Automated Service Provisioning in Heterogeneous Large-Scale Environment

Ibrahim Khalil and Torsten Braun
Institut für Informatik und angewandte Mathematik (IAM)
Neubrückstr. 10
3012 Bern, Switzerland
ibrahim,braun@iam.unibe.ch

Abstract

With the increasing complexity of network management activities due to naturally limited human involvement, carriers and service providers are looking to migrate from manual, static provisioning models to the more dynamic service-oriented automated provisioning models to meet customer demands for rapid service turn-up and obtain more customers and maximize revenue opportunities. We propose a novel distributed architecture where highly mobile and intelligent agents can take the responsibilities of not only provisioning, but also configuration audit management in a timely fashion. Although recent research have focused on using mobile agents for network monitoring or simple push-based device configuration in a distributed architecture, their ability have not been exploited in dynamic IP service provisioning. Using a simple push-based model to configure network services while ignoring dependencies among configuration elements may easily lead to configuration inconsistencies resulting in failure or inefficiencies. In this paper, we have taken a new approach to configuration modeling that is device neutral and based on which any existing or emerging IP services can be presented by encapsulating service semantics, including service-specific data. We have developed new mobile intelligent provisioning and audit agent architectures that use the knowledge built upon configuration dependency modeling. Examples of intelligent agent are presented to complement the proposed management architecture.

Keywords

Dynamic Service Provisioning, Service Management Architecture, Distributed Management, Configuration Modeling, Mobile Agent, Intelligent Agent, Automated Configuration, Automated Device Auditing.

1 Introduction

In large scale Internet service deployment scenarios Internet Service Providers (ISPs) and large enterprises often face the daunting task of provisioning huge number of network devices in a short period of time and continue ongoing device management. Manual provisioning methods and manual element interconnection across multiple discrete elements leaves a high margin for errors. Carriers and service providers are, therefore, looking to migrate from manual, static provisioning models to the more dynamic service-oriented automated provisioning models to meet customer demands for rapid service turn-up and obtain more customers and maximize revenue opportunities.

Existing Network management architectures ([19]) deploying SNMP [25] as the management protocol do not really fit well to suit the needs of automated provisioning in a large scale environment due to some well known limitations. Not only the centralized approach of SNMP has severe scalability limitations, but it is also a less powerful for making modifications to the network. SNMP Management Information Base (MIB) implementation for new emerging services like IPSec [13], MPLS [9], DiffServ [4] do not even exist in most vendors equipments so as to enable SNMP `set` command to dynamically configure these services. Attempts have been made ([24], [17], [18], [12],[21]) to address the scalability problem by decentralizing processing and control to distribute processing load and reduce the traffic around the management station. Most of the efforts have been made by using intelligent and mobile agents [11] that are able to perform management functions by carrying code, instructions,data and executing tasks on any network node. However, while addressing only scalability issue the vast majority of these agents [20], [1], [10], [6], [3], [8] have been mainly used in traffic analysis, fault management, network monitoring and performance management.

Despite the ability of mobile agents, in the area of network device configuration management [7], [2], [21] and automation they have played only limited role. In this regard, they mostly have been found up-loading management scripts that were compiled at machines hosting agents [22]. In reality, many of the dynamically created services at least partially depend on the current configuration states of the target devices. Using a simple push-based model to configure network services while ignoring dependencies among configurations elements may easily lead to configuration inconsistencies resulting in failure or inefficiencies. This often neglected but important issue of configuration modeling was however addressed in [27] with limited use in useful and complex IP service creation.

In this paper, we have taken a new approach to configuration modeling that is device neutral and based on which any existing or emerging new IP services can be presented in abstract form by encapsulating service semantics, including service-specific data. We have developed a new intelligent mobile provisioning agent architecture that uses the knowledge built upon configuration dependency modeling. We have also proposed a hierarchical look like high level network management architecture that has a central policy and data repository and deploys several intelligent agents playing the role of managers. The agents can take the responsibilities of not only provisioning but also perform ongoing device management often termed as *auditing*. The basic objective of audit management is to ensure that the expected configuration as stored in the policy repository is equivalent to the actual configuration in a physical device. This is necessary because accidental manual mis-configuration or configuration performed by any external process other than the automated system can easily result in unexpected behaviors of network devices. Intelligent audit agents periodically sent to visit network devices have procedures in place to reestablish lost service. Several real world examples of intelligent provisioning are presented to show the applicability of our approach. The rest of the paper discusses all in details.

