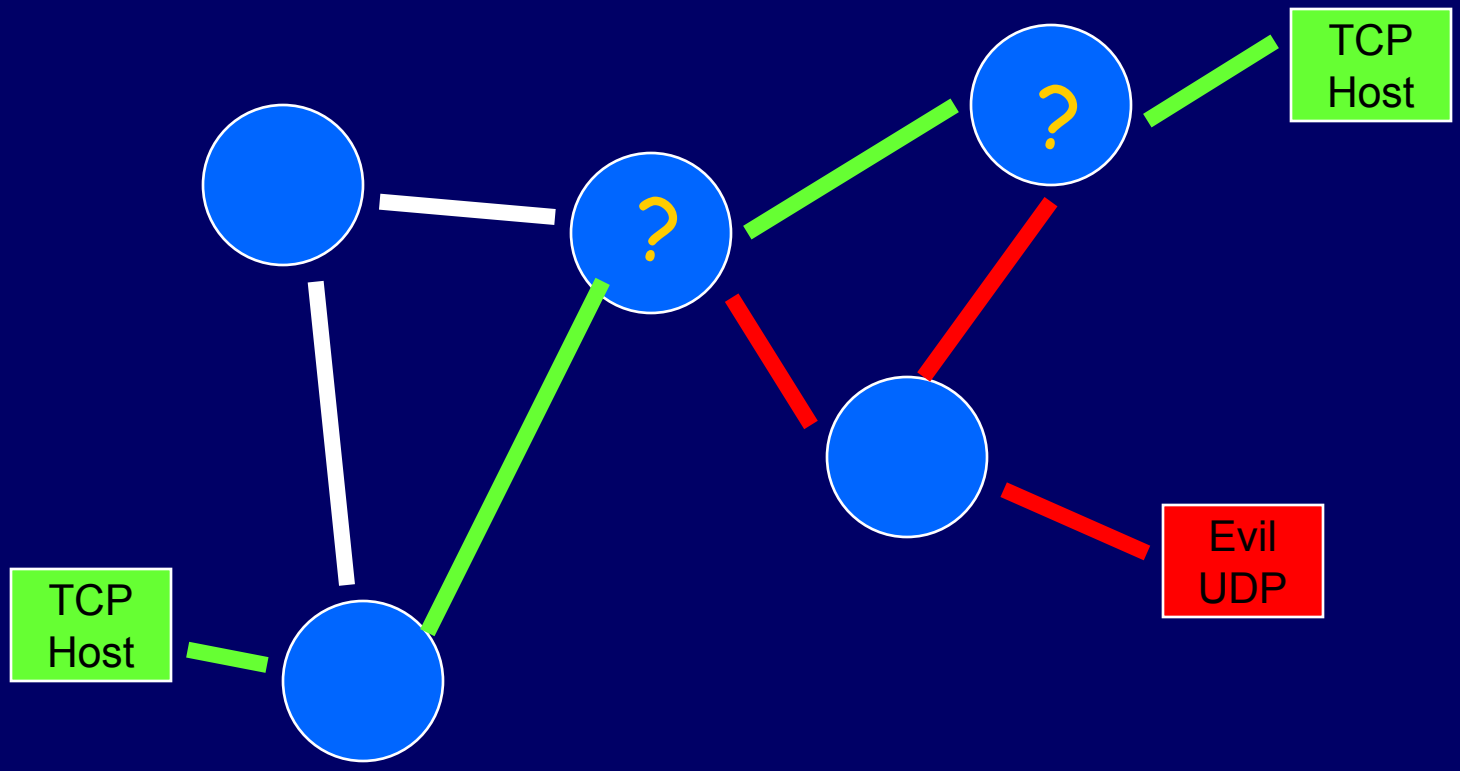


Denial of Service

- Easy to protect server hosts
  - Resource domains, interrupt masking, firewall shielding on host *itself*
- But service is unprotected between client and server site
- This problem *must* be solved with network-embedded functionality

Denial of Service attack

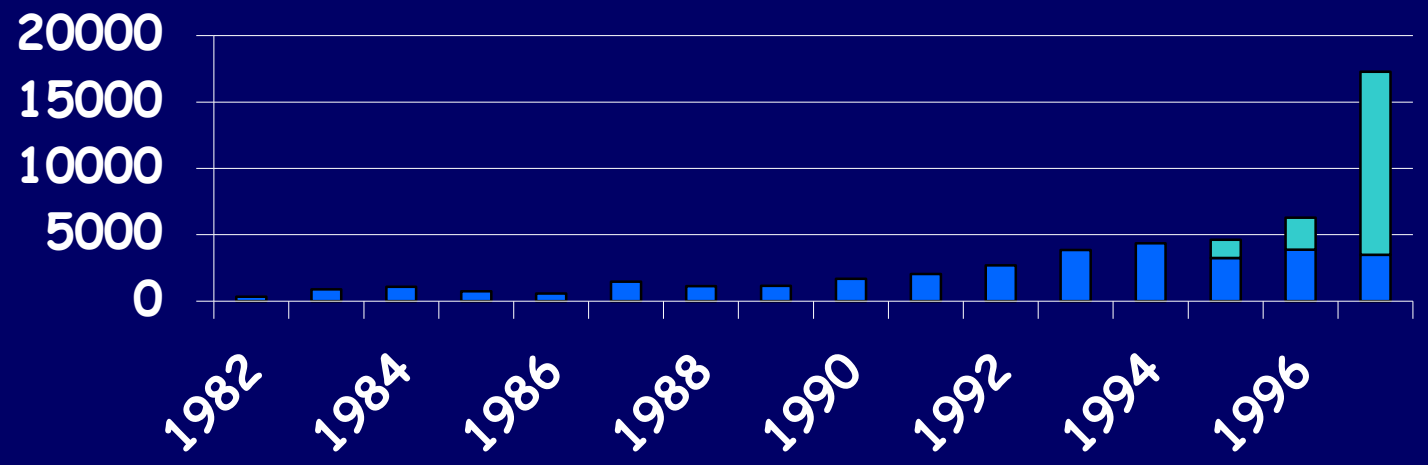
- Cross traffic in an Internet



2. Principles of Active Network Security

- Existing Internet is ad-hoc (& complex)
- AN complex in a different way, but leverages new design principles
  - Modern programming languages for safety
  - Extensive use of cryptography
  - Conscious of resource control & QoS
  - Safe, rapid adaptation to change

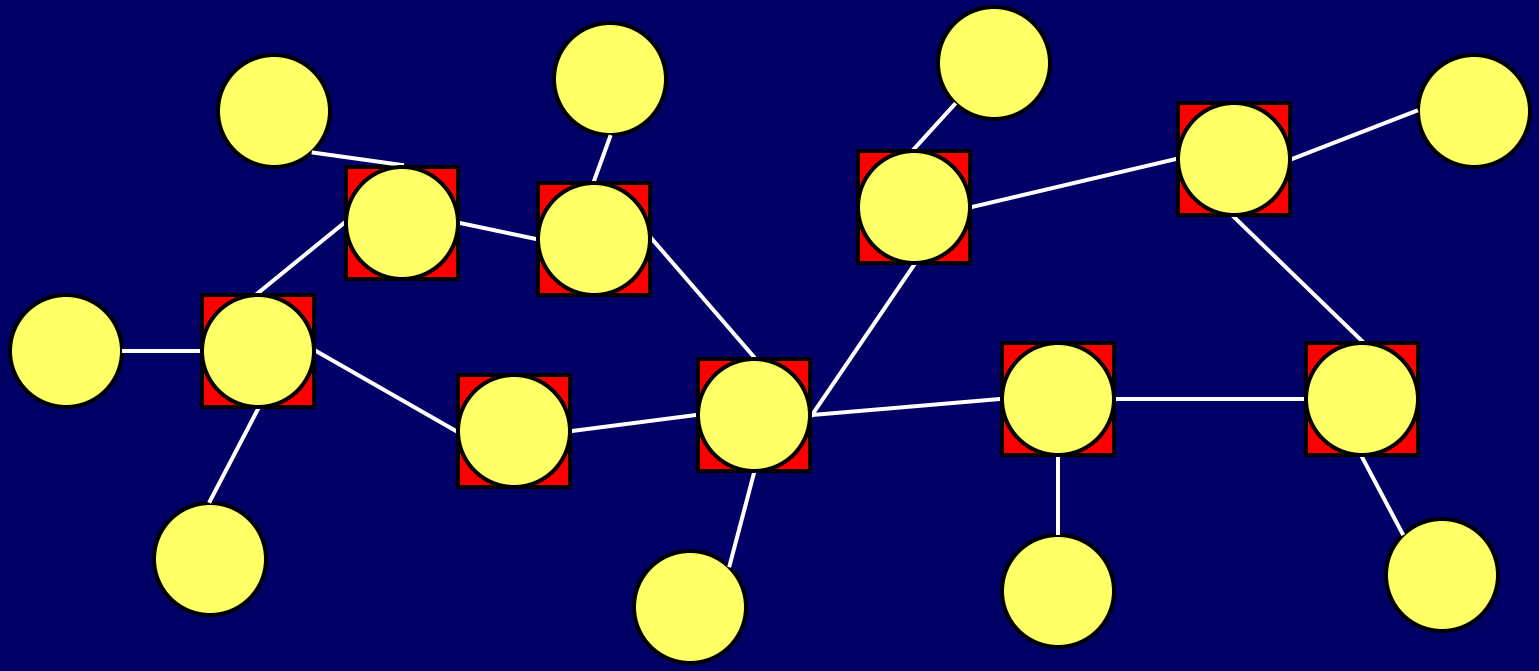
Complexity: RFC Pages '82-'97



- Draft pages overlaid since 1995
- log(f(pages,year))>log(f(users,year))?

