Active Policies in Knowledge Hyperspace
Intelligent Agents and Policy-Based Networking

Takeo Hamada, David C. Blight and Peter J. Czezowski

Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94086, USA

{thamada,dblight,peterc}@fla.fujitsu.com

Abstract

Policy-based networking (PBN), or policy-based management systems (PMS), is gaining popularity in enterprise network management due to its flexibility and a proposed architecture which enables management systems to access and accumulate network knowledge and policies in a database (directory service). In this paper, we study the limitations of the current PMS, and propose an extension of the current architecture named Active PMS (APMS). APMS is augmented with active policies, which are intelligent agents in knowledge hyperspace, interacting with lower level static policies and simple network management protocol (SNMP) and/or common information model (CIM) based network elements. Active policies do have drawbacks. One critical problem is policy conflicts caused by feature interactions, which is a part of the inherent nature of an agent-based management style. In this paper, we introduce a type lattice for policies, from which we derive the class hierarchy and visibility control of APMS implementation in object-oriented languages. The proposed architecture concepts are exemplified by an example of end-to-end QoS provisioning, where the hybrid architecture of active and static policies enables a natural migration from current generation PMSs to APMSs.

Keywords: *PBN, DEN, PMS, Active policy, Intelligent agent, APMS, Knowledge hyperspace, Type lattice, QoS, Active/static policy hybrid*

1. Introduction

Policy-based networking (PBN) has been attracting significant industry interest. PBN has been developed in response to the difficulty of managing IP based networks, which are based on a distributed control and management model. The flexibility of IP networks has been its major strength. The difficulty of managing IP networks comes from the flexibility given to the network and its users. Successful management needs to manage, or at the very least be aware of, a myriad of devices and services, many of which are from distinct layers in the OSI network model. Policies are the specification for the network operation, which may include policies for users and security, desktop and OS policies, and network operation policies. For successful end-to-end management of a network, the policies affecting network operation must be consistent with user and desktop requirements. To achieve this goal, there must be consistent modeling of requirements (policies), and network information.

The Internet Engineering Task Force (IETF) has been developing standards including: lightweight directory access protocol (LDAP) [LDAP], common open policy service protocol (COPS) [COPS], policy schema [Schema], policy framework and architecture [Framework], differentiated services schema [DiffServ], and IP security schema [IPSec]. These standards are supported in many of the PBN products from various vendors.

Although most PBN products and standards are still primarily limited to the enterprise domain (LAN oriented), the practical merits and the feasibility of PBN concepts are applicable to the WAN oriented public management. In comparison with the traditional public management concept such as TMN, PBN offers a more flexible, a customizable management solution, allowing each router/switch to be configured consistently and utilize network information available in the directory service. Networks may be configured with site-specific user and application knowledge.

This flexibility does not come without a cost. One issue is security for which we proposed a solution based on role-based access control (RBAC) [Hamada98]. Another issue is scalability. The original PBN is designed for a single domain network environment. Therefore, scalability to large networks has not been addressed. Though caching and other techniques will be

employed, more clear provisions for scalability issue for large-network/WAN deployment are sought. We have proposed a scalable PBN architecture based on a hierarchical abstraction principle [Blight99]. In this paper, we investigate a new methodology, called active policies in knowledge hyperspace, which is an integration of intelligent agents and PBN.

The remainder of this article is organized as follows. Section 2 provides background, including brief discussions of the current framework and schema standardization efforts, and the roles of agent technology in policy based networking. This section also introduces the active policy based management system. The concept of a knowledge hyperspace is described in the third section. Some implementation issues are considered in Section 4, including a load balancing example to illustrate the mechanisms required by mobile agents, discussion of the role of non-deterministic in their behavior, and the introduction of a type lattice for deriving policy class hierarchy. Section 5 contains an application example for an end-to-end QoS provisioning problem. Section 6 describes a prototype implementation of an active policy-based management system on an experimental network. Finally, Section 7 presents our conclusions and discusses possibilities for future work.

2. Background

This section discusses the current framework and schema standardization efforts, the roles of agent technology in policy based networking, and introduces the active policy based management system (APMS).

2.1. Current PBN Architecture (IETF)

The current PBN architecture, shown in Figure 1, employs a schema based upon the directory enabled network (DEN) initiative of the DMTF [DMTF]. Policies are entered into a policy management tool. This tool is responsible for collecting policies, determining their validity, and detecting policy conflicts (those conflicts that are detectable). Policies may be stored in a LDAP compliant directory service. The policy server retrieves policies from either the directory or directly from the policy management tools. The policy server interprets the policies, and configures individual devices or services within the network through a protocol such as COPS. It is important to note that not all elements in the framework need to be in separate devices.

In addition to configuring devices, the policy architecture also needs to be able to read network information from devices. If a policy server for QoS policies is configuring DiffServ queuing mechanisms, it must first be aware of the queuing capabilities of the devices it is configuring. There must be bi-directional communication between devices and policy servers.

The strength of policy architecture is measured by the strength of its information model. The policy schema defined by the IETF serves a good general schema upon which more specific policies may be derived. The use of LDAP as the standard policy access protocol makes efficient policy representation difficult, as the simple attribute-value pairs supported by X.500 do not easily represent the complexity of general policy specifications. The policy specifications may be fairly complex, and as such fairly difficult to parse and interpret. Many Internet devices do not want to incur the cost of supporting the full policy schema, so simplified specific policy specifications may be used for specific classes of devices. This is also important where the interpretation of the original policy specifications requires global network knowledge.

