Bowman: A Node OS for Active Networks

S. Merugu    S. Bhattacharjee    E. Zegura    K. Calvert

College of Computing; Georgia Tech
Department of Computer Science; U. of Maryland
Department of Computer Science, U. of Kentucky

Infocom 2000, Tel Aviv, Israel
Outline

• Background - Active Networking
• Bowman System Design
• Performance Measurements
• Configuration for Active Networking
• Concluding Remarks
Active Networking: What

• Programmable user-network interface(s)

• Control via:
  – mobile code in packets (capsules)
  – mobile code fetched from code repositories, based on packet header values
  – programmable signaling protocols
  – selection from set of fixed behaviors

→ Multiple execution environments
Active Networking: Why

- Faster deployment of new protocols and services
- Platform for research
- Services that exploit app and network knowledge
  - reliable multicast
  - application-specific congestion control, e.g., MPEG
  - network caching
  - network monitoring

→ High performance, access to low-level resources
DARPA Node Architecture

AN UNIs
Execution Environments
Node OS

ANTS
PLAN
... CANEses

IO ifs
CPU
memory
Bowman (and CANEs)
Bowman Design Goals

- Support per-flow processing
- Provide a fast path
- Enable a network-wide architecture
- Maintain reasonable performance
- Provide modularity and extensibility
- Leverage existing Host OS
Primary Bowman Abstractions

- Channels
  - communication endpoints
  - include protocol processing
- A-flows
  - computation
- State store
  - indexed by a unique key
  - includes named registries for data sharing between a-flows
Additional Components

- Dynamic extension mechanism
- Efficient packet classifier
  - match arbitrary number of header fields
  - returns first, all, or best match (with costs associated with each field)
  - dynamically extensible to different protocols
- Timers
- Network architecture via abstract topologies
Packet Processing Path

EE code

Bowman code

packet classifiers

input queues

output queues

cut-through

input channels

output channels

a-flow processing
Bowman Network Architecture

- Configure abstract links: endpoints plus protocol processing over physical topology (ALP)
- Select set of abstract links for virtual topology (ATP)
Performance Testing

Sun Ultra-5, 300 MHz
SunOS 5.7

100 Mbps

Bowman Node
Sun Ultra-2, 168 MHz
2 processors

100 Mbps

Sun Ultra-5, 300 MHz
SunOS 5.7

Compare to:
- Solaris kernel forwarding
- C gateway -- socket read/write of UDP segments
Forwarding Performance

Saturates 100 Mbps Ethernet for packets over 1400 bytes
Packet Processing Overheads

Bowman overhead relatively constant (~25 usec)
System read and write calls dominate processing time
Effect of Real-time Scheduling

Comparison of time-sharing (TS) to real-time (RT) mode
Three kernel threads: input, a-flow, output
Configuration for AN

• Monolithic approach
  – EE creates exactly one a-flow that subscribes to all packets addressed to EE
  – EE manages own resources

• Multi-a-flow approach (CANEs)
  – EE creates one control a-flow used for EE signaling and management
  – New a-flow for each user’s packets
  – Bowman schedules user computation
Selected Related Work

• Router plug-ins (WashU)
  – integrated EE (customizable IP) and NodeOS
  – NetBSD kernel modifications

• Janos (Utah)
  – Java-based NodeOS

• Extensible routers (Princeton)
  – Scout-based NodeOS
Future Work

• Security mechanism
• Resource management
• More complex output queueing disciplines
• Scalable topology instantiation

• EE-developers toolkit to run over DARPA NodeOS implementations?
a-flow processing

EE code

Bowman code

input queues

output queues

packet classifiers

input channels

output channels

cut-through