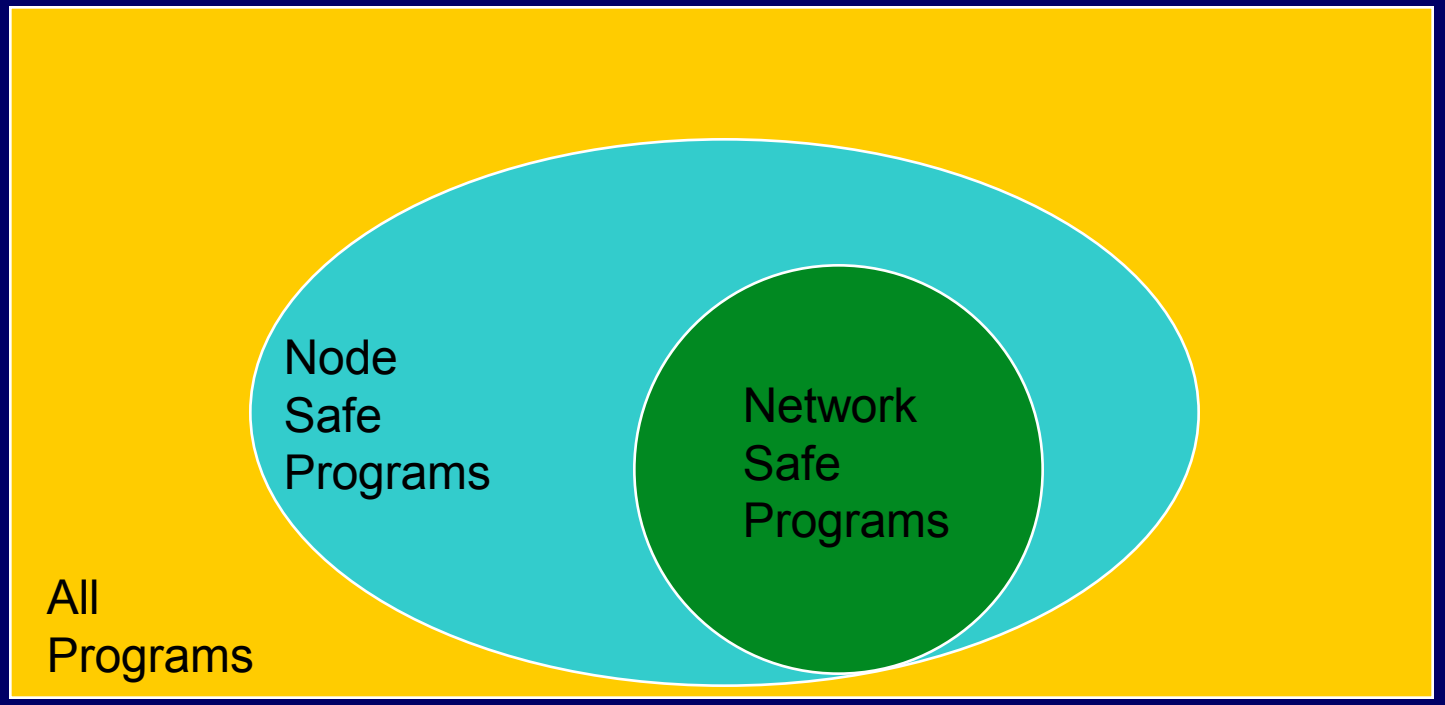
Active Network Model

- Packets can change the behavior of the switches "on-the-fly"
  - In-band active packets
  - Out-of-band active extensions



Restricting Programs

- Node safe versus network safe



Observations

- Code is either untrusted or trusted
- If untrusted, mechanisms must defend against misuse
- Alternative is to make trusted, e.g., by trusted compilation with restrictions
- Untrusted requires heavywight mech.

  Example – OS, *e.g.,* Cambridge Xenoserver

Principles

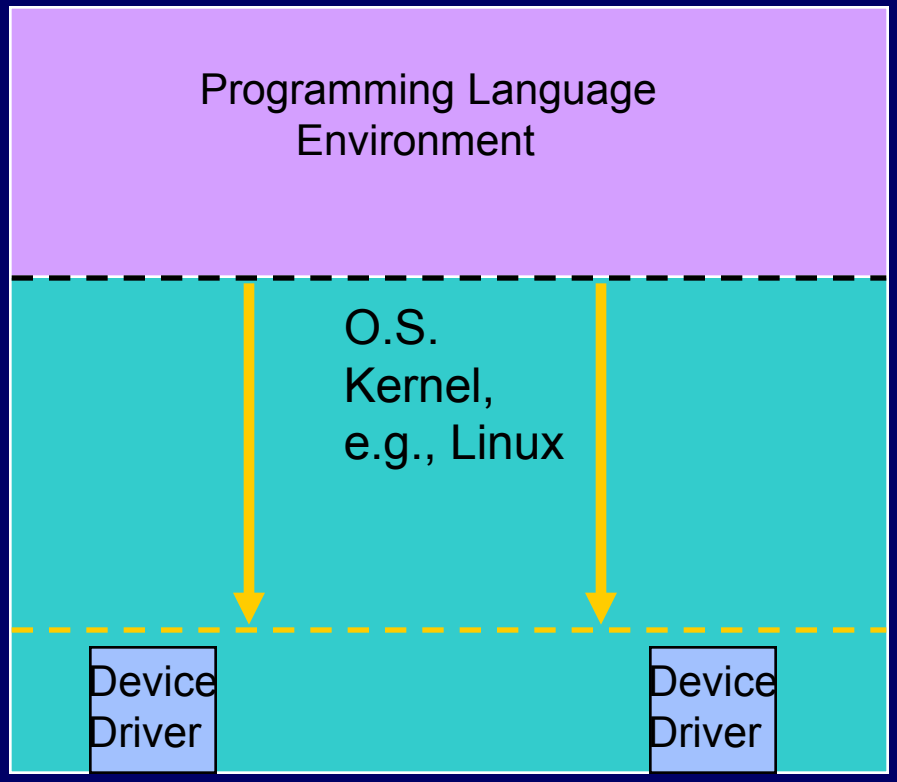
- Network safety without node safety hard
- Node guarantees can extend to net
- Distributed state hard to model, particularly at compilation time
- Cryptography is *not* security
- Resource management limited to layers under control (e.g., bridge for layer 2)

3. Protection

- What must be protected in a network?
- Bandwidth resources
- Memory resources
- CPU / computational resources
- Static versus dynamic protection
- Location of protection in architecture

How do we control programs?

- Safety & Security: P.L., O.S. or hybrid?



3.1 Languages / Programming

- ANTS
- ALIEN
- PLAN
- SNAP
- Language Independent Approaches

ANTS

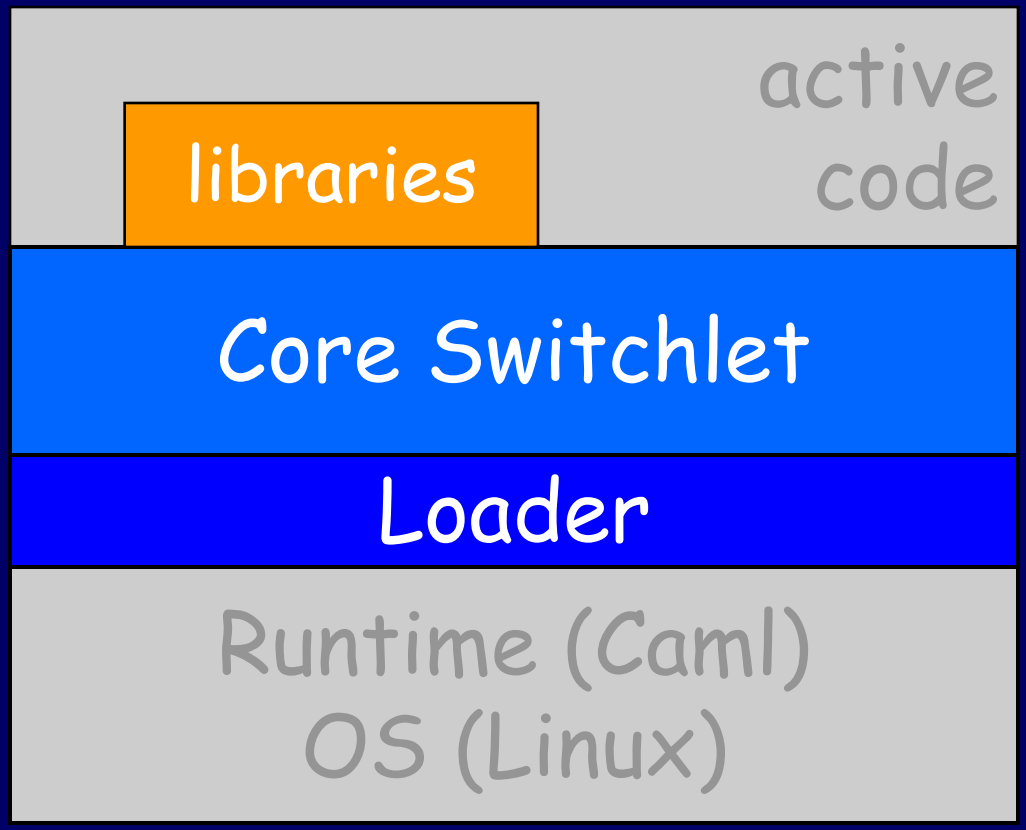
- Active Network Transport System
- See Weatherall paper in SOSP 99
- ANTS based on a Java platform
- Used for several applications such as Active Reliable Multicast (ARM)
- Security Model of Namespace Isolation
- Achieved with MD5 hash of module