The current policy schemas are very simple in scope. The core schema defines a set of policy conditions and a set of policy actions. When a condition is met, network action is executed. The current policy schemas are focused on single domain issues, and static configuration of devices.

This is one of the main reasons we think the current

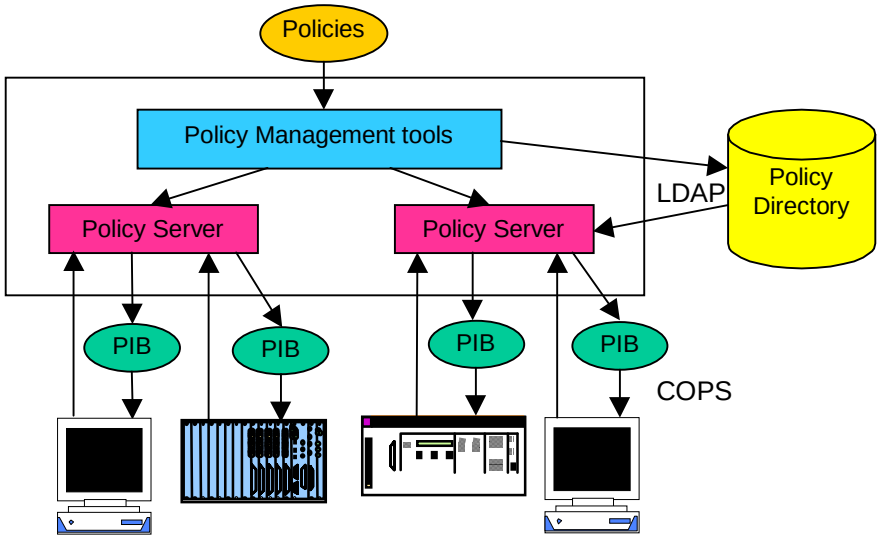


Figure 1. IETF policy architecture.

PBN architecture, despite its hype, is able to address only a fairly limited domain of issues; those can be translated into fixed configuration settings. For example, QoS issues often need complex interactions between relevant network components such as network measurement, fault management, configuration settings, where real management intelligence is taking place. The limitations of the current PBN architecture prevent building intelligent management solutions.

One possible solution, which we propose in this paper, is to augment the policy with a more semantically capable entity, called active policy. An active policy is essentially an agent, which may contain internal states and code to execute.

2.2. Policy Agent and Management Architecture

The term agent is overused with at least two different meanings in the context of network management. In the first sense, an agent is an "agent" for a stakeholder in the network environment. In the TMN manager-agent paradigm, the agent represents the management interests of the "manager", who is an network operator. There are other cases, such as QoS Broker [Nahrstedt95] and, Bandwidth Broker [Reichmeyer98], and Web robots. Though they are not explicitly called agents, a part of the system represents a stakeholder such as end-user and an ISP in a different domain, to negotiate resources or to obtain information on web pages. Active networks [Smith99][Brunner99] can be seen as a type of agent system in the sense that active packets are sent on behalf of end applications, typically for the network resource provisioning.

The second meaning of agent implies a new computational/management paradigm, with the expectation that the collective behavior of agents will realize a more distributed, self-organizing style of management [Resnik94][Forrest90]. Although there are overlaps, this line of thought emphasizes more on collective intelligence (dynamics) of agent systems than the intelligent behavior of individual agent. Expected architectural merits are: (1) scalability through distributed network management, (2) robustness through distributed storage of network knowledge over agents, (3) flexibility through agent control, in adapting to network evolution. Bieszczad et al. provide a survey of potential uses of mobile agents in network management, with examples of several actual and potential applications within the framework of the five functional areas of the OSI network management model [Bieszczad98].

One common attribute of all agent definitions is autonomy. The agent should be able to perform its task without requiring decision support from the user. Once an agent is dispatched, it should have sufficient knowledge resources to make local decisions. Communication with the user is normally limited to control functionality, and reporting of results.

The current PBN standards define policies as specifications and schemas, with which the system can hold the desired operational behavior of a network. The current standards are focused primarily on device oriented policies, which can be applied directly to resource objects within the network. For example, QoS policies may define DiffServ priority classes, in which multiple classes of service exist within a network. One class may represent high priority traffic, and another low priority. The QoS Policy for the network may simply state that certain resources are allocated for the high priority class, with the remainder used for the lower priority class. For example the high priority class may be configured as having 1 Mbps reserved on each network link. Note that this is not the only way to specify QoS policies, we could use other policies such as 50% bandwidth is reserved for high priority class. The QoS policy can be directly applied to each link, by configuring the output port on routers with the same policy. This policy can be implemented and guaranteed by ensuring the routers are properly configured. Such policies can be referred to as device centric.

