AUTO-DISCOVERY AT THE NETWORK AND SERVICE MANAGEMENT LAYER

Alexander Clemm, Anil Bansal
Cisco Systems, Inc.
170 West Tasman Drive, San Jose, CA 95134, USA
{alex, abansal}@cisco.com

Abstract: Auto-discovery capabilities of management systems typically pertain to network elements as a whole, i.e., the ability to automatically detect which network elements are connected to a network and to discover their type and physical and logical configuration. Service- and network-layer information, on the other hand, is in general not discovered but provisioned and provided by the organization operating the network and services. There are however scenarios in which the ability to auto-discover such information provides a lot of value. This paper describes the challenges that we encountered, the approach we took, and the lessons we learned in providing auto-discovery capabilities beyond the network element layer in the context of packet telephony and of metro Ethernet. We believe that these experiences will also be applicable to other contexts.

Key words: Network management, service management, auto-discovery, VoIP.

1. INTRODUCTION

A common function of element-layer management systems, as well as systems used for monitoring purposes, is auto-discovery. Auto-discovery is an overloaded term used in different contexts that can hence mean different things, but in general it refers to the ability of a management system to extract on its own certain information about what it is that needs to be managed, rather than requiring users to populate that information. When it comes to network and service layer management, however, information is rarely deduced from the network. Instead, the network and devices in it are considered service-agnostic – they support services, however information about services comes from the service provider. ("Service" refers here to the

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35674-7_66 instantiation of a service for a given subscriber of that service, i.e., a service instance, not the service offering). The provider of a service uses service provisioning systems to drive the network configurations required to support the services into the network. Hence, any service information is assumed to be known a priori, not in need of being auto-discovered – the master of this information is the service provider, not the network. This service information is also used as the basis for aspects such as service level agreements (SLAs) or billing [8]. To verify that a service is provisioned correctly, it is always possible to check whether the current configuration in the network corresponds to the configuration it is supposed to have in support of the service, i.e., whether the network configuration "as built" corresponds to the network configuration "as planned".

In many cases, this is completely adequate. Nevertheless, there are situations in which it would be desirable to be able to discover network and service configurations from the network directly, rather than depending on service-related information from other sources. Reasons for this include:

- A service management and service provisioning system gets deployed at a later stage, after initial network deployment. A service provider would like to be able to see and automatically retrieve what services had earlier already been configured on the network.
- A service provider has maintained poor service records and reason to believe that its service records are not up to date. This scenario can occur specifically where service related configurations are not directly associated with specific end subscribers. An example would be a wireless service provider who needs to provide certain service capacity in a certain area, for instance a certain number of channels for GSM, TDMA, and data traffic for base station traffic.
- Operations personnel within a service provider's organization have gone around service provisioning systems and provisioned service instances "by hand", resulting in certain network-layer configuration mismatches that are hard to troubleshoot.

We have found such scenarios to be applicable for instance in the context of packet telephony management or management of metro Ethernet services, for which we hence encountered the requirement to include the ability to auto-discover network and service layer information as part of the management solution. In this paper, we will discuss the challenges that must be addressed when attempting to provide network and service layer auto-discovery, and an approach and design pattern that deals with those challenges. Our experiences are based on systems that we built for the management of packet telephony (Cisco Packet Telephony Center – PTC) and of metro Ethernets (Cisco IP Solution Center – ISC), and which incorporate the concepts described. However, no inferences should be made about Cisco product features or product direction. We expect our experiences to be not unique to those particular services but transferable to other service domains.

The remainder of this paper is structured as follows. What we mean by service layer auto-discovery and some background are discussed in Section 2. Section 3 dives into the challenges and considerations for service-layer auto-discovery. A design pattern that we have devised to tackle those challenges is subsequently introduced in Section 4. Two applications of this design pattern that have been realized in actual systems, one concerning packet telephony, the other metro Ethernet, are presented in Section 5. Finally, Section 6 offers some conclusions.