2 Automated IP Service Provisioning Architecture

2.1 Provisioning Requirements in Large Scale Environment

As today's network infrastructure continues to grow and new IP services (e.g. VPN, QoS, MPLS) emerge, the ability to manage an increasing network complexity is considered as crucial for deployment. Growing trend also rise among corporate customers to out-source such complicated management services to Internet Service Providers (ISP) not only to avoid the

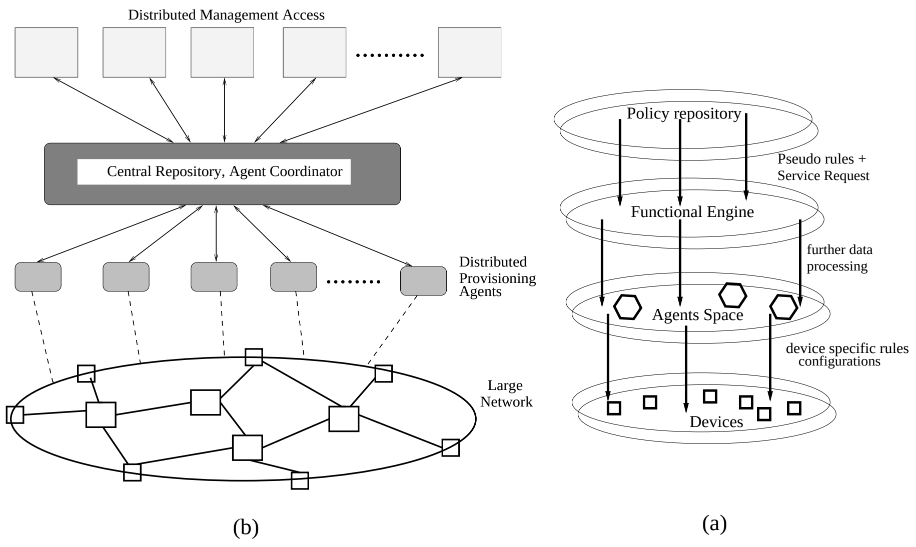

Distributed Management Access
Central Repository, Agent Coordinator
Distributed Provisioning Agents
Large Network
(b)
Policy repository
Pseudo rules + Service Request
Functional Engine
further data processing
Agents Space
device specific rules configurations
Devices
(a)

Figure 1: (a) Proposed Network Architecture, (b) Translation of device independent meta-data to device-specific configurations by Intelligent Provisioning Agent

complexities of management of those services, but also for economic reasons. Although this opens the possibility of a tremendous business opportunity for ISPs, they also see challenges in deploying and managing the network services efficiently and cost effectively. It is not unlikely that a single ISP might need to timely manage thousands of network devices. The effective way to meet such challenges is to provide automated service provisioning. Managing devices not only means one time service activation for customers, but also providing ongoing device management often termed as audit management. Ongoing device management is needed for several reasons such as:

- Services configured by human administrators many lead to mis-configuration or cause conflict with existing services activated via automated system.
- If network devices crush due to power failure or for some other reasons, valuable configuration information that were thrown into the devices might well be lost partially or fully.
- One can not rule out the possibility of an inside or outside intruder changing some configuration settings.

In summary, configuration performed by any process other than automated system can easily result in malfunctioning of the managed services in network devices. If the Service Level Agreement (SLA) states that should any such problem occur that it would deal within x minutes of the occurrence, then the automated system would need to send a delegated entity capable of fixing the problem in a time period close to x. That delegated entity, if we term that as an intelligent agent, should have the knowledge and ability to perform audit management in an appropriate manner. Such agents will need to understand the complete dependencies in a configuration storage of a network device while provisioning a new service or modify an existing one.

2.2 Proposed Architecture

The proposed hierarchical look-like network management architecture shown in Figure 1(a) addresses the provisioning requirements mentioned above in a large-scale environment. In such an environment, with a large number of different types of network devices, service providers will need to deploy a network architecture that can use huge number of automated agents acting as managers to provision and perform on going device management and provide distributed management access to the large number of potential customers that want to create and modify services dynamically on demand. The proposed architecture simply aims to achieve this. It comprises several management interfaces, a central repository system with agent communicator, and several service provisioning agents mainly performing the role of managers. Having several management interfaces not only facilitates dynamic service creation by the very customers that out-source services to ISPs, but also allows several system administrators to manage a large enterprise or ISP network from a network operations center or distributed locations.The agent coordinator performs no management tasks, but only delegates service requests to the provisioning agents. Therefore, the agent coordinator does not become the processing bottleneck.

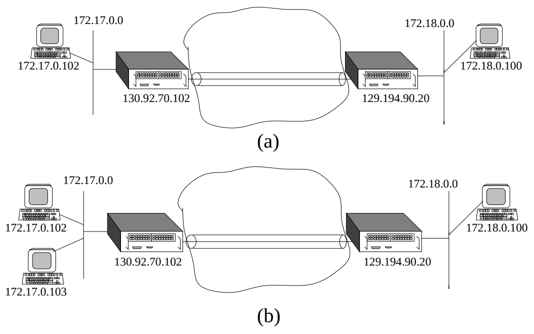
The service-specific intelligent provisioning agents build a generic service environment that enable customization of service creation and control of network resources. By encapsulating service semantics, including service-specific data and the logic to interpret the data in service provisioning agents we can provide an abstract network interface, separating services from the underlying network. As shown in Figure 1(b), these agents are able to translate user requests and pseudo rules extracted from data/policy repository into device-specific configurations to automate service provisioning process.

By using intelligent provisioning agents we address both scalability problem and lack of ability of traditional network management approaches to automatically provision a very large network. The scalability problem is well handled by the agents' ability to decentralize processing and control, and, distribute processing load to reduce the traffic around the management station. Provisioning agents are able to filter and process data locally at the network nodes without the need for transmission to the Top Level Manager (TLM) like agent coordinator. In fact, traditional distributed management architectures ([19] do not help much to alleviate the scalability problem as management repository is partitioned and replicated, and thus require another protocol to maintain the consistency of distributed repositories. Unfortunately, perfect consistency is impossible to achieve making such approach less effective in time-sensitive distributed provisioning.