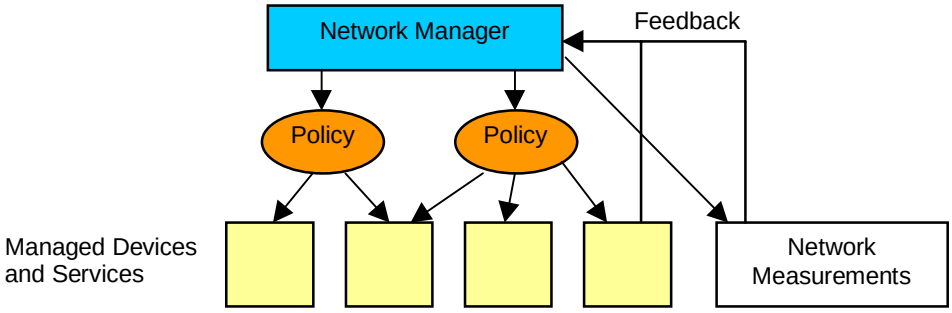
While the above policy would represent an improvement in the Internet QoS, it does not fully support the level of control we would like to have. A more useful QoS policy might state that the network is required to provide specific QoS to specific applications. An example would be that each IP telephony application should receive 16 Kbps, each web application 100 Kbps, each Video stream application 1 Mbps, and all other best effort. These QoS requirements can not be achieved by applying the policy to each network device. Such a policy can be referred to as network centric.

A network centric policy needs to be interpreted in the network management system of a network. The network management system needs to retrieve network resources and capabilities from the network, and with this knowledge, decide upon the appropriate device centric polices which will implement the network centric policy. The management system is essentially acting a policy synthesis system for device specific policies. Note that the device centric policies may be synthesized from multiple network centric policies.

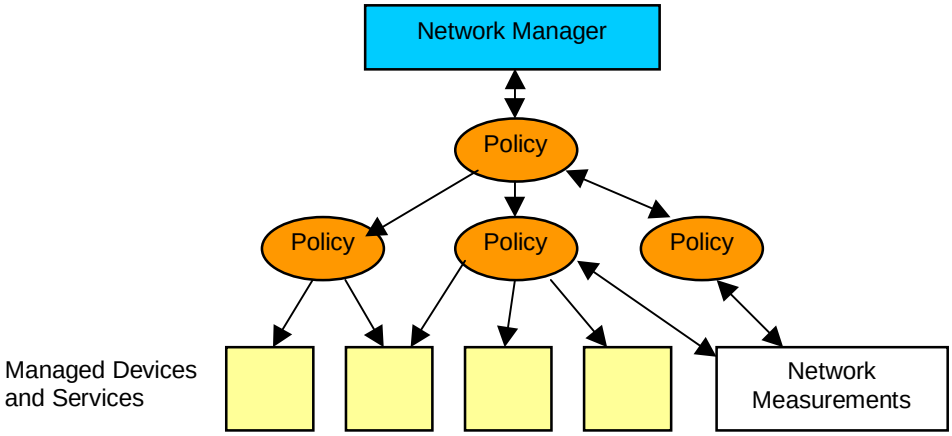
With network centric policies, the policy becomes an active part of the network. Feedback from the network should be processed by the policy, and depending upon the state of the network, new device specific policies synthesized. In this model of network policies, the policies must become active parts of the network.

2.3. Active PMS

Figure 2 illustrates the differences between the conventional PMS and our proposed APMS approach. Intelligence provides adaptability to rapid changes in the environment. Although human operators will not be replaced, enhanced system adaptability and proactive management are the major goals of the active policy-based management system (APMS).



(a) Policy based system.



(b) Active policy based system.

Figure 2. Policy based management systems.

Management tasks are largely divided into two groups. The first group has deterministic objectives and reasoning processes, where configuration changes can be pre-computed. Monitoring of fixed number of devices and setting alarm conditions for these devices, setting DiffServ parameters on routers belong to this group. For this group of tasks, the current PMS provides a sufficient, if not ideal, framework.

The second group of tasks has characteristics opposite to the first group. This is due to the fact that the dynamic nature of the network can not be known exactly real-time, and the information is statistical, not deterministic, with a certain degree of inaccuracy. Take an example in network measurement. Network measurement is the basis of proactive network management, e.g. load balancing, active caching, etc. Accurate network measurement is in general hard. Even if it is possible, it can be too costly, or the information may not be available in real-time.

APMS addresses this second group. Active policies act upon knowledge hyperspace, which contains agents with application specific views, in such a way that active policies act on network elements via collections of agents, and also the policies retrieve network information from the agents.

Active policies can be seen from at least three distinctively different perspectives. These distinctive views may be also seen as three evolutionary steps of policy-based networking. In the first view, a policy represents a set of parameters which dictates particular behavior of network devices such as routers and bridges. This is strictly operational view, and policy representation is so device-specific at the physical level, adding mobility to policy does add little value, if any.

In the second view, a policy is a piece of management knowledge, which can be applied more generically to management tasks such as QoS provisioning, CPE configuration management etc. Knowledge representation, therefore, needs to be more device-independent and abstract, so that the policy can be generically applicable to common network management issues. Adding mobility to this policy makes sense, as an active policy can be applied to a particular situation, e.g. network congestion, so that the policy can be called upon when the need arises.

