

Active network vision & reality: lessons from a capsule-based system

David Wetherall
CSE, U Washington

Big Picture

- Wetherall did his PhD at MIT – thesis work on introducing services to the network
- Aiming for two benefits
 - Enable new applications that leverage computation in the network
 - Accelerate pace of innovation by decoupling services from infrastructure (IPv6 rollout)
- Reports on progress-against-goals instead of here's-my-new-active-network-system

Big Picture

- Three main results described
 - Programming model (capsules)
 - Security and protection in the model
 - What type of applications will the model allow you to build
- Programming artifact (ANTS)
 - Java-based system for building active nodes

What is ANTS?

- Architecture for building customized network services (Java – based)
- 3 Parts to ANTS:
 - Programmable routers are “*active nodes*”
 - Soft store – Code cache
 - Active nodes receive and process “*capsules*”
 - Extension of IP packet – conventional IP header followed by capsule header
 - “**API**” for use by active nodes for network interaction and service development
 - environment queries, soft-store manipulation of service-defined objects, routing

ANTS components

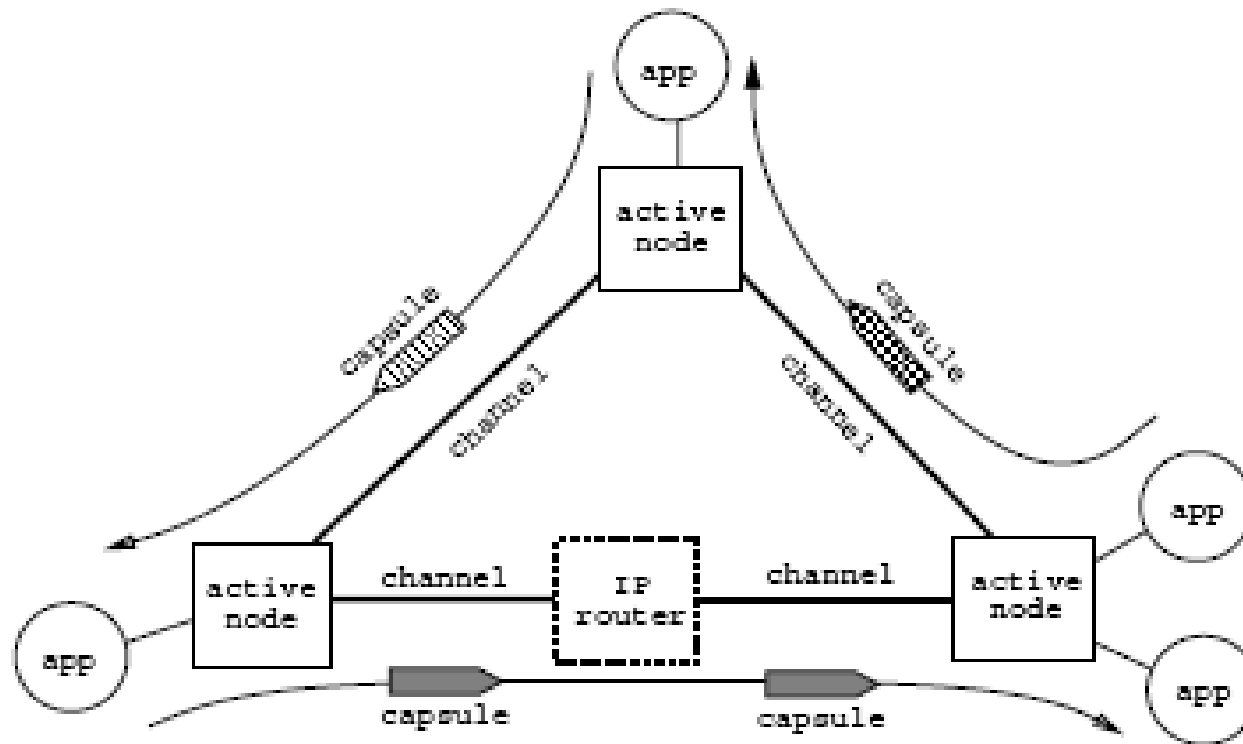


Figure 1. Entities in an ANTS active network

Capsules

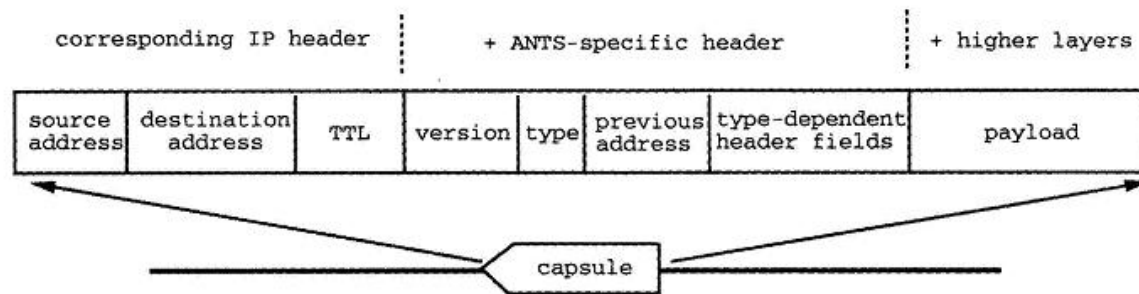


Figure 2. Key features of the capsule format

Method	Description
int getAddress()	Get local node address
ChannelObject getChannel()	Get receive channel
Extension findExtension(String ext)	Locate extended services
long time()	Get local time
Object put(Object key, Object val, int age)	Put object in soft-store
Object get(Object key)	Get object from soft-store
Object remove(Object key)	Remove object from soft-store
void routeForNode(Capsule c, int n)	Send capsule towards node
void deliverToApp(Capsule c, int a)	Deliver capsule to local application
void log(String msg)	Log a debugging message

Table 1. Active node API

Capsules

- Originally extension of IP packet format
 - ANTS toolkit uses UDP datagrams in overlay with capsule as payload
- Capsules are typed
 - Type is an MD5 fingerprint
 - Identifies the code that should handle the capsule
 - Types can have descendant types

Active nodes

- Receive and route capsules to correct forwarding code
- Capsule type field used for demux to Java class
- Demand-loaded & cached if not already present
 - One-way correspondence between capsule and code
 - Code carried by reference, not by value
 - saves state, exploits locality, code-distribution orders-of-magnitude more expensive than regular processing
- What about nodes that aren't “active”?