2. SERVICE LAYER AUTO-DISCOVERY

Auto-discovery is often encountered in the network element management context. It concerns the ability of a management system to automatically discover information about the network, specifically to discover what devices are in the network, what type of device they are, what their physical configuration is and how they have been logically configured. Frequently auto-discovery occurs by a management application pinging a range of IP addresses specified by the user. Upon receiving a response, the device's entity MIB is queried to identify device type and physical configuration. Subsequently, the device can be displayed on a topology map and is available for monitoring, MIB browsing and other management purposes. This way, auto-discovery obviates the need for management systems to be populated with this information by the user or obtain it through seed files or other systems. It allows the management system to obtain an accurate picture of the devices that are actually deployed in the network.

The situation is different as far as services are concerned. Management information about services is not discovered. Instead, the master of service management information is the service provider that provisions them respectively its operations support system, not the network. A service provisioning system is used to drive the required device configurations to support the services into the network, with the service provider keeping track of what services are provisioned. The provisioned services can be verified by checking whether the actual network configuration corresponds to how it was supposed to be provisioned, comparing whether the network configuration "as built" corresponds to the network configuration "as planned". (Please note that throughout this paper, we refer with service auto-discovery to the automatic discovery of management information that represents instances of services. This is not to be confused with auto-discovery of services themselves, e.g., through advertising of services by application servers in a network, e.g., [6, 9].)

The following are some examples of network and service layer concepts that would generally be provisioned but not auto-discovered by service providers:

- An H.323 zone in a VoIP (Voice over IP) network
- An instance of a residential ETTH (Ethernet To The Home) data service
- An instance of a transparent LAN service (TLS) in a metro Ethernet setting

Not having the capability to automatically discover network and service layer management information is completely adequate in many cases. However, as mentioned in the introduction, in practice scenarios can be encountered where service instances are not necessarily completely known and a capability to automatically discover them is desirable. For example, in packet telephony management, the scenario can occur where a network and service management system encounters an existing deployment when it is introduced. To be useful, this system needs to be populated with service-layer managed objects ("service MOs"), such as information on H.323 zones. Entering this information is a tedious and redundant exercise; after all, the network had already been provisioned earlier with those services. Likewise, even with a network and service management system in place, provisioning can sometimes occur working around it and configuring network elements directly (e.g., through the devices' command line interface - CLI). This leads to network-layer concepts and instances of services being introduced through the back door, without the management system knowing about them and without proper service MOs being created. This is a problem where network layer integrity might be violated without the management system knowing about it. Also, without service MOs being created, there is no way for an Operations Support System (OSS) to subsequently refer to the network and service layer concepts that have been introduced. In some cases, records of what services have actually been provisioned in the network do not exist due to poor operational practices at a service provider, resulting in a network that is overall poorly planned and resources in the network being tied up without generating revenue. In each case, it would be desirable to have a capability to auto-discover services (and network-layer concepts such as connections, which we include in our discussion without each time explicitly referring to them separately).

So what do we mean by service auto-discovery? We refer to it as the ability of a management system to automatically detect what service instances of a given type of service are present in a network, without requiring the user or an OSS to "tell" the system about those instances. In general, this involves the ability to derive service information from network element level management information ("NE MOs"), such as information on the logical configuration of network devices. To provision a service, configurations need to be applied and driven down into the network, often several devices. Hence the service instances present in the network can be "reverse engineered", respectively derived from the network configuration. Service auto-discovery assumes that information about the network elements is already known to the management system and in place, as it is a prerequisite. That information could be known as the result of (network element) auto-discovery and device configuration discovery that has taken place earlier. Figure 1 depicts the relationship between service auto-discovery and service provisioning.