In the third view, a policy is most akin to rules in the rule-based expert system or a rule in the context of traditional artificial intelligence, where are a set of rules are designed to solve a complex management problem. Knowledge representation is at the most abstract level,

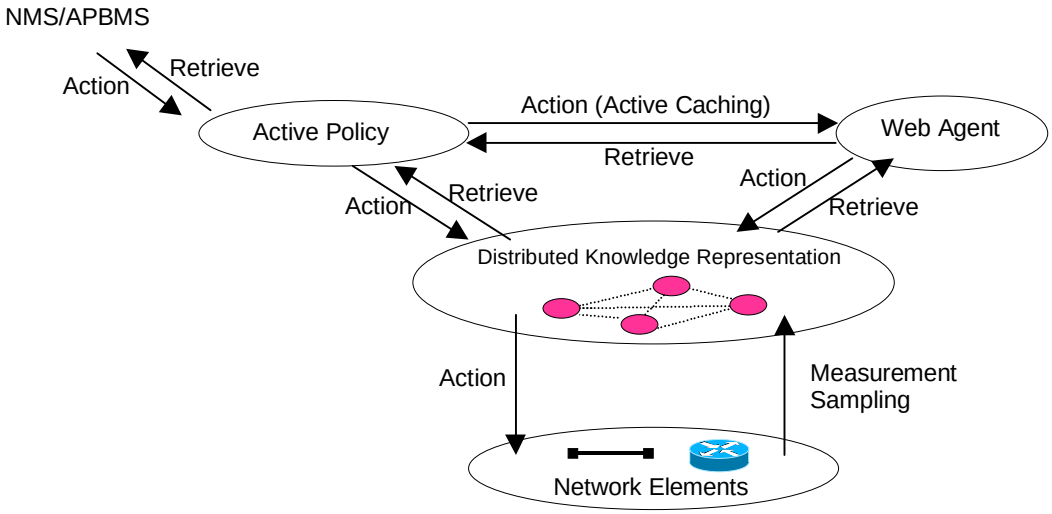


Figure 3. Knowledge hyperspace.

so that the rules can handle the problems at the most generic and general level. The power of abstraction, however, can both serve as advantage and disadvantage for the overall effectiveness of the management system.

For example, QoS provisioning can be modeled at several different abstraction levels. At the lowest level, policy can carry a separate set of parameters for each individual router or bridge along the path, so that the devices can reserve some bandwidth for a given set of IP addresses. It is also possible to use RSVP parameters to do the same, given that the devices support RSVP. At the most abstract level, a policy is written in such a way that it understands the QoS provisioning at a high level, information model as those given in DMTF CIM [DMTF] and QoS policy information base (PIB) [PIB]. These abstract policies are translated on the fly to more device-specific parameters, to control actual network devices.

By comparing these three views on policies, it is clear that the benefit of policy mobility, i.e. active policy, is the largest when the level of abstraction of policies is the highest. In other words, the design of successful APMS depends on the success of network abstraction and resultant schema of network information model.

3. Knowledge Hyperspace

It is a classical statement repeated in the history of computer science that the schema (data structure) and the policy (algorithm) should be made for each other, at a proper abstraction level. In a sense, the relationship between the knowledge hyperspace and the active policy is a re-telling of the same story. An active policy makes practical sense, only when it works at a proper abstraction level, i.e. a knowledge hyperspace, whereas the practical meaning of knowledge hyperspace is determined only through a set of active polices which work upon the knowledge hyperspace (see Figure 3 above).

This somewhat mutually dependent definition of terms is probably inevitable, and we can even extend the notion a step further that two sets of agent share a knowledge hyperspace, when these two sets interact with each other, and exchange some information between them.

The implication of the above statement is the following. It is possible to see, and it is often beneficial to see, that the whole set of network resources and active policies as if they consist of loosely connected knowledge hyperspaces, which interact with each other through management knowledge carried by active policies.

Knowledge hyperspace is a loosely connected set of different agent groups. These agent types have different methods and knowledge representation, and they play a plug-able or dynamically expandable part of the whole knowledge hyperspace. Since each network application, e.g. e-mail, requires application specific knowledge, these application specific issues are handled by special agent groups.

Each agent collects and stores network information in a distributed manner, while walking through the nodes in the network. They may have accesses to network elements directly, or may take advantage of available simple network management protocol (SNMP) management information bases (MIB) or common information model (CIM) MIBs [DMTF] to collect raw network data.

In Figure 3, Web Agent, which performs load balancing among nodes in the network (Section 4, Figure 4), are able to communicate with active policies and distributed knowledge in the network, which is represented in SNMP/RMON MIBs in the network. For example, an active policy collects node performance data from the node RMON MIB. Based on the samples obtained, the active policy may make a decision, so that a Web agent starts or stops active caching. Web agent

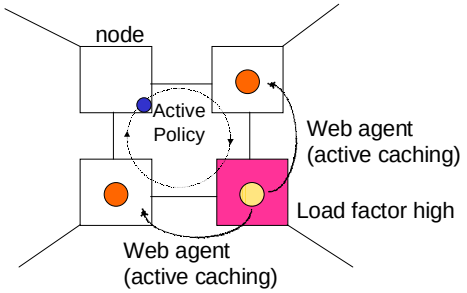


Figure 4. Dynamic load balancing using active policies.

can also makes decision on its own, based on statistical data on hit counts of destination URL addresses. These destination URL addresses may be recorded by HTTP proxy nodes.

These data are statistical samples, and are also prone to statistical errors. There is no guarantee that the information of each agent is consistent with the others, and chances are higher that they are not. For load balancing, however, it does not pose a big problem, since load balancing does not require high measurement accuracy.

Agents can communicate with each other by passing messages while sharing some common placeholder. Active policies, which are agents, can communicate with agents in the knowledge hyperspace using the agent-to-agent communication mechanism to implement policies and to retrieve information from agents. These active policies also serve as a medium between NMS or APMS and other agents in knowledge hyperspace. In other words, the representation scheme of the active polices will be translated into those used in agents in the knowledge hyperspace when they act or retrieve.

4. Implementation Considerations

This section presents some issues to be considered when implementing an active policy-based management system.