In this architecture, by using mobile agents we have not only provided decentralization, but by embedding intelligence we have also added dynamism to the agents that are able to perform configuration and audit even in hostile situation when actual device state might be different from expected device state. We build intelligent provisioning agent as an autonomous software entity embedding mobile code, configuration data which has the capability of moving itself in the network and executing onto specific network nodes. On a network node, the provisioning agent can analyze and retrieve local configuration information, install and execute code, take decisions, thus leading to fully decentralized network management activities. This obviates the need for MIB implementation in the network devices. None of the previous works [20], [1], [10], [6], [3], [8] have considered performing such level of complex provisioning and audit management.

3 Configuration View in Network Devices

Dynamic IP service creation in network devices relies on the fundamental assumption of having an accurate and consistent abstraction of the underlying network, particularly a view of essential parts of configuration information stored in those devices spread across different device-specific repositories in different formats. The complexity of large IP networks and the paucity of commercial configuration tools means that this is often an unrealistic assumption. Hence, populating a service model must be closely coupled with checking for



```
crypto isakmp policy 1
 hash md5
 authentication pre-share
 lifetime 500
crypto isakmp key GENEVA-BERN address 129.194.90.20
crypto ipsec transform-set ah-md5-hmacANDesp-des ah-md5-hmac esp-des
crypto ipsec transform-set ah-md5-hmac ah-md5-hmac
crypto ipsec transform-set ah-sha-hmac ah-sha-hmac
crypto ipsec transform-set esp-encryption esp-des
 crypto map genbern 160 ipsec-isakmp
 set peer 129.194.90.20
 set transform-set ah-md5-hmacANDesp-des
 match address 140
interface FastEthernet0/0
 ip address 130.92.70.102 255.255.0.0
 no ip directed-broadcast
 traffic-shape group 150 1000000 100000 100000 1000
 crypto map genbern
access-list 140 permit ip  host 172.17.0.102 host 172.18.0.100
access-list 150 permit ip  host 130.92.70.101 host 129.194.90.20
```

(c)

Figure 2: (a) Setup for VPN Network (b) Expanded Setup for VPN Network (c) Partial IOS Configuration File of Cisco Router 130.92.70.102

possible configuration mistakes. Relationships between different configuration elements are implicit with repositories containing replicated and interdependent configuration information leading to inconsistency. Many of the dynamically created services at least partially depend on the current configuration states of the target devices. Using a simple push-based model to configure network services while ignoring dependencies among configurations elements may easily lead to configuration inconsistencies resulting in failure or inefficiencies.

In this section, we present a new device neutral approach to configuration modeling and based on which any existing or emerging new IP services can be presented by encapsulating service semantics, including service-specific data. We present a generalized view of configuration information in a network device and try to model it as clusters of configuration elements. For this purpose, we show examples of IPSec VPN configurations (Figure 2) in a widely used Cisco IOS device. Our previous experience [15], [14], [5] with VPN provisioning and also complexity of VPN configurations in various commercially available devices are the main motivations behind using these specific examples here.

Basically, no widely accepted standard for network configuration exists, and SNMP, in reality, simply remains a tool for network monitoring. Access mechanisms and manipulations of configuration information in device repositories are largely vendor-specific. While many devices are accessible via TELNET/FTP, others need to be accessed via HTTP or LDAP. The configuration modeling process is independent of the device access mechanism. However, the intelligent agents built upon the knowledge derived from configuration modeling use various device access drivers (i.e telnet driver, HTTP driver, etc.) as required.

3.1 Configuration Element

The configuration element is a customizable object with several attributes in a configuration space (i.e repository) of a network device. Configuration space, which we will describe later, represents the complete configuration information of a single device. The attributes of a configuration element reflect, how it would behave when embedded in a service and also how they can be linked to other elements to create a service. The attributes of a basic configuration element as shown in Figure 3(a) are:

- *body*: identifies the type of action a configuration element performs.
- *hook*: configuration elements are usually linked to each other in a configuration space. A hook helps to attach one element to the other.