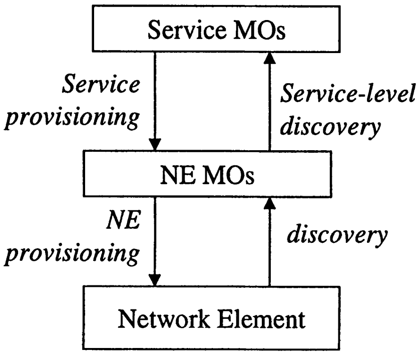

Figure 1: Relationship between provisioning and discovery

The need to auto-discover management information applies also to the network management layer. An example are connections in an ATM network. For the purposes of this paper, we treat network and service level auto-discovery jointly, and our concepts apply to both. MOs at the network management layer and at the service management layer are referred to collectively as "service MOs".

Management information constitutes the starting point for our considerations on service discovery. Service (and network) management layer information builds and depends on information at the network element layer, aggregating and abstracting information from it. Hence the mapping of service and network layer concepts onto information concerning individual network elements is reflected by the relationships between NE MOs and service MOs that express these dependencies (figure 2).

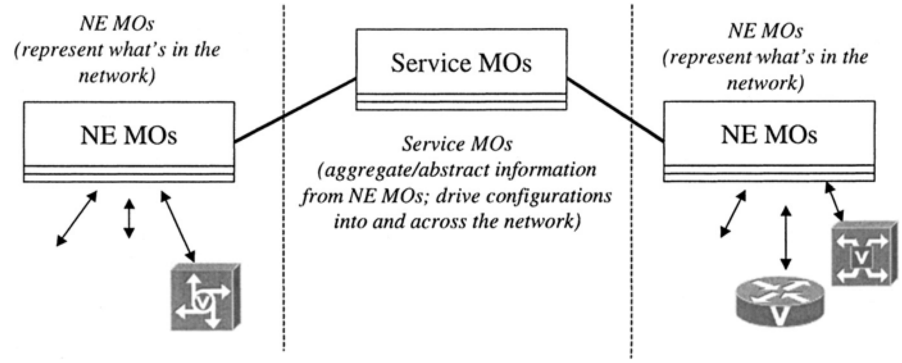
NE MOs
(represent what's in the network)
NE MOs
Service MOs
Service MOs
(aggregate/abstract information from NE MOs; drive configurations into and across the network)
NE MOs
(represent what's in the network)
NE MOs

Figure 2: Dependencies between NE and service MOs

The knowledge of these relationships is essential for service auto-discovery. One focus for service auto-discovery is hence to identify and be on the lookout for NE MOs of certain classes that are capable of maintaining relationships with service MOs, respectively that service MOs usually are dependent upon. The presence of such an NE MO is then an indication that a service MO may also be present. For example, an NE MO representing the endpoint of a connection is an indication for a corresponding MO representing the connection itself. Another trickier example would be an NE MO representing a DS0 port which may or may not be assigned to a service (and hence related to a service MO). Because service MOs usually aggregate and abstract information from several NE MOs, possibly spanning across multiple NEs, part of service auto-discovery concerns also identifying the various NE MOs that go together and collectively support a service MO. In the connection example, this would involve another NE MO that represents the other connection endpoint. In the port example, it might involve a set of other MOs representing connection endpoints, cross connects (relating the port to the connection represented by the connection endpoints), and service features on a feature service that refer to that particular port that jointly together with the port make up a residential subscriber service. Accordingly, the following are key aspects for service auto-discovery:

- Identification of NE MOs of classes that service MOs usually depend on
- Identifying which NE MOs would match up to relate to the same service instance

- Creation of service objects based on aggregate NE MO information

In addition to service information that can be derived from the network, there can be certain aspects about service objects that cannot be derived, as they concern service layer aspects that are not represented in the network elements but are maintained by the service provider. An example for this would be the customer information about who subscribes to a given service instance. This type of business information is not part of the network configuration and its automatic discovery from the network would require clairvoyant capabilities. Clearly, there may be information in a business management system that might be associated with the network information. However, this is not the emphasis of this paper and would be subject to further work; we assume that this type of information will still need to be associated by the service provider.