4.1. Load Balancing Example

The example shown in Figure 4 above illustrates how active policies work in practice. Each node represents a computational platform, where agents can spawn, stay, move, meet, interact with each other, and die. An active policy is an agent that is injected from the managing APMS. The active policy moves around the node cluster, checking to see if the policy specification associated with the policy is being met.

A load-balancing agent may move through a network. At each node, the agent checks the availability of system resources. If there exists only insufficient resources (for example the CPU load factor is above a threshold), the agent may cause application agents residing on the node to move to other nodes.

In the example shown above, web agents represent application processes on the network. The policy agent detects a high load factor on one node, and web agents are evicted from that node. The web agent processes on the new nodes copy caches from the high load factor node to the new nodes, re-starting active caching on the new nodes. When the high load factor period ends, the two outstanding web agents can be consumed back to one web agent process in the original node. In pseudo code, the active policy should look like:

```
do_loadBalance() {
   if (load factor is high)
      evict application agents,
      restart in neighbors;
   else
      consume application agents in the neighbors,
      restart at this node;
}
```

When the system is not overloaded, i.e. processing time is linear to the load, it is clear to see that the average response time of the system is inversely proportional to the active policy density, i.e. more active policies the shorter the expected response time.

The simplest load-balancing agents would simply examine load factor (or other resource), and act when a threshold is reached. A more proactive agent should ideally balance the load with respect to other nodes. This could be accomplished by having the agents keep an average load factor value for a finite set of previously visited nodes. In this case the agents would keep the network at a much more constant overall load factor.

4.2. Non-determinism in APMS

The Active policies in a APMS are all operating independently, autonomously, and in non synchronized manner. As a result the APMS is a non-deterministic management system. This characteristic has both merits and demerits. The positive side is that the behavior of the APMS may be dynamically and incrementally changed, by injecting new types of active policies, and possibly evicting the old ones. Non-determinism also can be designed in the agent behavior, such that the agent system covers the entire problem space in randomized samples.

The advantage of non-random, deterministic behavior is that you can precisely specify the behavior of the agents. It is possible to design the system so that tests a performed in a predetermined order. The power of random agents is that we can give up precise control for autonomous, self-adapting behavior. A simple example is an agent which performs a simple localized test network. The agent can be configured with a precise course through a set of nodes, in which all nodes will be tested at regular intervals. If a new node is added, the course is modified accordingly. An alternative approach is to model each agent as a random walker, whose course through the network is not predetermined, but decided locally after each test. The advantage of the randomness is that no configuration of the agent is

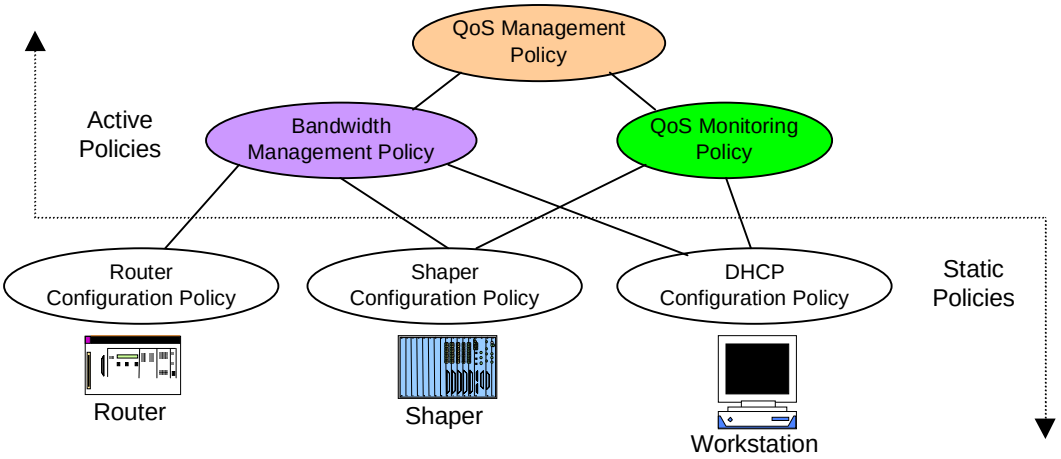


Figure 5. Type lattice of policies.

required, and yet the agent will visit each node (although not at regular intervals). If a large number of agents are active, the aggregate behavior will likely resemble the deterministic behavior.

The negative side of non-determinism is that it may lead to an unpredictable behavior, caused by the non-determinism built-in the APMS and the agents. In particular, active policies may conflict with each other when one set of agents (active policies) affects the behavior of the other set. In a severe case, mutually inconsistent or exclusive policies may co-exist in the same knowledge hyperspace, making the system behavior unstable. For example, suppose there are two active policies, one requesting 100Kbps for the marketing department and another requesting 1Mbps for a video conference. When there is a critical link in the path that can not grant both requests, two policies become mutually incompatible, the resources can be deadlocked. In a lesser degree, two policies can interfere with each other either directly, or indirectly by side-effects of their operations, leading to hard-to-detect policy violations.

One way to approach this problem is to introduce a more systematic way to regulate control flow of active policies, in such a way that potential hazards can be detected, and more systematic checking of policy interference can be done in order to avoid deadlock situation.

The issue of conflicts in policies has been extensively studied. The unfortunate reality is that there is no general purpose method by which policy conflicts can be determined before execution. The problem of conflict detection is analogous to the halting problem in computer systems. If we limit the expressive power of policies, it may be possible to create a system with detectable policy conflicts, however such a system may not be powerful enough for a general purpose IP network management system. It is possible that policy conflict analysis can be done for a limited subset of management problems such as QoS management. Although not all conflicts can be detected, there will be common policy conflicts, which may be easily detected. We still recommend the use of policy conflict detection techniques, however we do not fully rely on this approach.