The ALIEN Active Loader

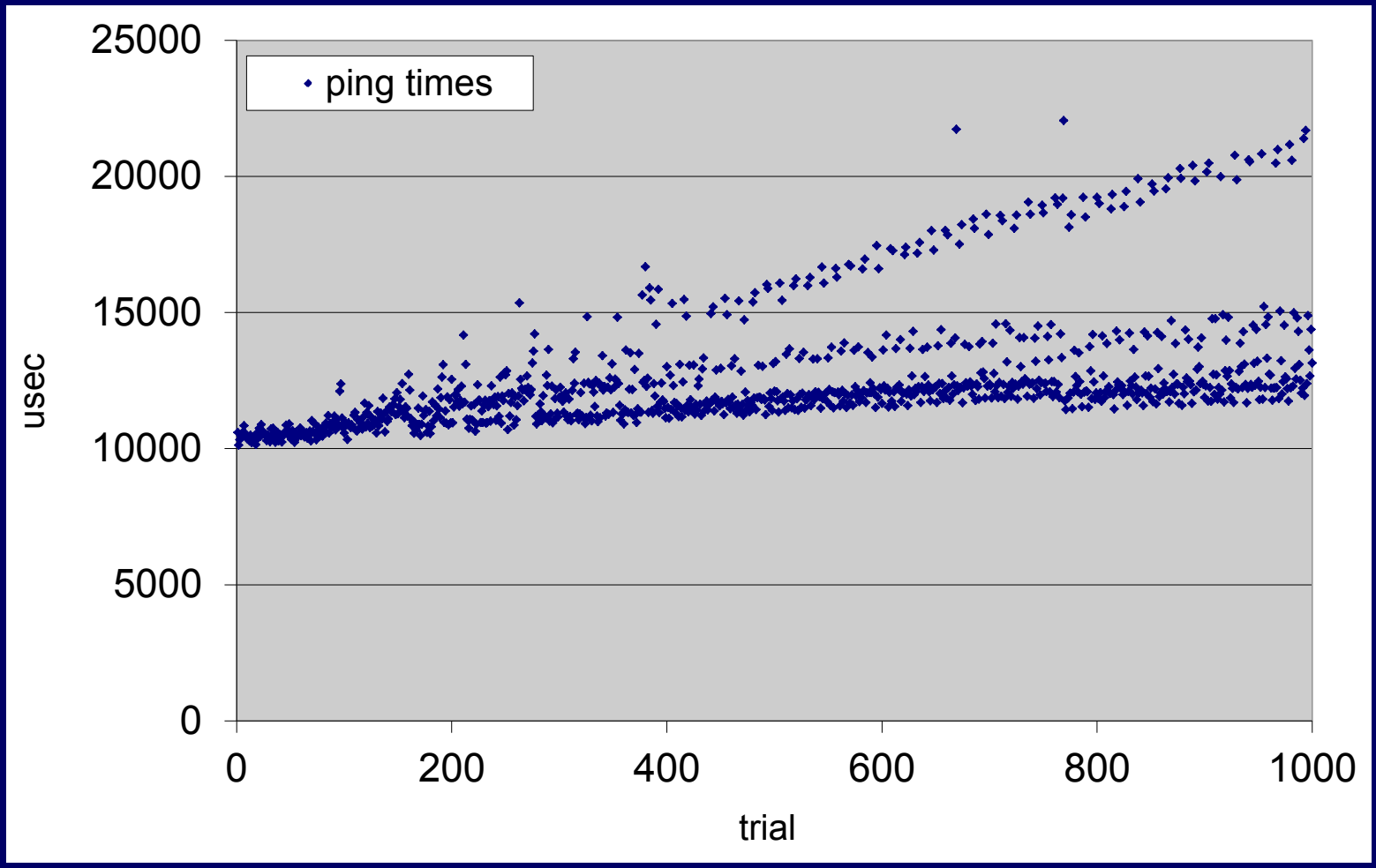
- See Alexander IWAN 99
- Active extensions and packets in CAML
- Namespace isolation via module thinning
- Only privileged portions of the system can directly access shared resources
- Digital signatures for remote accesses
  - Resembles a capability model

ALIEN in an Active Element

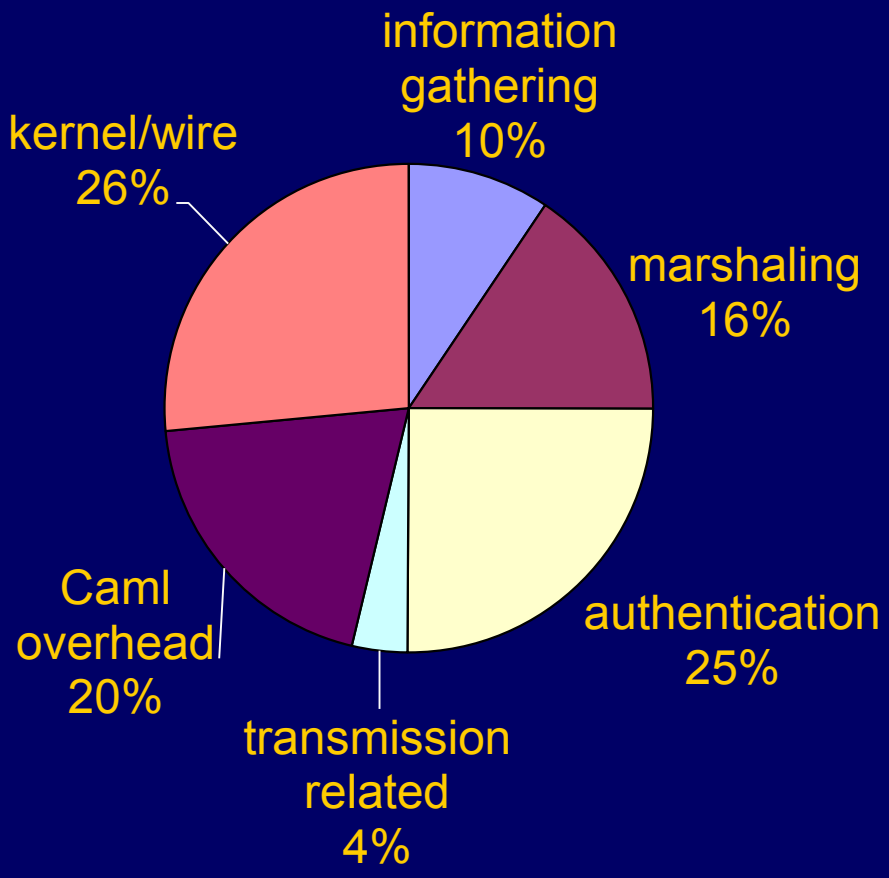
- Three layer architecture



saneping Performance



Overall Breakdown of Costs



Major Costs

- Kernel/Wire (26%, 3078 μs)
  - Kernel time + transmission time
  - To avoid
    - Reduce size of packet
    - Reduce or avoid kernel boundary crossing cost
- Authentication (25%, 2910 μs)
  - Mostly cost of performing SHA-1 (4 times)

Cryptography is Expensive

- Implemented in C because too slow in Caml
- Times to hash 4MB of data

	bytecode	native
Caml Int32	86.45s	61.99s
Caml int	36.03s	2.48s
C		0.33s

The take-home lesson:

- Must reduce per-packet crypto costs:
  - Active extension amortizes costs
  - ANTS caching amortizes costs
  - Smaller packets (Dense CISC, a la BBN)
- Or, find another way to avoid crypto in the common case…

Packet Language for Active Networks (PLAN)

- Hicks, Kakkar, Moore, Gunter, Nettles
- Capsule-based approach
- CAML runtime
- Highly-restricted domain specific language (a safe "glue" language, like the UNIX shell), extensible via ALIEN
- Active extensions do restricted things

Safe and Nimble Active Packets

http://www.cis.upenn.edu/~switchware/SNAP

- Build on first-generation successes
  - Notably, PLAN/PLANet
- Address open issues of
  - Resource safety
  - Performance
- SNAP adds flexibility over IP without sacrificing safety or efficiency

SNAP Language Design

- Stack-based bytecode VM
- SNAP instructions
  - Simple computation, environment query, control flow, packet sends
- Resource safety via language design
  - Execute in constant time and space
  - All branches go forward

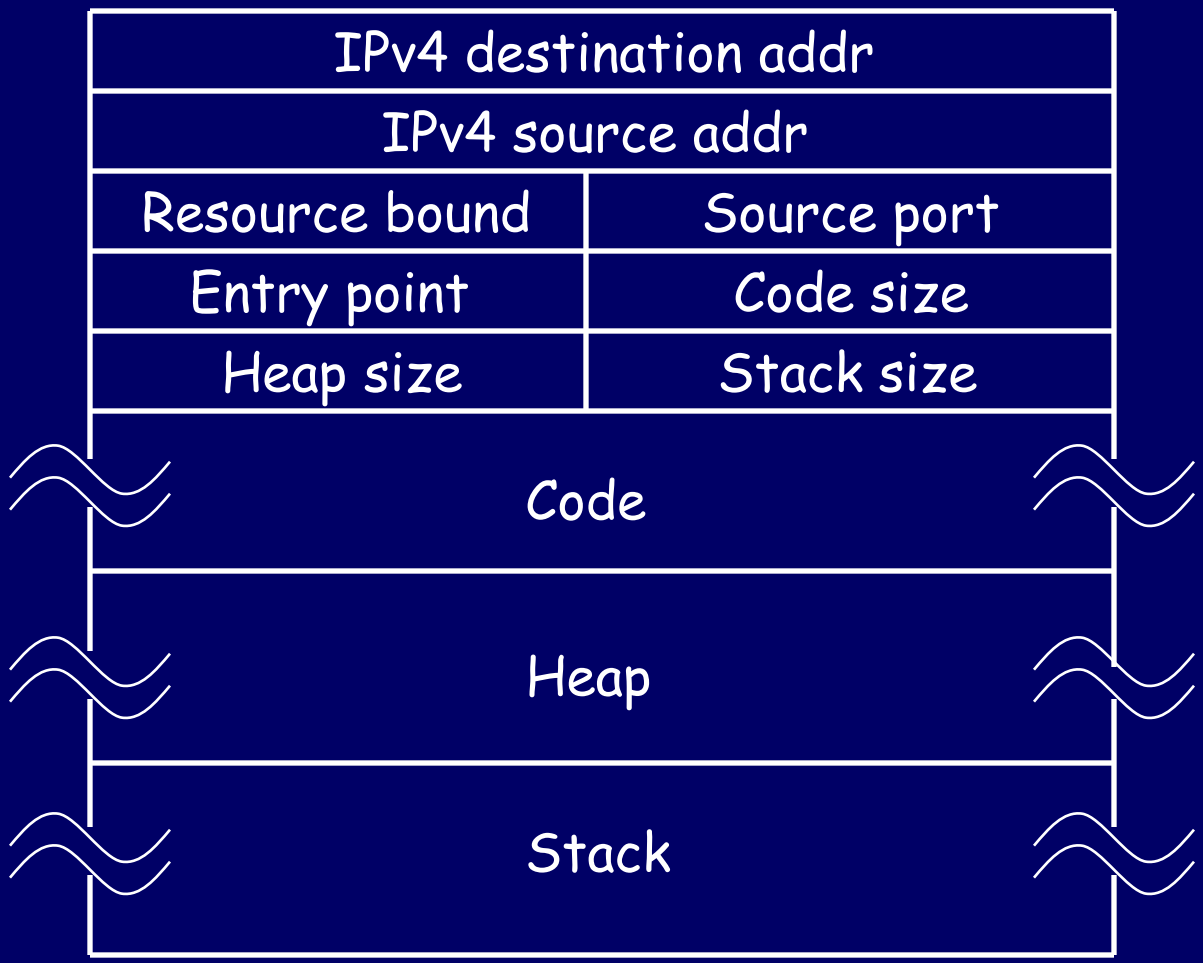
Safety and Security

- Node integrity
- Isolation
- Resource safety
  - TTL-like resource bound
  - Linear CPU and memory usage
  - *A priori* guarantees