3. SERVICE LAYER AUTO-DISCOVERY CHALLENGES

As indicated, key to our approach to service-layer auto-discovery is the identification of NE MOs that service MOs could be related to and that hence indicate their possible presence, and derivation of service-layer information from those NE MOs. This sounds reasonably straightforward, although as so often the devil is in the details. The following are some of the aspects that need to be considered.

Multiple ways to instantiate a service. There can be different ways in which the same instance of a service can be instantiated. This means that in general, we need to know about different possible mappings, so we know what things to look for. Of interest are not so much the differences and variation in actual provisioning steps taken, but different ways in which the same service can be reflected in the resulting network configurations.

Identification of "matching" NE MOs. Generally, a service MO does not map one-to-one on an NE MO but is distributed over several MOs. This raises the question how we can find out which NE MOs "match", respectively would be related to the same service MO. How can we tell whether they potentially belong to the same service MO, or whether they would belong to different ones? In many cases, NE MOs related to a service MO contain information about other NE MOs to which they are related. An example are NE MOs that represent connection endpoints and contain information about the IP and port addresses that help identify the corresponding endpoint on the other side. (That MO in turn contains the first MO's NE's IP address information.) In some cases, only some of the NE MOs related to the same service object may have information about their relationship to the other NE MOs. An example is a DS0 port MO, cross-connect MO, and connection endpoint MO on the same network element that are all related to the same subscriber service. The cross connect MO may contain information relating it to the DS0 port and to the connection endpoint, but the connection endpoint MO may not be aware of it relating to either cross connect or DS0 port (see figure 3).

For such cases, service discovery generally needs to focus on the NE MOs that would have information allowing to identify other NE MOs related to the same service MO first. Only subsequently the discovery will expand to try to find the other

NE MOs that contain no such information but that were in part identified by the NE MOs found earlier.

Dealing with rainy day scenarios. Service instances may have been misconfigured. For example, referential integrity problems may exist. An example concerns signaling backhaul in packet telephony, where a media gateway controller might have been provisioned to backhaul signaling to a certain media gateway, but the media gateway expecting signaling backhaul to occur between it and a different media gateway controller. The way management information is represented must account for this possibility. It needs to be able to represent network and service layer information "as built", which includes a lot more possibilities compared to a representation of management information "as planned" that does not need to account for all the things that can possibly go wrong. Information models that represent services typically model services only "as planned", as indicated for instance in the cardinality of relationships. For instance, modeling of a point-to-point connection always involves two endpoints. How would the same information model represent a connection that was "broken" because the endpoints don't match up, violating referential integrity? The model needs to be capable to represent both, the network and services "as planned" with all their constraints, and the network and services "as built", with possible violations of planned constraints. Alternatively, two parallel models need to be maintained. In any event, we need to be able to deal with the world the way it is, not the way we wish it would be. A related question concerns how rainy and sunny day scenarios can even be distinguished. For example, if an NE MO is missing, it might be because truly a misconfiguration has taken place, or because the other NE MO has simply not yet been identified or discovered.