Demand - loading / code distribution

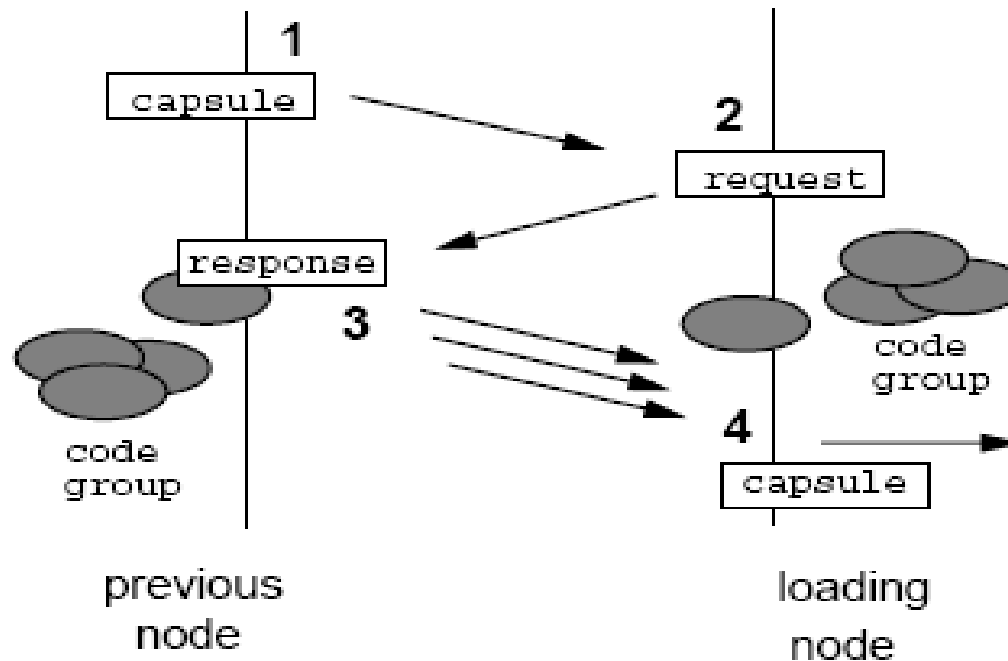


Figure 3. Demand loading of capsule code

End user software supplies code to nodes at edges, then this mechanism is used within the overlay.

Capsule feasibility

- Are capsules feasible?
- What does the question mean?
 - This is *software routing*
 - Not only that, there's computation involved
- Can software routing of capsules be done quickly enough to keep up with line (network) speed?

Capsule feasibility

- What are current router capabilities?
 - PC – 70K packets/sec (100Mbps Tput)
 - Cisco +4 Router of Doom – 2.4Gbps Tput – 4M packets/sec
 - Hard to predict composition of network, but...
 - Today there are programmable network devices that can keep up with nGbps line rates, and still leave headroom on packet path.
 - ANTS architecture allows core routers to be not active

Capsule feasibility

- Performance tests
 - 2 operations – code distribution and capsule forwarding
 - Code distribution is at a coarser granularity and relies on caching to work.
 - Active node Latency degrades linearly with capsule size (this is good)
 - Other costs are artifacts of Java (C would be faster) and user-level implementation (in-kernel would be faster)

Security and Safety

- Who can introduce new services? – two basic concerns
 - Protection – can my service damage yours (intentionally or not)
 - Resource management – can my service consume arbitrary resources

Protection threats

- Corrupt the ANTS runtime at a node?
 - Java sandboxing (other implementations possible (PCC, SFI))
- Code spoofing (incorrect or corrupted code)?
 - Capsule type is a MD5 fingerprint of the code – no way to spoof unless you can break MD5
- Corrupt soft-state at node?
 - Soft-state also controlled by fingerprint
 - A class can only access its own soft-state
 - State sharing through hierarchy of fingerprint types

Resource management threats

- A capsule forces unconstrained resource consumption at a single node?
 - Watchdog timer for node runtime
 - Capsule TTL fields
- An application floods the network with capsules, starving other apps?
 - the Internet doesn't address this either

Resource Management Threats

- A capsule consumes large amounts of resources across a subset of the network?
 - By creating other capsules and routing them
 - Internet is static; ANTS is not
 - Per capsule hop limit?
 - Can still inflict damage
 - Some behavior is OK (multicast) – hop limit not related to topology
 - Punt (rely on certified code \Rightarrow limits ‘any user can introduce new code’ idea)

What types of applications can be built?

- Ones that are difficult to deploy in the Internet
 - not tied to administrative setup, which tends to mirror physical setup
 - services whose implementation is spread along the datapath
 - make use of network level info: topology, loss, load...
 - variants of routing, forwarding, flow setup, congestion handling
- Characteristics
 - Expressible – restricted API
 - Compact – limit of 16KB
 - Fast – node runtime is limited, no blocking
 - Incremental – not all nodes are active nodes
- Network layer services tend to be best for this, rather than application code