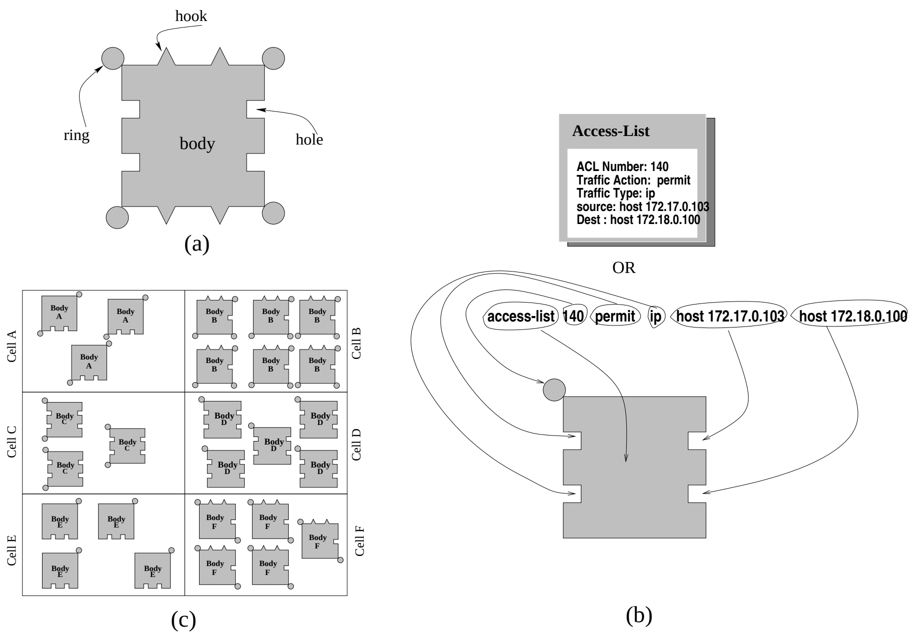


Figure 3: (a) Configuration Element and its Attributes (b) Modeling a Configuration Element in a Network Device (c) Mapping Configuration Space

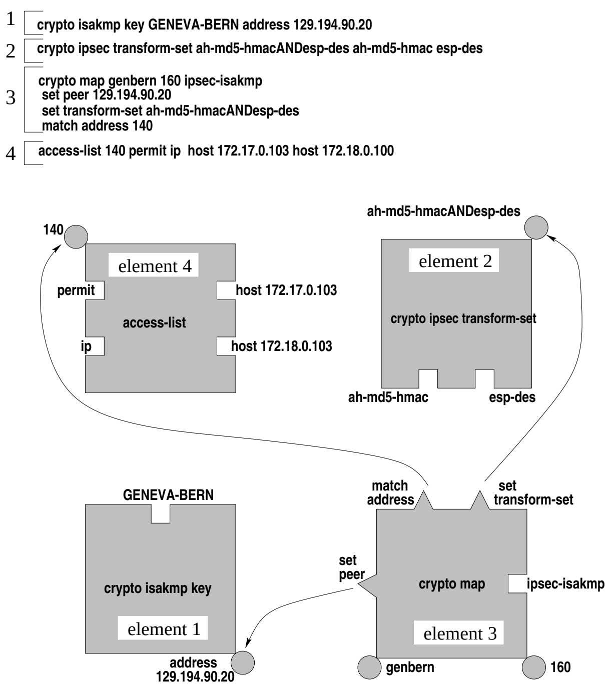
- *ring*: is the identifier of a particular configuration element and it is that part to which hook of another element attaches in order to create a composite element or service.
- *hole*: defines the behavior of configuration element. Holes are filled up with parameters that actually determine the behavior.

Figure 3(b) shows an example of modeling a basic configuration (non-composite) element. This particular element classifies traffic based on source and destination addresses. Clearly, the body of the element is `access-list` as this identifies what it does (i.e classifies traffic). `permit` (action on traffic), `ip` (type of traffic), `host 172.17.0.102` (source) and `host 172.18.0.100` are the parameters of several holes of the element and determine its behavior; `140` uniquely identifies this element, i.e. acts as a reference to this. The element in the device can be viewed as a simple text line or can be presented via a GUI. Regardless of how it is viewed in heterogeneous devices, we can always model an element with the same sets of attributes.

3.2 Composite Element

A composite element is a collection of several basic configuration elements. This is created by hooking an element to the ring of another.

Figure 4 shows an example of modeling a composite configuration element relating to IPSec tunnels configuration in device 130.92.70.102 for the setup shown in Figure 2(a). Three non-composite configuration elements with body labels `access-list`, `crypto iksamp key` and `crypto ipsec transform-set` linked to another element having a body `crypto map` that binds them with the hooks and create a composite element basically defining the tunnel configuration part in peer device 130.92.70.102.



```
1  crypto isakmp key GENEVA-BERN address 129.194.90.20
2  crypto ipsec transform-set ah-md5-hmacANDesp-des ah-md5-hmac esp-des
   crypto map genbern 160 ipsec-isakmp
3   set peer 129.194.90.20
    set transform-set ah-md5-hmacANDesp-des
   match address 140
4  access-list 140 permit ip  host 172.17.0.103 host 172.18.0.100
```

Figure 4: Example of Modeling a Composite Configuration Element

3.3 Composite Service

When one or more basic or composite configuration elements are bundled together to perform a network service, the bundle is called the composite service, partial or full, depending on the nature of the service. For example, traffic policing based QoS configuration in an edge device can be a complete composite service while for a VPN tunnel service creation two network devices will have to configured. Therefore, in a VPN tunnel service creation each device will have partially complete composite service. The example shown in Figure 4 is a partially complete VPN composite service in network device 130.92.70.102.

3.4 Configuration Space

Figure 3(c) shows the whole configuration information of a single device that can be logically mapped to a configuration space viewed as clusters of cells, where each cell contains one or more of the configuration elements. However, a cell consists of only homogeneous configuration elements. For example, all `access-list` configuration elements as shown in Figure 2(c), can be viewed to be part of a cell having body tag `access-list`. Similarly, all the configuration elements having the body `crypto ipsec transform-set` are parts of another cell in the configuration space.