Compilation techniques

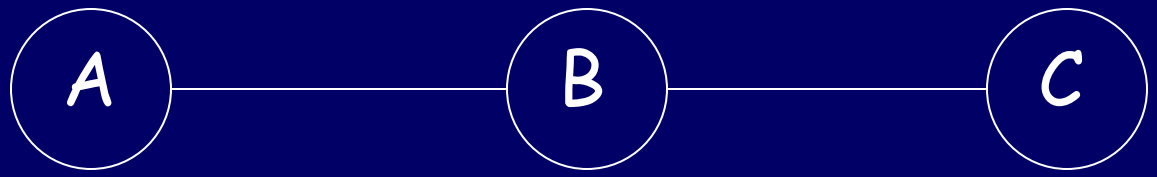
- SNAP: no backward branches!
- Function inlining
- Loop unrolling
  - User provides upper bound
  - Generally few iterations anyway
- Code size issues
  - Most PLAN programs very simple
  - Resource use tied to access bandwidth

Packet format



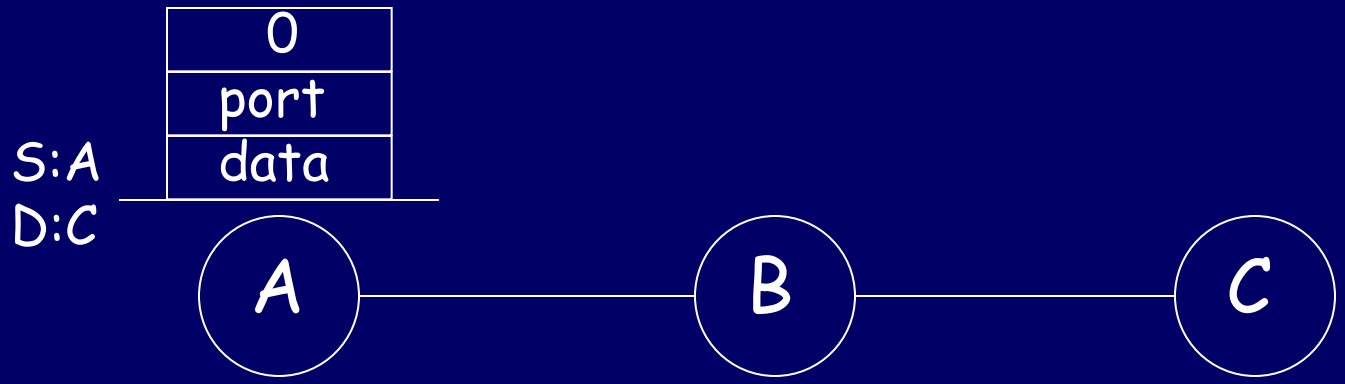
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



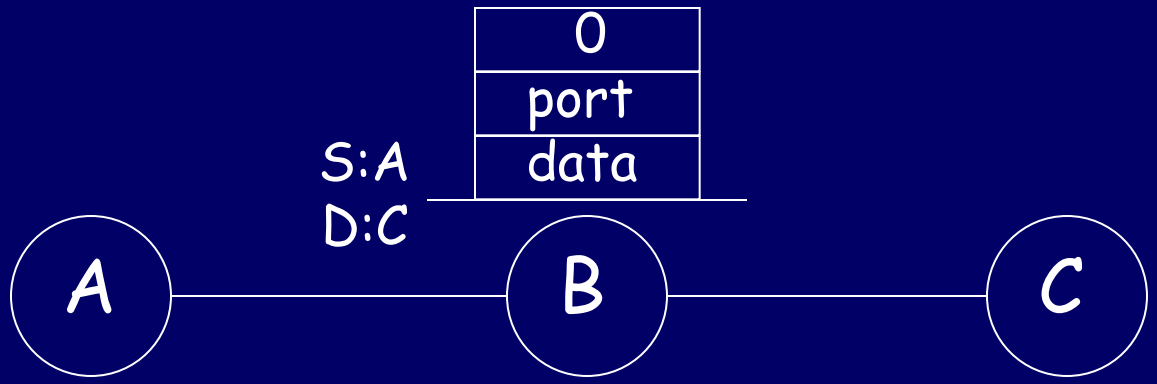
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



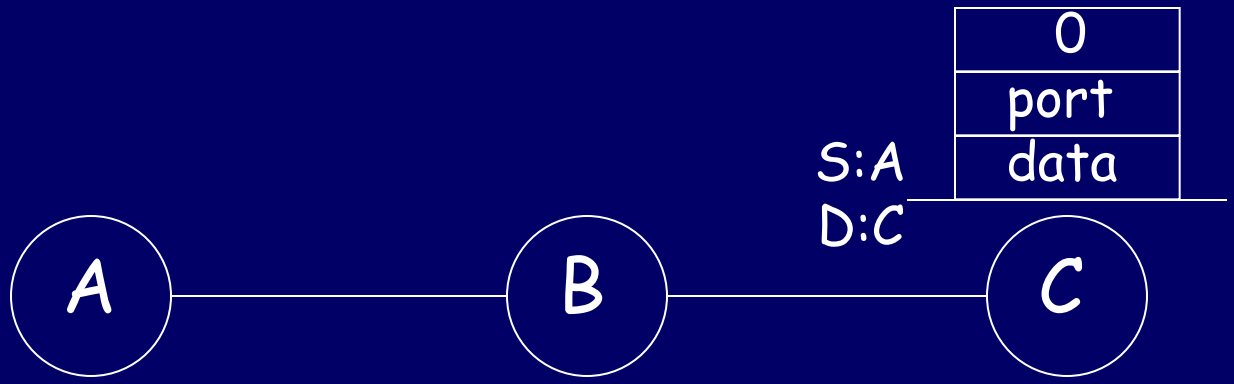
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



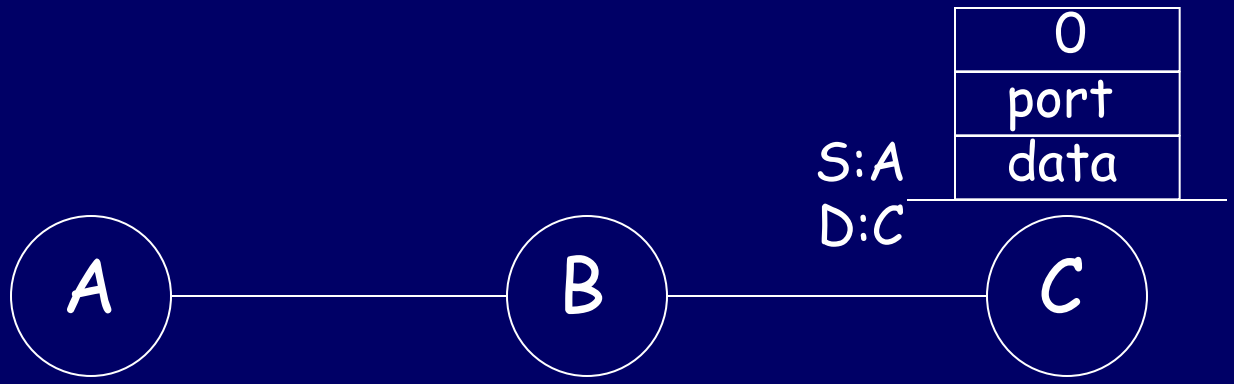
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



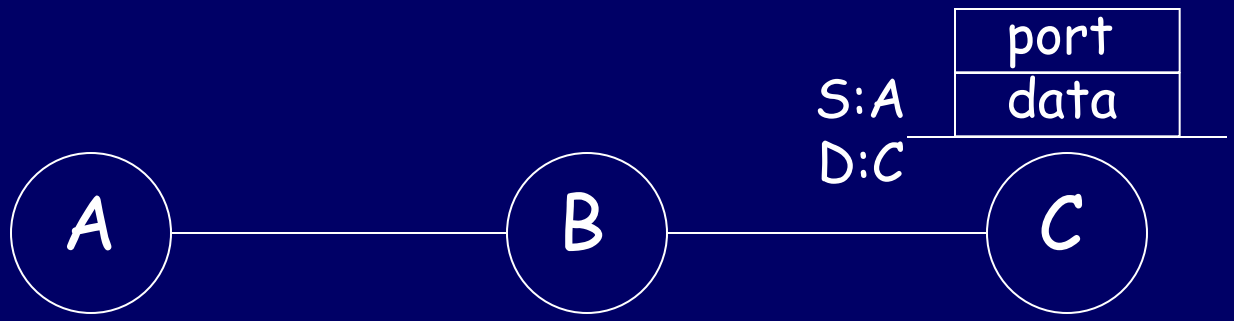
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



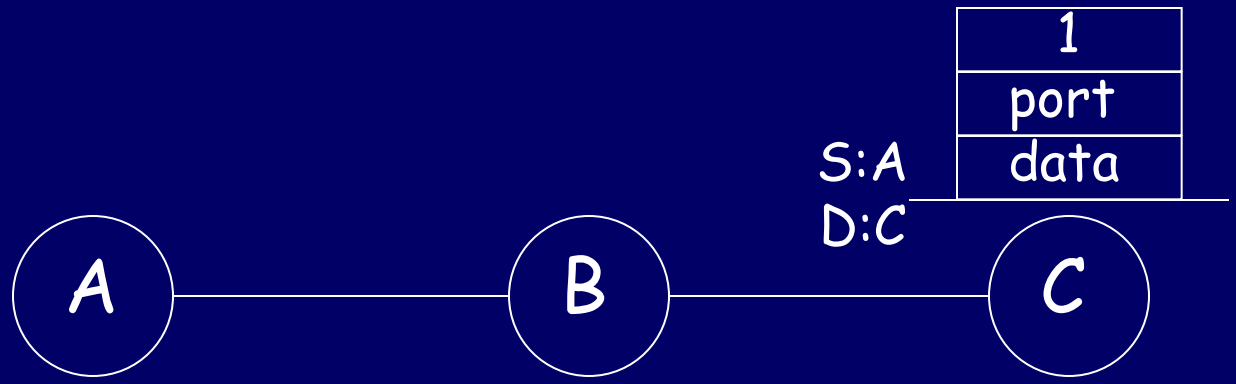
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