If policies conflicts do not have the potential to crash a network, but merely affect performance, or non-management access, then we can let the active policies detect conflicts through monitoring of policy compliance. A typical example of policy conflicts is a user with multiple roles. If John Doe is a manager in the engineering department he may be allocated resources based on his manager status, and his engineering status. If these policies are non-consistent, and conflict occurs. In a multi agents policy system, an agent will exist for each policy applicable to a role. No matter which order the resources for John Doe are allocated (by manager agent first, or by engineering agent first), each agent will monitor policy compliance. If the policy conflict causes at least one of the policies to be violated, the violation will be reported to management system. This will require operator correction, and a policy conflict can not generally be resolved without operator input. In fact automated correction could likely hide a policy violation, which should have been fixed.

4.3. Type Lattice of Policies

The type lattice of policies represents resource and behavior dependency among policies. Policies higher in the lattice can change the status, change the parameter settings, or create a new instance of lower policies, in a way akin to file write permission. Figure 5 illustrates that the QoS management policy is on the top, followed by the bandwidth management and QoS monitoring policies. Lower policies are non-active policies, as those supported by the current PMS. The bandwidth management policy has permission to manipulate the router, shaper, and DHCP configuration policies to perform network resource provisioning utilizing these lower management policies.

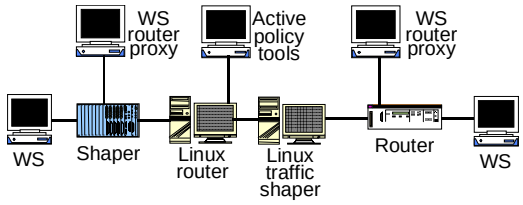


Figure 6. APMS prototype implementation on an experimental network.

The lattice also shows that the bandwidth management and QoS monitoring policies are potentially in contention for the shaper and DHCP configuration policies. The contention may not actually occur, but even if it occurs, it may be avoided. For example, by splitting the shaper configuration policy into two non-overlapping sets, or by disallowing the QoS monitoring policy to change the shaper configuration policy, conflicts can be avoided.

It should be noted, however, that the lattice only gives coarse-grained characterization of the structure of active policies. Even if the lattice has no cross-links, implying that there is no potential for contention between different class of active policies, the class may contain competition within, leading to feature interactions. In the previous example, two bandwidth management policies are in competition. Unless a precedence relation exists between the two requesters (marketing dept. and conference), some agent synchronization mechanism is required to handle the two competing transactions.

5. End-to-end QoS Provisioning Example

The type lattice of policies extends naturally into a class hierarchy, in such a way that the upper policy treats the lower policies in the lattice as its own sub-classes. For example, both the bandwidth management and DHCP configuration policies can be made subclasses of QoS management policy, so that these subclasses expose private or protected interfaces only to their respective super-class.

Combining active and static policies, the architecture is illustrated in a end-to-end QoS provisioning example (see Figure 6 above). The procedure starts with a QoS management policy activated at the workstation on the left. This active policy, migrating toward other workstations on the right, spawns active bandwidth management policies on the nodes such as shapers and routers on the way, or interacts with them when they already exist on the nodes. At the end workstations, the QoS management policy spawns and implements a QoS monitoring policy, such that negotiated QoS are observed at the end workstations. At each node, active policies such as bandwidth management and QoS monitoring take advantage of static policies from the LDAP policy directory to resolve device dependent configurations, thus keeping the active policies at a high level. Note that the type lattice keeps the feature interaction between the active policies to a minimum.

Bandwidth management policies, in particular, manage the limited bandwidth resources, thus indirectly interacting with each other and themselves, affecting their behaviors. When bandwidth is assigned for best-effort traffic or DiffServ style streams, however, this feature interaction does not pose a problem. For IntServ, the operation of the bandwidth management policy needs to be atomic, to guarantee end-to-end availability of bandwidth.

6. Active Policies Implementation

We are currently prototyping an Active policy implementation on a experimental IP PBN network. The network consists of workstations, routers, traffic shapers, and management systems. The network contains both a Cisco 7200 series router, and multiple Linux/PC systems with QoS routing, DiffServ functionality enabled, and IPSec VPN services. A Structured Networks IPath100 traffic shaper is also included in the network.

Windows and Unix based workstations, and Linux based routers and traffic shapers can easily be adapted to support active policies, by installation of a Java virtual machine (JVM). Devices for which software modification is not supported (Cisco and 7200 and IPath 100), can be managed by active policies through a proxy service on attached workstation.

Policy management tools are located on one of the network workstations. These tools allow policy specification compatible with current IETF draft standards, and emulate the standard IETF PBN architecture. The tools are currently being augmented to support active policies by creating a policy agent with each specified policy. Each Java based active policy agent is very simple because the policy parsing and interpretation are performed by the policy management tools before agent creation. The agents are simply encoded with specific tasks and agent control, mobility, and communication methods. Selection of Java as implementation language was a natural one when this prototype project was started, due both to the ready availability of Java development tools, and to the language features Java offers. Visibility control mechanism Java offers, i.e. public and private at class level, import and export at package level, are crude, but they provide the basic control mechanism for the type lattice separation realization.