4 Intelligent Agents for Service Provisioning

The goal of our automated provisioning architecture is to develop distributed collections of intelligent software agents that operate mostly independently to perform a variety of service-specific configuration information retrieval and provisioning tasks in a network device. In this section, we develop an architecture for an Intelligent provisioning agent whose service-specific intelligence is derived from configuration modeling and device-specific rules. The agents can translate user request and pseudo rules extracted from data/policy repository into device-specific configurations to dynamically provision network devices.

4.1 Problem of Conventional Provisioning Agent

Automated agents [22] that exist today for provisioning network devices consider translating device-specific rules, but are not truly interactive and intelligent. These agents basically push configuration scripts to the devices and can be characterized as follows:

- These provisioning agents assume that since the scripts are correct in syntax the devices will be configured correctly. This always does not happen. The basic problem of this approach is that it does not consider the current configuration state in the devices. Consequently, potential conflicts might arise while agent pushes configuration script to the certain network devices. Also, this approach does not allow to provide an device independent repository.
- Most of the agents keep device state in an external repository and assuming that to be the actual device state at all times. However, human system administrators might as well manually configure those devices, in which case, the external repository based device state no longer reflects the actual device state.
- Because such agents lack interaction, in case of a conflict no way remains to identify it.

4.2 Agent Architecture

We considered the general problems of the conventional provisioning agents and have designed our agent that does not suffer from the same problems. Figure 5 shows the internal architecture of our proposed intelligent provisioning agent. The main functional components of the agent follow:

4.2.1 Knowledge Base

In earlier section we discussed about modeling a configuration element and mapping such elements in a configuration space. The main motivation was to to create a knowledge base for a specific service. The service, for example, could be dynamic VPN service creation, or QoS configuration, or QoS enabled VPN, in which case both VPN and QoS configuration is necessary in a network device. For a particular service, the knowledge base, contains complete dependencies of configuration elements constituting the service, their composition in a configuration space including the device-specific rules.

4.2.2 Element Linker

This uses the knowledge base to create links between various configuration components to build a new service. Rings of some configuration elements are attached to the hook of another one to create links. Link assembler does this job when requested by cell-specific mini agents in an agent kernel. Link disassembler traces back the links of an existing service to modify one of the configuration elements to add or remove a new service.

4.2.3 Agent Kernel

It is the main functional engine of an intelligent agent to support automated intelligent configuration. The essential parts forming the kernel are:

- *Puller*: It retrieves configuration information (for example Cisco IOS file) from the target device to read the current configuration state in that device. This pulled information can be a simple ASCII file or HTML embedded. Whatever the format is, the kernel, with its parser, extracts appropriate values and configuration elements.
- *Cell-Specific Mini Agents*: These are specialized in managing configuration elements that are cell-specific. A particular mini agent only knows how to configure elements with similar bodies. The agent kernel has several mini agents that work cell-wise and the results from these when combined with the help of element linker (which of course derives knowledge from the knowledge base) produce the desired action needed to create the required service.
- *Pusher*: This sends the ultimate combined configuration results to the network device through device connector.

4.2.4 Interface to Dispatcher

This is a communication port of the agent for the Central Dispatcher to send service request. This is usually achieved by having the server socket program waiting for service request on a specific port. The interface also serves as an indicator of service type the agent can provide, since agents are service-specific.

4.2.5 Interface to Report Collector

This is yet another communication port that is actually a client socket program. This interface basically sends all event reports prepared by the reporter of the agent kernel to report collector located in the central repository system.

4.2.6 Device Connector

As configuration sent by the pusher needs to be sent in different ways to the devices, Device Connector has several plug-ins like TELNET, HTTP, FTP, SNMP etc. to facilitate such service in a heterogeneous environment where devices have various access mechanisms.

4.3 Example of Intelligent Provisioning

In this section we will show some VPN service-specific implementation examples of the intelligent provisioning agents.

The first example in Figure 6(b) shows C++ code of a cell- specific mini agent that intelligently pulls out available traffic-classifier number (usually called `access-list` number) from a network device. Every time there is new a source-destination pair for VPN traffic a new `'access-list'` number is needed that uniquely identifies the traffic. If the valid starting number is 100 for a particular device and there is no traffic classifier present in that device the mini agent returns 100; otherwise, it returns the next unused number. It does so by searching for configuration elements having body `access-list` in the configuration space represented as `ConfigFile` in the code. The string `ConfigFile` contains the whole Cisco

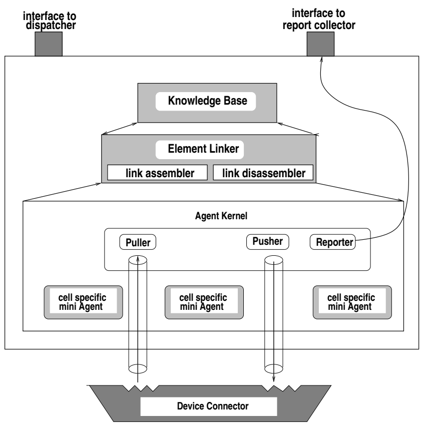


Figure 5: Intelligent Provisioning Agent Architecture

IOS configuration file as shown in Figure 2(c). This helps us to avoid storing device-specific data in policy repository. In this example, we do not need to store `access-list` numbers because this is device-specific and having only source-destination pairs as VPN traffic in the policy repository should be enough for other device-specific agents to translate that into appropriate device-specific configurations by deploying similar agents.