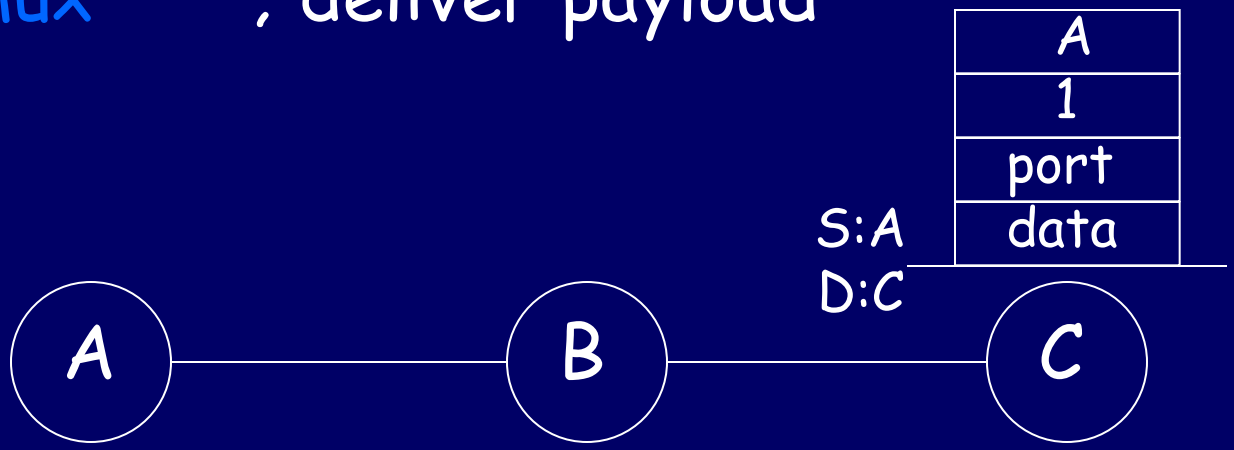
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



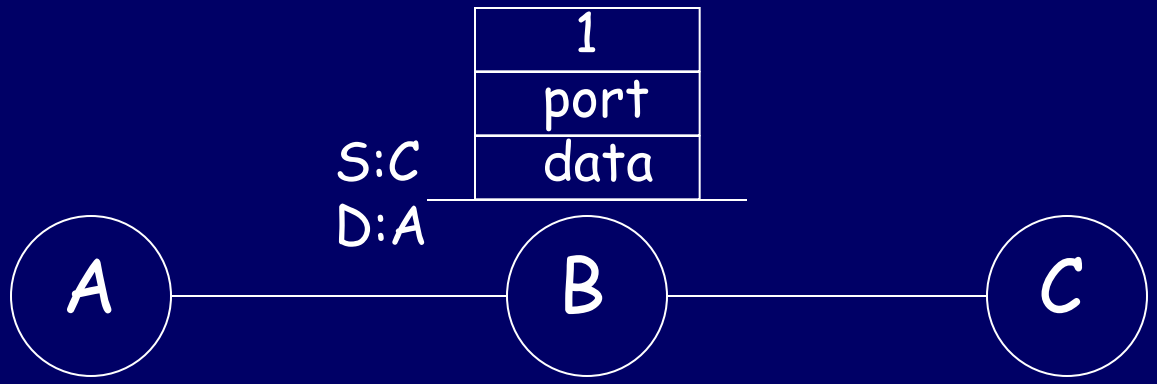
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



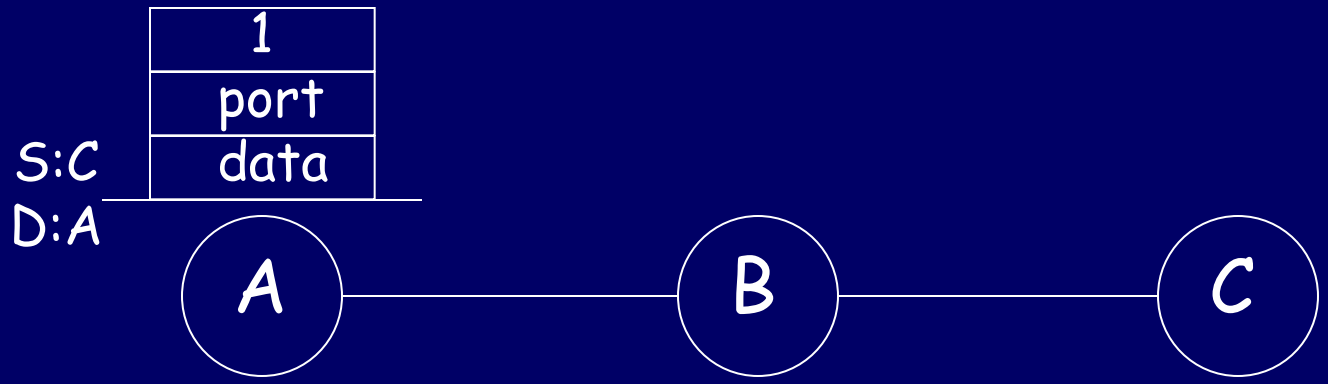
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



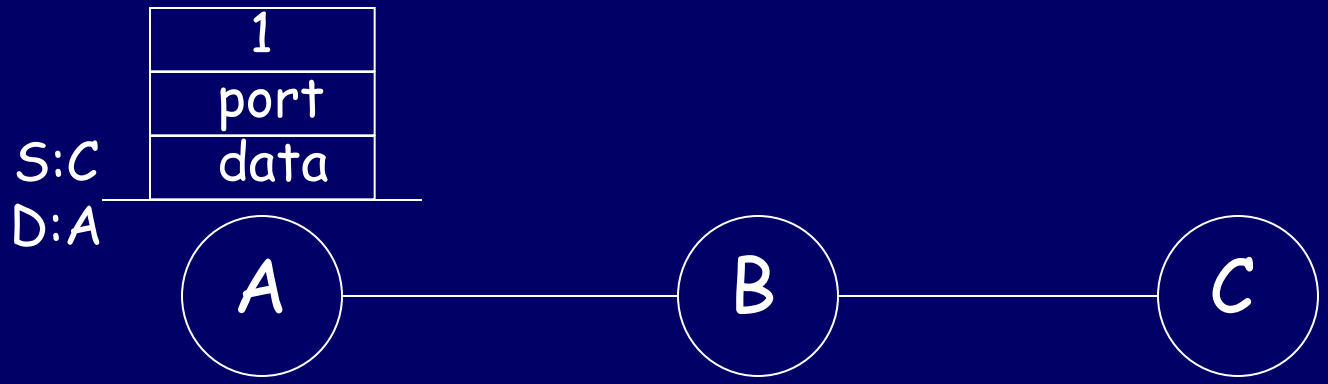
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



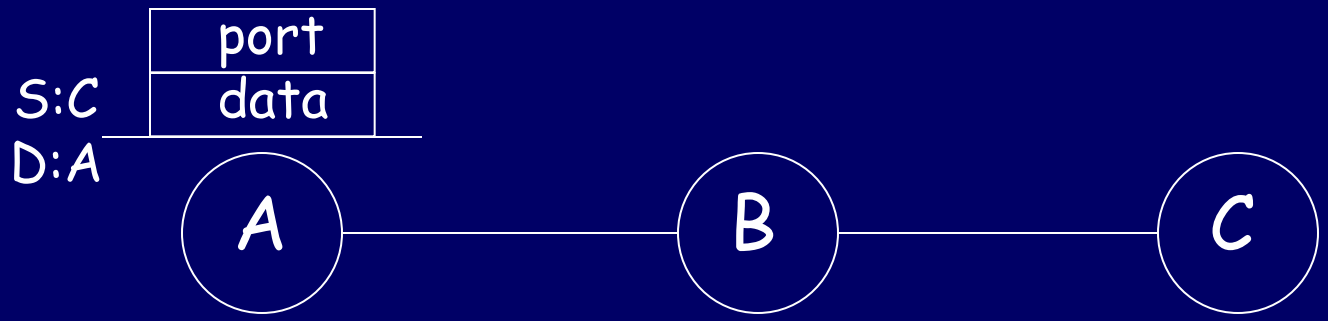
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



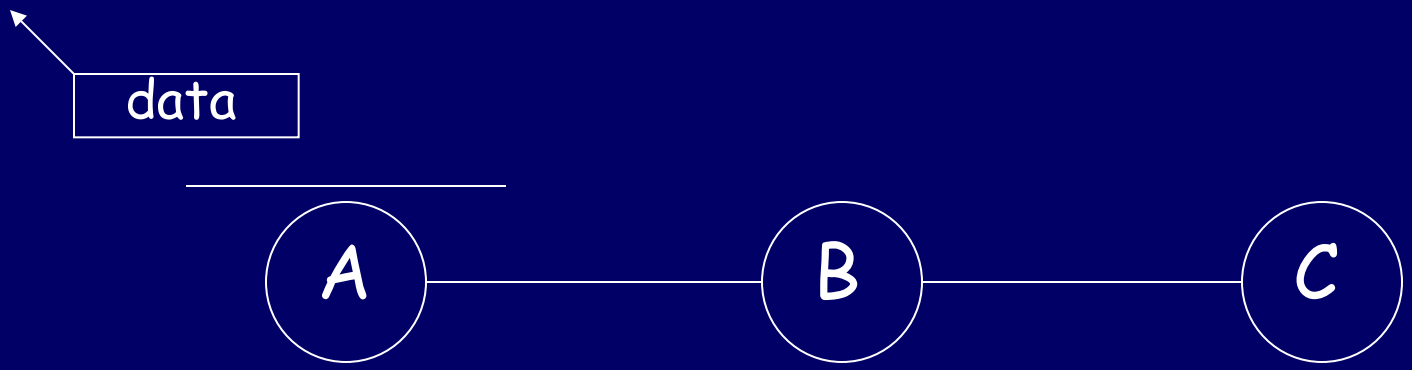
Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



Example: SNAP ping

```
forw        ; move on if not at dest
bne  5      ; jump 5 instrs if nonzero on top
push 1      ; 1 means "on return trip"
getsrc      ; get source field
forwto      ; send return packet
pop         ; pop the 1 for local ping
demux       ; deliver payload
```