The policy management system supports two types of active policies: internal and external. External policy agents are agents that are dispatched to nodes in the network. These agents are used to monitor QoS compliance and other measurable specifications within the network. Internal agents are those that exist in knowledge hyperspace. The policy management tools read in network information and forms an internal representation of the network. Internal agents are allowed to move in this domain. Active policies for

configuration will normally exist as internal agents. They will determine a configuration for the network (complete or incremental) based on the internal network model. Once the configuration is complete, policy servers are notified to implement the configuration. External agents can then be dispatched to measure compliance.

Figures 7 and 8 show screen shots from an active policy prototype, using Java-based simulator currently being tested on the prototype network. In this simulation, a hypothetical enterprise network with WAN connections is used, of which the prototype network of Figure 6 will form one leg of the hypothetical network. The objective of this simulation is that the expected benefits of active policies, in particular scalability and incremental network intelligence enhancement, can be realized. It is not our objective to measure the actual performance of active policies in this simulation.

In this hypothetical example, a single enterprise domain is connected by multiple WAN connections, including a private line and VPNs through two different ISPs. The configuration is determined by factors such as cost, reliability, and QoS requirements. In this example, when active policies (internal agents) are activated, they visit four gateways (Head-office, Europe, NA, Asia) in an ordered manner, so that the WAN configuration satisfies the enterprise connectivity requirements. Once such a configuration is achieved (in this example, a mesh WAN connection between four gateways), external agents are spawned to monitor the network performance.

7. Conclusion and Future Work

In this paper, we defined an active PMS architecture and presented its architectural components. We defined the active policy as an intelligent agent, working in loosely connected knowledge hyperspaces. Non-determinism in intelligent agent-based systems can introduce both merits and demerits into active PMS. The major causes of the non-determinism were analyzed, and a mechanism to control agent interactions, which is named type lattice, is proposed. This type lattice is extended to design agent class hierarchy and visibility control in object-oriented languages.

To illustrate the proposed architecture, an example for end-to-end QoS provisioning using active policies is explained. Throughout the example, it is shown that undesirable feature interactions are both minimized and controlled by following the proposed architectural principles. The example also shows that the active policy principle is applicable to wide class of services such as best-effort, DiffServ, and IntServ, with a few additional considerations on active policy deployment. The example also shows that active policies and static policies are mixed in the hybrid architecture, enabling to see the active PMS as a natural extension of the current generation PBN.

For further work, we envision that active policy interference and feature interaction are critical issues for the applicability of APMS. We are currently studying deployment aspects of APMS. Though Java provides natural candidate for its wide availability and rich language features, we are investigating April [McCabe99] for its superior features of agent-to-agent communication.

References

[Bieszczad98] A. Bieszczad, B. Pagurek, and T. White, "Mobile agents for network management", *IEEE Communications Surveys*, v. 1, no. 1, pp. 2-9, 1998.

[Blight99] D. Blight and T. Hamada, "Policy-Based Networking Architecture for QoS Interworking in IP management", *Proceedings of IM'99*, Boston, May 1999.

[Brunner99] Brunner and R. Stadtler, "The Impact of Active Networking Technology on Service Management in a Telecom Environment", *Proceedings of IM'99*, Boston, May 1999.

[COPS] J. Boyle, et al., "The COPS (Common Open Policy Service) Protocol", IETF Internet Draft, draft-ietf-rap-cops-06, Feb. 1999.

[DiffServ] Y. Snir, Y. Ramberg, J. Strassner, "QoS Policy Framework Information Model and Schema", IETF Internet Draft: draft-snir-policy-qos-infomodel-00.txt, Oct 1999.

[DMTF] Distributed Management Task Force, http://www.dmtf.org

[Forrest90] S. Forrest ed., *Emergent Computation*, MIT/North-Holland, 1991.

[Framework] M. Stevens, W. Weiss, H. Mahon, B. Moore, J. Strassner, G.Waters, A. Westerinen , J. Wheeler, "Policy Framework", IETF Internet Draft, draft-ietf-policy-framework-00.txt, October 1999.

[Hamada98] T. Hamada, "Role-based Access Control in Telecom Service Management", *Proceedings of APNOMS'98, pp.293-304*, Sendai, Sept. 1998.

[IPSec] J. Jason, M. Jeronimo, "IPsec Policy Schema",IETF Internet Draft: draft-ietf-ipsec-policy-schema-00.txt, Oct 1999.

[LDAP] M. Wahl, S. Kille, and T. Howes, "Lightweight Directory Access Protocol (v3)", IETF RFC 2251, Dec. 1997.

[McCabe99] F. G. McCabe and K. L. Clark, "April - agent process interaction language", *Intelligent Agents*, M. Wolldridge, N. Jennings, eds., LNAI, v. 890, Springer-Verlag, 1995.

[Nahrstedt95] K. Nahrstedt and J. M. smith, "The QoS Broker", *IEEE Multimedia*, Vol.2, No. 1, pp.53-67, Spring 1995.

[PIB] M. Fine et al., "Quality of service policy information base", IETF Internet Draft, draft-mfine-cops-pib-01.txt, June 1999.

[Reichmeyer98] F. Reichmeyer, L. Ong, et al, "A Two-Tier Resource Management Model for Differentiated Services Networks", IETF draft, draft-rotzky-2-tier-management-00.txt, Nov. 1999.

[Resnik94] M. Resnik, *Turtles, Termites, and Traffic Jams*, The MIT Press, 1994.