The second example in Figure 6(a) shows C++ code of another cell-specific mini agent that smartly configures an IPSec VPN service in a network device. Assume that we currently have a network setup as shown in Figure 2(a) and a VPN administrator wishes to create a tunnel between device 130.92.70.102 and 129.194.90.20 for a source-destination pair 172.17.0.103 and 172.18.0.100 using security parameters AH authentication and ESP encryption. Figure 2(b) shows the expanded network setup. An automated provisioning agent that ignores configuration dependencies among different configuration elements would probably generate the following configuration for the network device 130.92.70.102.

```
crypto map genbern 161 ipsec-isakmp
set peer 129.194.90.20
set transform-set ah-md5-hmacANDesp-des
match address 141
access-list  141 permit ip host 172.17.0.103 host 172.18.0.100
```

However, the intelligent agent enters the target device, reads the existing configuration, and analyzes it using the knowledge base and detects that a tunnel already exists with the same security parameters between the same peer routers although for different LANs. In doing so, it actually searches configuration elements in a cell having body tag `crypto map` whose attributes are the same as the newly requested one. As one such already exists, it discovers that adding the above configuration lines would be unnecessary. The mini agent, therefore, pulls the existing traffic classifier number (i.e.`access-list` number 140), and simply adds the line `access-list 140 permit ip host 172.17.0.103 host 172.18.0.100` to the current configuration.

5 Intelligent Device Auditing

When configuration faults occur on the network device, it is imperative that problems be resolved quickly to decrease the negative impact on user productivity. Network managers

```
/* setting the values of source, dest address that would be
    tunneled. 'peeridaddr' is the remote crypto endpoint for the
    tunnel and the rest are ipsec tunnel authentication and
    encryption parameters. */
   string source = "172.17.0.103";
   string dest= "172.18.0.100";
   string peeripaddr ="129.194.90.20";
   string AH_authentication="ah-md5-hmac";
   string ESP_encryption="esp-des";

/* setting the regular expression search string */

   string ipsectransformset=AH_authentication;
   ipsectransformset +="AND";
   ipsectransformset +=ESP_encryption;
   string searchstr;
     searchstr  = "crypto map genbern \\d+ ipsec-isakmp.*\n";
     searchstr += ".*set peer "+peeripaddr+".*\n";
     searchstr += ".*set transform-set "+ipsectransformset+".*\n";
     searchstr += ".*match address (\\d+).*";
 /* searching in configuration space for a similar tunnel. */
    ConfigFile contains complete configuration information */
     regx re(searchstr);
     regx::iterator ipsec = re.find(ConfigFile);
     if (ipsec == re.end()) {

     No match found. Code to proceed as usual.
             ⋮
      }
/* if match found re(1) contains traffic access list number.
   Just generate the ACL config line and that's enough for
   tunnel configuration */
    string configline;
    configline = "access-list ";
    configline += re[1];
    configline += " permit ip host ";
    configline += source;
    configline += " host ";
    configline += dest;

 /* send this configuration line down to the router */
   ConfigRouter(configline);
```

(a)

```
/* searching for all ACL Numbers */
  regx re("access-list (\\d+) .*");
  regx::iterator acl = re.find_n(ConfigFile);
  if (acl == re.end()) {
      int StartACL = 100;
      return StartACL;
    }
  /* Vector V would contain all the ACL Numbers */
  vector <int> V;
  int c=0;
  for (regx::iterator i = re.begin(); i != re.end(); i++) {
    if (1&c) V.push_back(atoi(re[c].c_str()));
    c=c+1;
  }
  /* Searching for the Maximum ACL Number. *it is the highest
     number, so return (*it +1) to be used as the next ACL. */
  vector<int>::const_iterator it = max_element(V.begin(), V.end());
  return (*it+1);
```

(b)

Figure 6: (a) Partial Code for Smart Provisioning: Example 1 (b) Avoiding Device-Specific Data in Repository: Example 2

must respond quickly and have procedures in place to reestablish lost service and maintain beneficial service levels. Several reasons justify why such problems might occur. Services configured by human administrators may lead to mis-configuration or cause conflict with the existing services activated via the automated system. Also, if network devices crush due to power failure or some other reasons, valuable configuration information thrown into the devices might well be lost partially or fully. One cannot rule out the possibility of an inside or outside intruder changing some configuration settings. Intelligent audit management handles such problems to recover from any mis-configuration caused by any process other than the automated system. Audit management ensures that the configuration State in the policy repository is equivalent to actual device state as illustrated in Figure 7. Similar to provisioning agents intelligence is built in the audit agents that gather and sort the configuration data to quickly identify the cause and location of faults in a network device and automatically fix the problem when it occurs. Depending on SLAs with customers, audit agents can be periodically sent to the network devices to check the consistency of network device configuration.