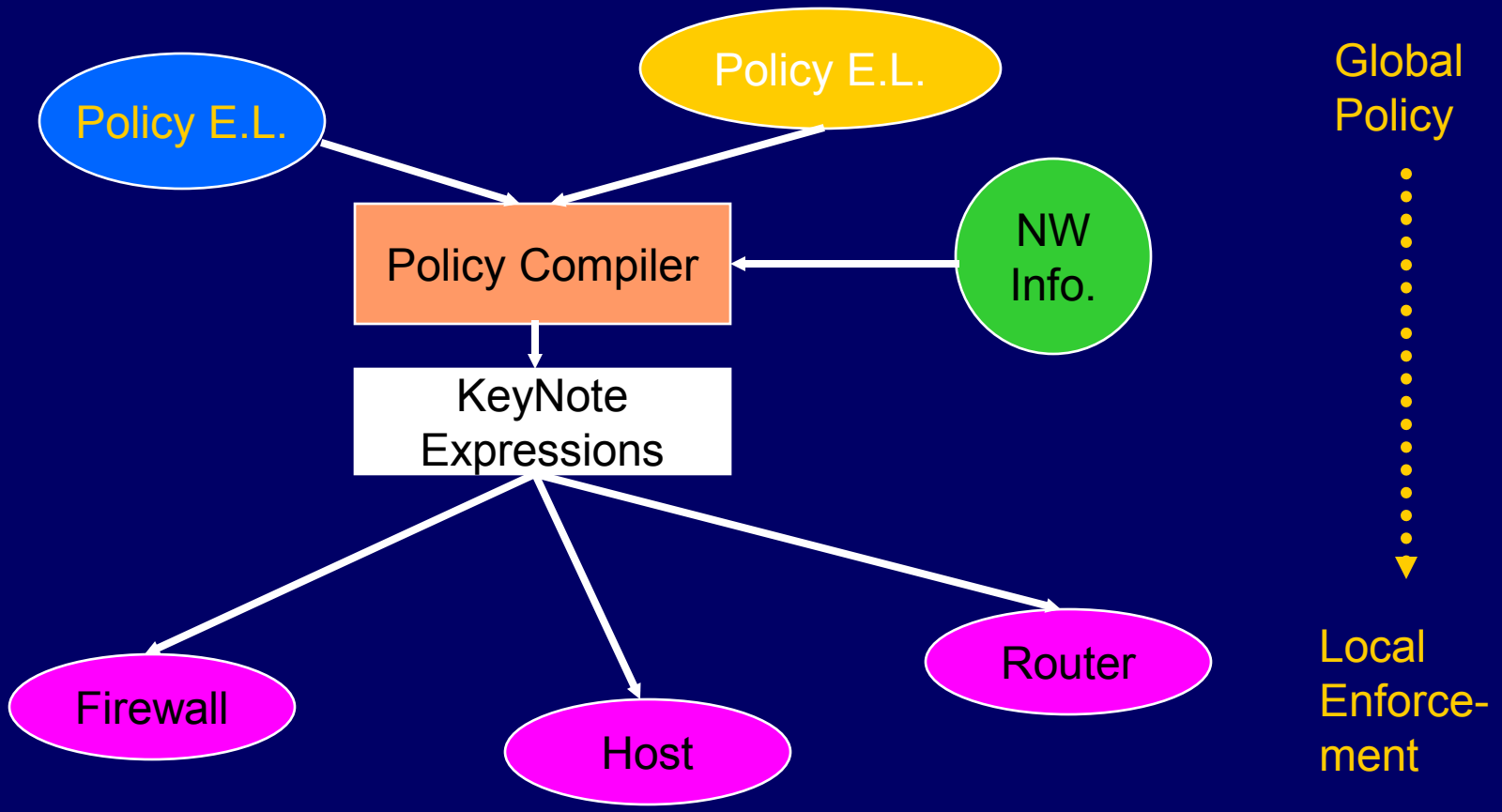
SNAP Contributions

- Provable resource safety
  - Linear resource usage
  - Guaranteed via language restrictions
- Retains flexibility of previous systems
- Performance comparable to an IP software router

Language Independent Approaches

- Proof-carrying code – a major idea
  - Program (mobile code) carries with it a proof of correctness
  - Much easier to verify proof than to perform proof
- Code-signing
  - Challenge with code signing is trust management

STRONGMAN Architecture



STRONGMAN

- Penn / AT&T Research
- Logical "meta-KeyNote"
- High-level *policy* compiles to KeyNote
- Policy-based configuration of *groups* of security endpoints (firewalls, hosts, routers, …)
- Multiple *policy expression languages* compile to common KeyNote policy model

Describing Actions in KeyNote

- <Attribute,Value> **Action Environment**

  $filename "/home/stan/foo"
  $owner "stan"
  $hostname "lake.sp.co.us"

- Attribute semantics application-specific
- An Action always associated w/Requestor

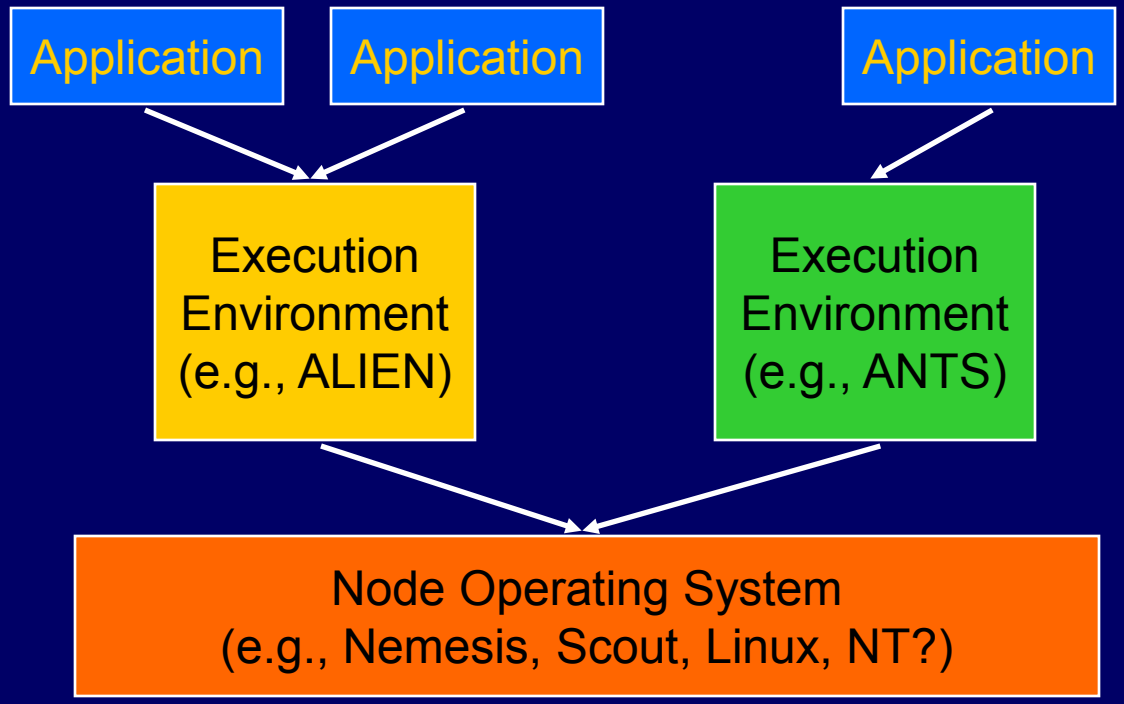
KeyNote Example

- Authorizer: *stan's public key*
- Licensees: *wendy's public key*
- Conditions: $file_owner == "stan"
  && $filename ~= "/home/stan/[^/]*"
  -> "true";
- Signature: stan's signature

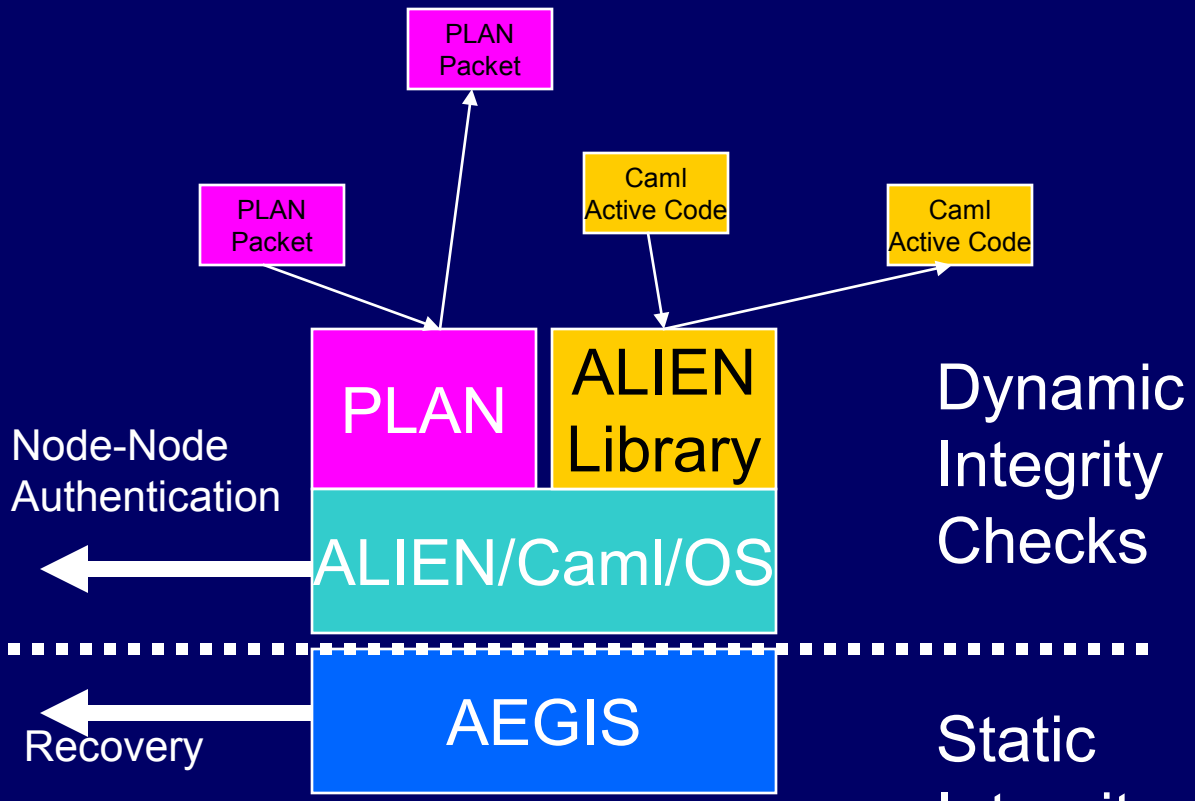
3.2 Node Architecture

- A Standard Architecture has been defined by DARPA
- There are various places where security principles must be applied
- When consider entire node, must take NodeOS & hardware into account
- Several NodeOS: Scout, Janos, AMP (see Peterson, *et al.*, IEEE JSAC)

"Active Network Architecture"