[Schema] B. Moore, E. Ellesson, and J. Strassner, "Policy Framework Core Information Model", IETF Internet Draft, draft-ietf-policy-core-info-model-01.txt, October 1999.

[Smith99] J. M. Smith, et al., "Activating Networks: A Progress Report", *IEEE Computer Magazine*, pp.32-41, Apr. 1999.

Takeo Hamada graduated from the University of Tokyo with BE and ME degrees in Electrical Engineering in 1984 and 1986, respectively. He received his Ph.D in Computer Science from UCSD in 1992 for a research in physical VLSI design. He has been with Fujitsu since 1986. From 1995 until the end of 1997, he was with the TINA-C core-team at Red Bank, NJ, where he engaged in research on service management and resource management architecture in Telecommunication Information Network Architecture (TINA). He has been with Fujitsu Laboratories of America since 1998. His current research interests include network management, service management issues in IP network, Policy-Based Networking. He currently serves as TINA-C service management WG leader.

Dr. Blight is currently a member of research staff at Fujitsu Laboratories of America, leading research on Policy Based Networking in the Network Computing Research group. His currently research interests include network management, policy based networks, QoS and network reliability in Internets. Dr. Blight also participates in international standard organizations including Internet Engineering Task Force (IETF), Desktop Management Task Force (DMTF), and the OpenGroup. Prior to joining Fujitsu labs of America, Dr. Blight was an assistant professor at the University of Manitoba. Dr. Blight introduced an industrial relevant graduate level course in telecommunication systems, and has been actively involved in the Internet Innovations Center. Dr. Blight was also Staff Professor with Telecommunications Research Laboratories (TRLabs) in the Data Networking and Software Laboratory in Winnipeg. Dr. Blight holds a Ph.D. from the University of Manitoba, with bachelors and masters degrees in electrical and computer engineering. His approach to network management is to emphasize the how to handle the complexity and changes in modern networking systems.

Peter J. Czezowski joined the Network Computing Research Group of Fujitsu Laboratories of America, Inc. as a Member of the Research Staff in March 1999. He holds a M.Sc. in Electrical Engineering (1994) from the University of Manitoba, Canada, where he is currently a Ph.D. candidate. His research interests include policy based network management, agent based computing, and the Internet. He is a member of IEEE.

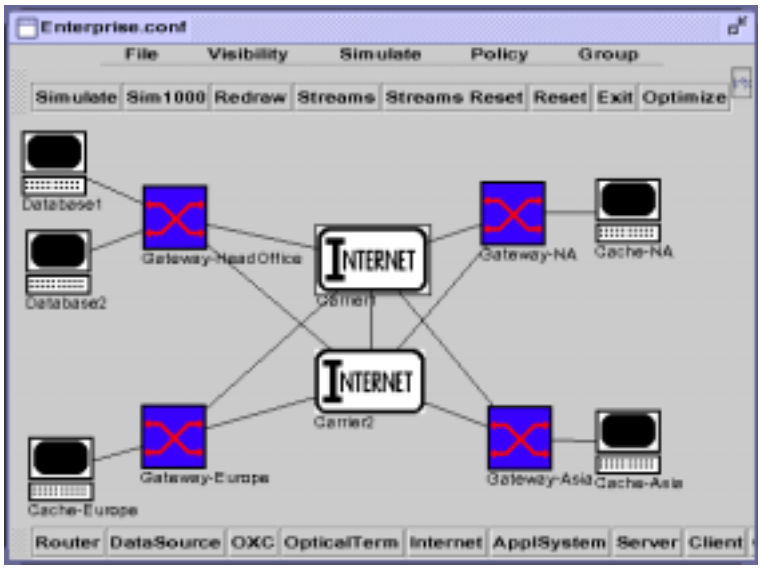


Figure 7. Screen shot (1): before WAN configuration policy is applied to the network

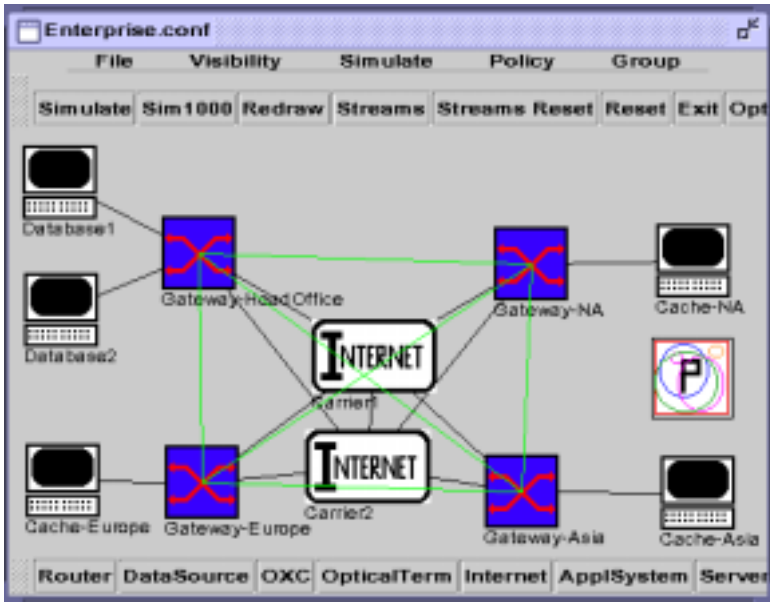


Figure 8. Screen shot (2): after WAN configuration policy is applied to the network.