5.1 Audit Management Architecture

With the objective that the configuration state in the repository must be equivalent to the actual device state, we have proposed intelligent audit management architecture (Figure 8). Audit management process is done periodically for each device under the control of automated management system and basically goes through the following cycle of actions:

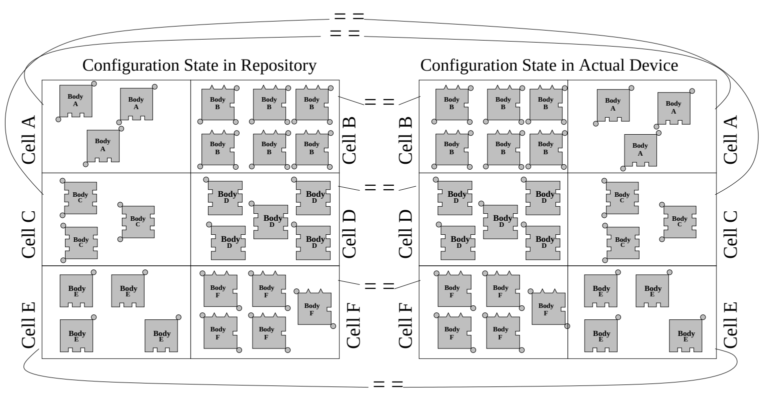


Figure 7: Objective of Audit Management for Device Configuration: Configuration State in the repository must be Equivalent to Actual Device State

5.1.1 Audit Request Generation

The central repository system maintains records of all the service requests provisioned by the automated system. For a specific device, the Audit Manager (i.e. audit agent) can extract all service requests and generate expected configuration state for that particular target device. This is done by the device-specific interpreter (DSI) which utilizes the same knowledge base as used in a provisioning agent. This step, therefore, is quite similar to original service provisioning of a new service request. However, unlike the latter case when an intelligent provisioning agent actually configures a service, the DSI simply prepares expected configuration for provisioned services and passes that to configuration violation detector (CVD) to identify any unwanted modification in the physical device.

5.1.2 Problem Finding

At this stage the audit management agent first pulls the current configuration state from the device and the CVD compares that with the expected configuration state received from DSI. If any configuration violation is detected (i.e. expected configuration and actual device configuration do not match) the violation detector signals that to configuration violation locator (CVL). Since we map the whole configuration space by dividing them into clusters of cells, the violation locator locates cells where one or more configuration elements may have been changed.

5.1.3 Problem Fixing

For a specific service, changes or mis-configuration in one configuration element may easily affect other attached elements that constitute the service. Therefore, to audit a service all appropriate affected cells are located to be fixed by re-provisioning process. The Problem fixing stage basically sets the configuration state in the device to the expected state by making specific corrections in corrupted cells (of the the configuration map) located by the CVL. Violation corrector determines what needs to be changed in the corrupted cells. Usually some configuration elements might need to be added, modified or deleted. For these actions to take place, cell-specific mini agents re-provisions the network device to bring the configuration state of the device back to the desired state. Although not shown explicitly in Figure 8, both in problem finding and fixing stages the knowledge base and other components in the intelligent agent architecture needs to be invoked for the re-provisioning agents to take appropriate actions.

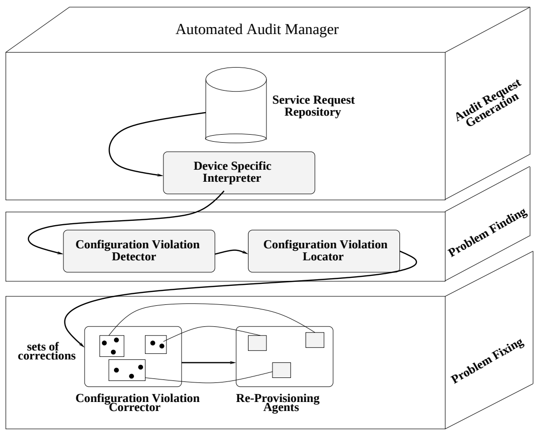

Automated Audit Manager
Service Request Repository
Device Specific Interpreter
Audit Request Generation
Configuration Violation Detector
Configuration Violation Locator
Problem Finding
sets of corrections
Configuration Violation Corrector
Re-Provisioning Agents
Problem Fixing

Figure 8: Automated Audit Management Agent Architecture

6 Conclusion

In recent years, intelligent mobile agents have proven to be indispensable tools for providing network management assistance. They will become even more indispensable as networks continue to expand and companies continue to minimize their personnel requirements. For ISPs and large enterprises, the capability to effectively and remotely provision large number of network devices from a centralized or distributed locations becomes even more important. We have proposed a new distributed architecture where highly mobile intelligent agents can take the responsibilities of not only provisioning but also configuration audit management in a timely fashion. Although recent research have focused on using mobile agents for network monitoring or simple push-based device configuration in a distributed architecture, their ability have not been exploited in dynamic IP service provisioning. In this paper, we have taken a new approach to configuration modeling that is device neutral and based on which any existing or emerging new IP services can be presented in abstract form by encapsulating service semantics, including service-specific data. We have developed new mobile intelligent provisioning and audit agent architectures that use the knowledge built upon configuration dependency modeling. We have not addressed the issue secured delivery of configuration by the agents. This can be easily achieved by establishing a secured management tunnel between the agent and the network device using IPSec or SSH protocols. However, if the devices do not support such protocols, a scalable new mechanism for secured configuration delivery will be an interesting research issue. Performance evaluation in terms of provisioning response time could be useful. Also, further investigation of advanced policy management [23] [26] [16] in large scale environment would be interesting.