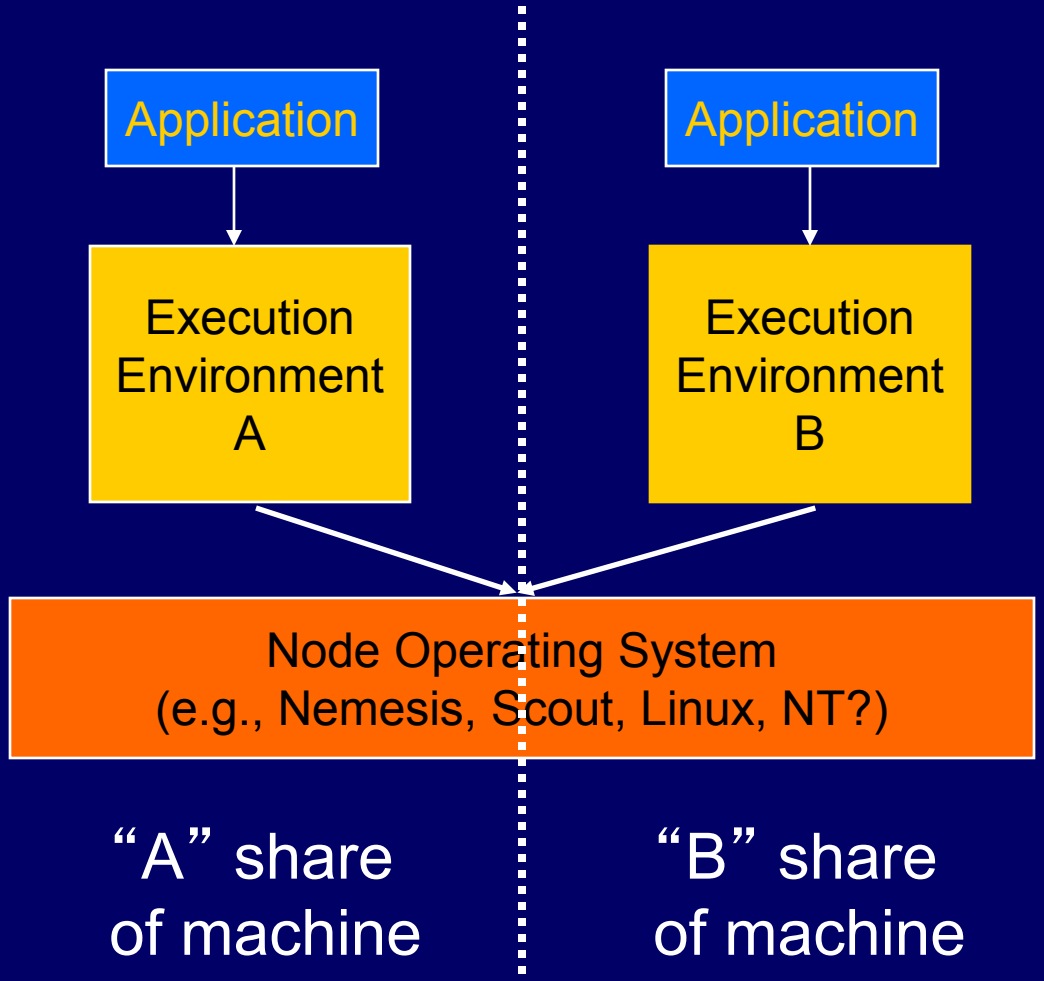
Example: SwitchWare Architecture



Resource Controlled Active Network Environment (RCANE)

- Manage CPU, Memory and Bandwidth
  - Challenge: Modern PL heaps (GC)
  - Challenge: Interrupts
  - Challenge: CPU/Mem/BW tradeoffs
- Approach
  - Nemesis for NodeOS + SwitchWare EE
  - See Menage, IWAN 99, Alexander, *et al.*, JCN, March 2001.

RCANE Vertical Architecture:

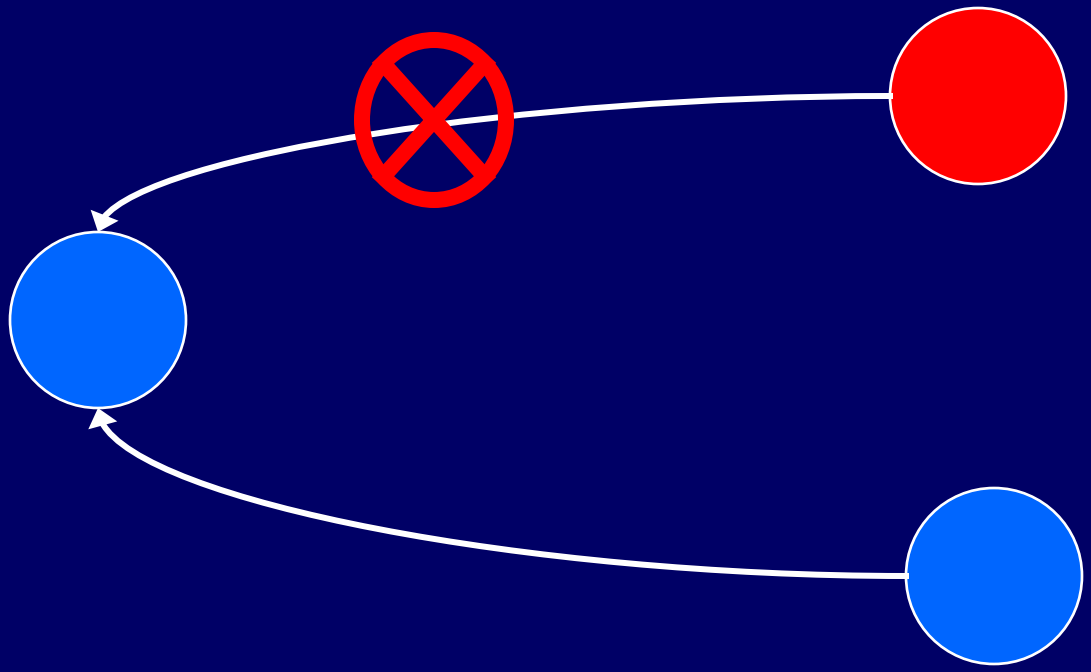


3.3 Network Based

- Nodes are interconnected to form the network
- Likely to interoperate nodes with conventional IP network devices, either as overlays or in management roles
- Distributed Resource Control rather than purely local, as in programming environment or in node

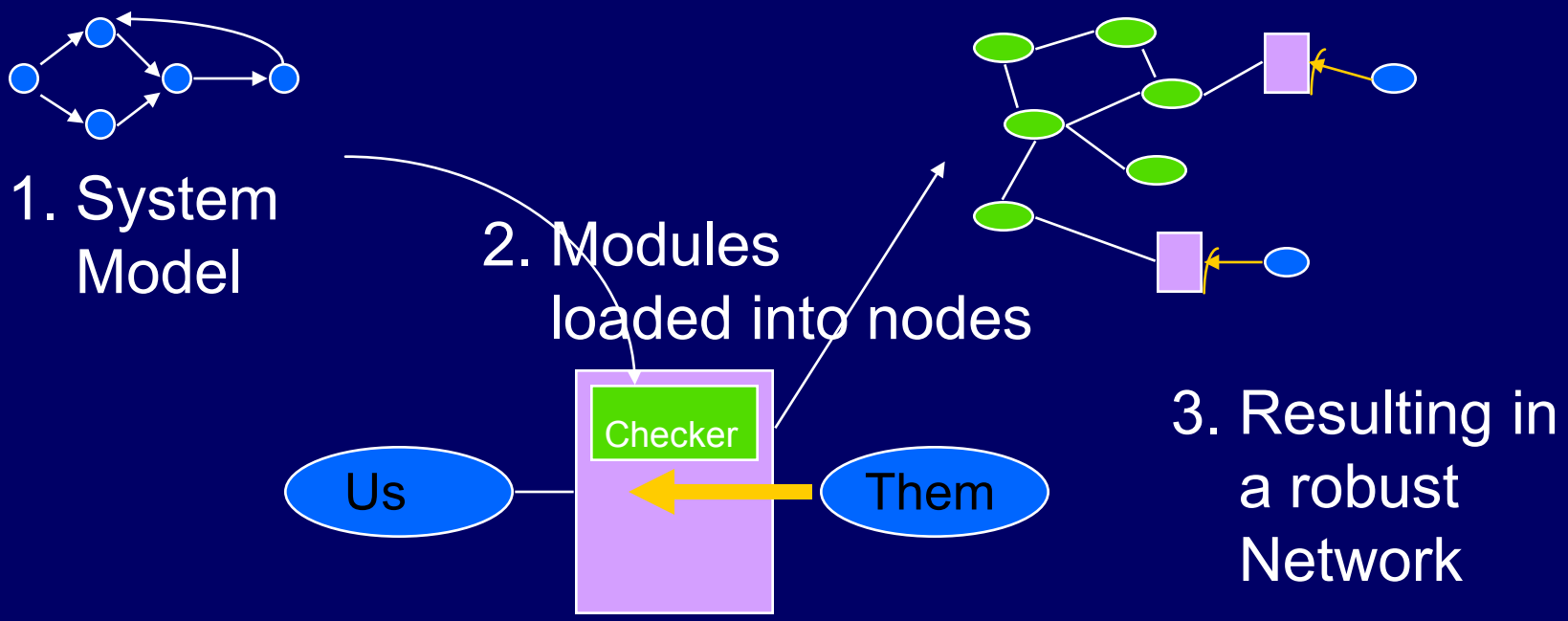
SANE Security Model

- Only process packets from trusted hosts

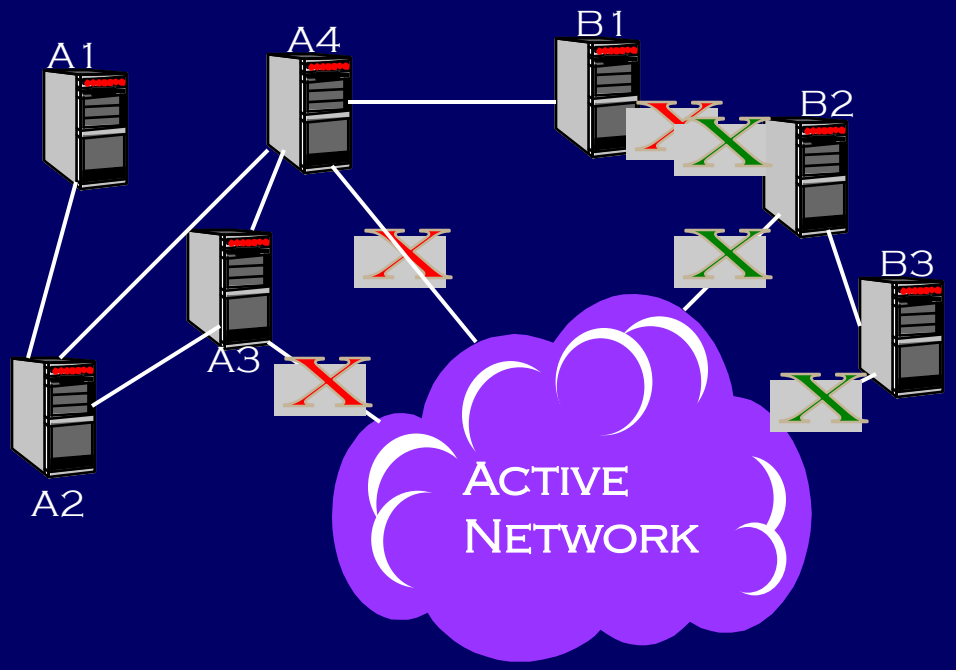


Model->Modules->Actions

- Syntax, Semantics, Node vs. Network
- Example: Securing a Network



Mutually Suspicious Nodes



- Nodes Authenticate their Neighbors
- Establish Trust Relations with Peers (PolicyMaker)
- Use Trust Relations to Solve Existing Problems (eg. Routing)
- Optimize Authentication