References

[1] M. Baldi, S. Gai, and G. P. Picco. Exploiting code mobility in decentralized and flexible network management. *First Int'l Workshop on Mobile Agents Mobile Agents'97,Berlin, Germany*, pages 13 – 26, April 1997.

[2] Paolo Bellavista, Antonio Corradi, and Cesare Stefanelli. An Integrated Management Environment for Network Resources and services. *IEEE Journal on Selected Areas in Communications*, 18(5):686 – 701, May 2000.

[3] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Communications Surveys*, 1(1):2 – 9, September 1998.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weis. An Architecture for Differentiated Services, December 1998. RFC 2475.

[5] Torsten Braun, M. Günter, and Ibrahim Khalil. Management of Quality of Service Enabled VPNs. *IEEE Communications Magazine*, 39(5), May 2001.

[6] M. Cheikhrouhou, P. Conti, J. Labetoulle, and K. Marcus. Intelligent Agents for Network Management: Fault Detection Experiment. *Sixth IFIP/IEEE International Symposium on Integrated Network Management (IM'99), Boston, USA*, pages 63–83, May 1999.

[7] S. Crane, N. Dulay, H. Fossa, J. Kramer, J. Magee, M. Sloman, and K. Twidle. Configuration Management for Distributed Software Systems. *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (ISINM 95),Santa Barbara, USA*, pages 29 – 42, May 1995.

[8] M. El-Darieby and A. Bieszczad. Intelligent Mobile Agents: Towards Network Fault Management Automation. *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, pages 611–622, 1999.

[9] F. L. Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen. MPLS Support of Differentiated Services. Internet Draft `draft-ietf-mpls-diff-ext-09.txt`, April 2001. work in progress.

[10] G Goldszmidt and Y. Yemini. Delegated agents for network management. *IEEE Communications Surveys*, 36(3):66 – 70, March 1998.

[11] Alex L. G. Hayzelden and John Bigham. Agent Technology in Communications Systems: An Overview. *Knowledge Engineering Review*, 1999.

[12] M. Katchabaw, H. Lutfiyya, A. Marshall, and M. Bauer. Policy-driven Fault Management in Distributed Systems. *Proceedings of the The Seventh International Symposium on Software Reliability Engineering (ISSRE)*, November 1996.

[13] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol, November 1998. RFC 2401.

[14] Ibrahim Khalil and Torsten Braun. Edge Provisioning and Fairness in DiffServ-VPNs. *IEEE International Conference on Computer Communication and Network (I3CN)*, Oct 16-18 2000.

[15] Ibrahim Khalil, Torsten Braun, and M. Günter. Implementation of a Service Broker for Management of QoS enabled VPNs. In *IEEE Workshop on IP-oriented Operations & Management (IPOM'2000)*, September 2000.

[16] T. Koch, B. Kramer, and G. Rohde. On a Rule Based Management Architecture. *Proceedings of the Second International Workshop on Services in Distributed and Networked Environment,Canada*, pages 68 – 75, June 1995.

[17] Thomas Koch and Bernd Krmer. Towards a Comprehensive Distributed Systems Management. *Open Distributed Processing, IFIP*, pages 259 – 270, 1995.

[18] Hanan L. Lutfiyya, Andrew D. Marshall, and Michael A. Bauer. Configuration Maintenance for Distributed Applications Management. *Proceedings of the CAS Conference (CASCON)*, pages 43–57, November 1997.

[19] J.P. Martin-Flatin, S. Znaty, and J.P. Hubaux. A Survey of Distributed Enterprise Network and Systems Management Paradigms. *Journal of Network and Systems Management*, 7(1), March 1999.

[20] A. Puliafito and O. Tomarchio. Advanced network management functionalities through the use of mobile software agents. *In Proceedings of the 3rd International Workshop on Intelligent Agents for Telecommunication Applications (IATA'99),Stockholm, Sweden*, August 1999.

[21] Morin C Sahai A. Towards Distributed and Dynamic Network Management. *Proceedings of the IEEE/IFIP Network Operation and Management Symposium (NOMS),New Orleans, USA*, February 1998.

[22] J. Schonwalder, J. Quittek, and C. Kappler. Building Distributed Management Applications with the IETF script MIB. *IEEE Journal on Selected Areas in Communications*, 18(5):702 – 714, May 2000.

[23] M. Sloman and E. Lupu. Policy Specification for Programmable Network. *First Int. Working Conference on Active Networks (IWAN99),Berlin*, June 1999.

[24] Morris Sloman. Policy Driven Management for Distributed Systems. *Journal of Network and Systems Management*, 2(4), 1994.

[25] William Stallings. SNMP and SNMPv2: The Infrastructure for Network Management. *IEEE Communications Magazine*, 36(3), March 1998.

[26] Rene Wies. Policies in Network and Systems Management - Formal Definition and Architecture. *Journal of Network and Systems Management*, 2(1):63–83, March 1994.

[27] Y. Yemini, A. V. Konstantinou, and D. Florissi. NESTOR: An Architecture for Self-Management and Organization. *IEEE Journal on Selected Areas in Communications*, 18(5):758 – 766, May 2000.