Node to Node Authentication

- Once at Boot Time, Periodically Thereafter (Crypto "heartbeat")
- Modified Station-to-Station Protocol (Well Known and Understood)
- Key Can be Used to Authenticate on a Hop-by-Hop Basis, Encrypt Sensitive Information
- Make Traffic Analysis Hard

4. Applications

- Firewalls
- Active Queue Management (AQM)
- FIRE
- Packet Marking
- Hash-Based IP Traceback

4.1 Firewalls

- Preceded organized A.N. effort
- Response to flaw in Internet scheme
- End-to-end argument results in network security dependent on hosts
- Hosts are insecure, ergo, net insecure
- Example: DDoS sources

Why a firewall is an A.N.

- Consists of filtering rules – see, e.g., the Berkeley Packet Filter (BPF) of McCanne and Jacobson
- These rules are composed of pattern/action pairs – e.g., IP source + destination address and port numbers, with actions such as drop, pass and log
- The programming is *event-driven*, in the style of PLs such as awk. Safe because limited.

Advanced Firewalls and A.N.

- Firewall technology widely used
- Would like "learning" for rule DB, action set
- Example: Intrusion Detection Systems (IDS) are string-matchers and event loggers
- Connect IDS to firewall policy for greater dynamics

Firewall Management

- Common policies needed at multiple firewalls
- One approach is careful manual configuration
  - Not scalable

4.2 Active Queue Management

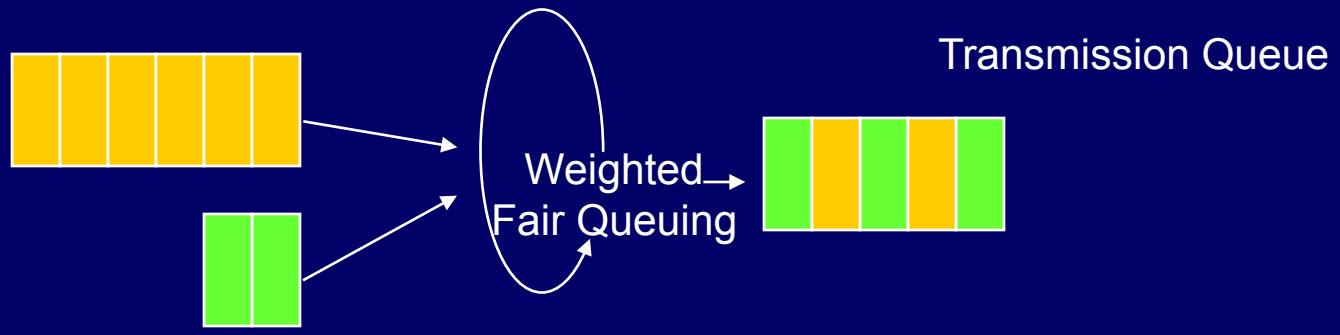
- Challenge of edge-provided congestion control – all edges must "play nice"
- Evil TCPs, or UDP lead to a problem
- Thus, network-embedded solutions have appeared as an attempt to solve these problems – called AQM in IP world
- Floyd and Jacobson RED, BLUE
- Stoica Core-Stateless Fair Queuing

Basic idea of AQM

- Packets assigned to queues in node
- When queue lengths hit "high water" mark, packets begin to be dropped
- Dropped at random location in queue
- Fairness, as "hogs" will be over-represented in queue
- Jeffay, *et al.* suggest tuning tough in SIGCOMM 2000 paper

Piecewise A.N. Node Solution: Loadable "Queue Management"

- Discriminates between "flows"
- Separate queue for each current flow
- Queues are serviced WFQ
- Control via RSVP, QoS Broker, etc.



4.3 FIRE

- Flexible Intra-AS Routing Environment
- Partridge, *at al.*, SIGCOMM 2000
- Link-State Intra-Domain Protocol
- Run-time reprogrammability
  - Information advertised
  - Routing Algorithms (by traffic class!)
  - All done in Java
- Can think of as an Active BGP

4.4 Packet Marking

- Savage, *et al.* paper in SIGCOMM 2000
- Approach to dealing with Denial of Service Attacks
- Basic idea is to (statistically) mark packets that go by

Probabilistic Packet Marking

- Routers periodically mark packets with their ID
- Only mark if packet not already marked
- Infrequent, since "slow path"
- Prob(mark)*#packets large with DOS
- Can be used to locate DOS source
- Improved by Song, Perrig in Infocom 01

4.5 Hash-Based IP Traceback

- Snoeren, *et al.*, SIGCOMM 2001
- Identifies originator of IP packet
- Generates audit trails within the network
- Routers enhanced with Source Path Isolation Engine – SPIE
- SPIEs contain packet digests

5. Summary

- Security of Active Networks themselves
- Use of AN techniques to secure IP nets
- Use of AN techniques to build more secure networks of all types

Security of Active Networks

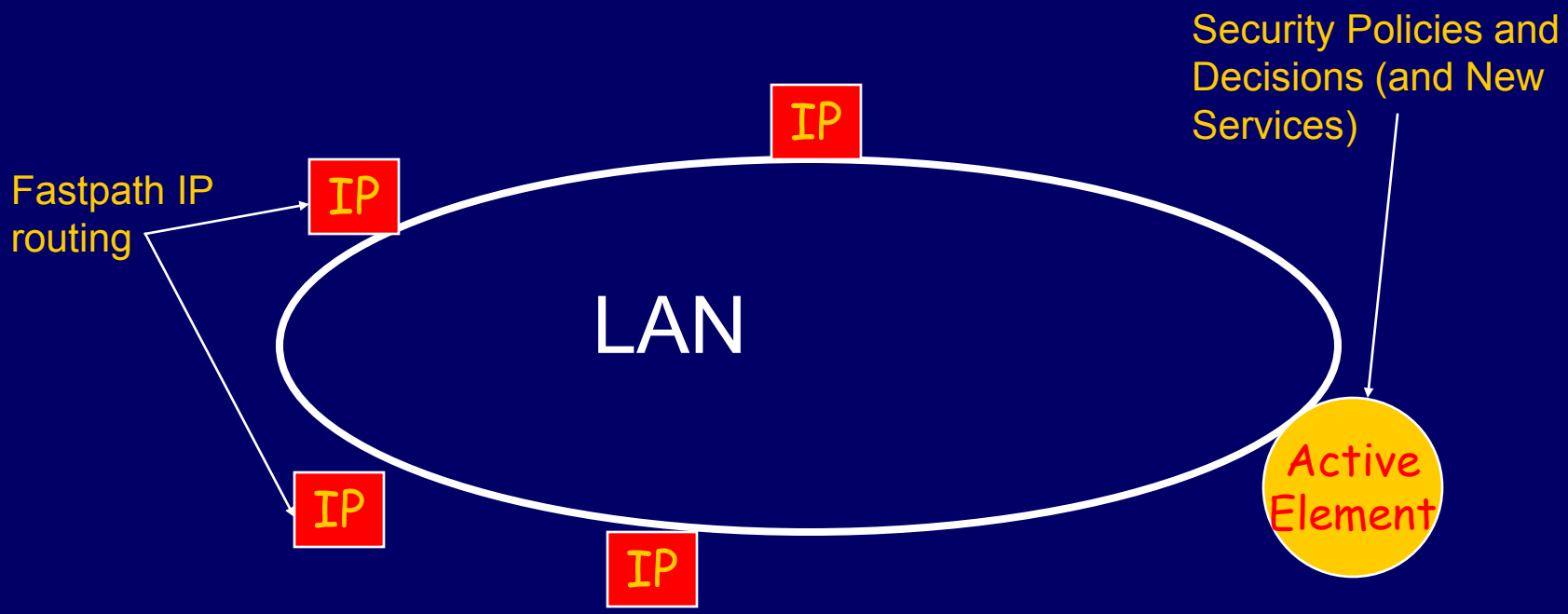
- Languages and Node Architectures show secure nodes can be built, for both active extensions and active packets (Alien, PLAN, SNAP, ANTS)
- Cryptography extends many node protections to networked uses
- Resources can be managed at both node and network levels – RCANE, NodeOS

Use of AN techniques to Secure IP networks

- Firewalls, NATs, more active follow-ons
- Programmable AQM (as with programmable BGP in FIRE)
- Packet Marking and IP-traceback techniques to locate attackers and attacks
- Active Router Control models

Active Router Control (e.g. FAIN)

- Routers co-located with Active Nodes



AN techniques for more secure networks of all types

- Begin to construct Internet overlays with provable properties such as SNAP resource bounds
- Many functions of "non A.N." devices are done in software (often see hacks of Cisco IOS in bugtraq)
- Migrate A.N. techniques into devices

Acknowledgements

- DARPA and NSF for high-risk funding
- D. Scott Alexander for ALIEN slides
- Jonathan Moore for SNAP slides
- Collaborators: Dave Sincoskie, Bill Marcus, Angelos Keromytis, Bill Arbaugh, Tony Bogovic, David Feldmeier, Dave Farber, Scott Nettles, Carl Gunter, Mike Hicks, Jon Moore, Kostas Anagnostakis, Stefan Miltchev, Paul Menage and Sotiris Ioannidis