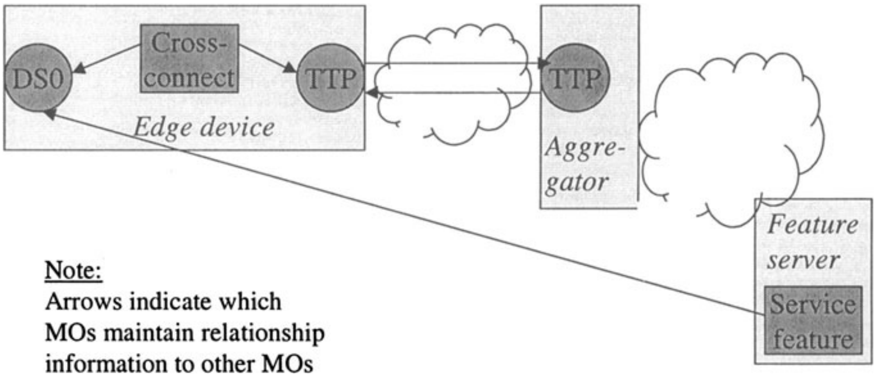

Figure 3: NE MOs involved in providing a residential subscriber service

Incremental discovery. In general, it is unacceptable having to wait for a long period of time, until all network information is precisely known (which might never fully be the case due to the constant changes occurring in real-life networks), to start discovering service MO. Service auto-discovery needs to be able to cope with incomplete information about the network, filling in missing pieces as it goes along and able to indicate to users the status of the discovery process. This is also related to the previous point.

Obtaining accurate NE information: Information has to reflect the current information in the network, requiring periodic synchronization of the NE information that service-layer auto-discovery is based on. This is associated with the usual challenges of keeping a management cache from going stale, nothing unique to service layer auto-discovery but nevertheless worth to be mentioned. Periodic upload and synchronization can be complex if the network or the EMS does not have the ability to provide only the changed information. If the complete network information is uploaded every time, the onus lies on the NMS to find out the deltas that changed since the last upload and update its information accordingly. Event based mechanism can be challenging if either the configuration change events do not carry enough configuration information, if there are too many of them, or if they are not delivered in a reliable manner.

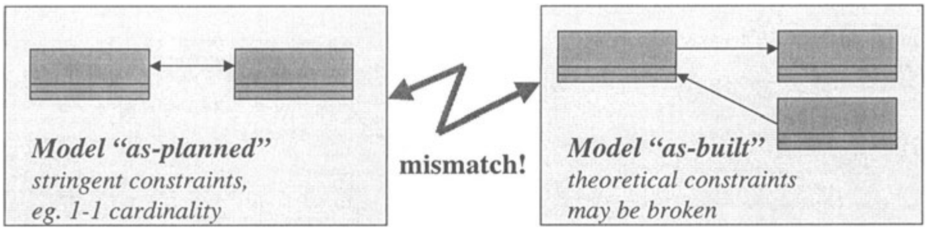
Model "as-planned"
stringent constraints, eg. 1-1 cardinality
mismatch!
Model "as-built"
theoretical constraints may be broken

Figure 4: Rainy day scenarios – discrepancies between "as built" and "as planned"

4. AN APPROACH FOR NETWORK AND SERVICE LAYER AUTO-DISCOVERY

The following outlines our approach to discover service MOs from the network. It has been applied in the context of packet telephony networks as well as Metro Ethernet networks and expect that it can be applied also to other domains.

4.1 Initial analysis

We start with an analysis of the model of the management information that is involved. We assume that a model representing management information from the NEs, i.e., containing the NE MOs, will already be in place. (As mentioned earlier, as a prerequisite for service auto-discovery we assume that NE MOs have been discovered already, so they need to populate some model.) Certain categories of NE MOs will typically be part of (or referenced by) a service MO. We will refer to those NE MOs as "service-supporting NE MOs". For example, a termination point will be indicative of a connection.

If we do not have one already, we define a model of the network and service layer concepts that need to be discovered. This model can be arbitrarily defined; it can even include new services that were not known at the time when the NE MOs were originally defined. As far as the service MOs aggregate and abstract information originating from the network, they have dependencies on NE MOs. We need to explicitly identify these dependencies respectively relationships between service MOs and service-supporting NE MOs. Service-supporting NE MOs generally maintain relationships with other service-supporting NE MOs, jointly serving to support the higher-layer abstraction represented by the service MO. In many cases, the NE MOs will contain information that points to the other NE MOs that they are in relationship with. For example, an NE MO representing a termination point may include the IP address and port number of the remote end.

Knowledge of these relationships and dependencies is the basis on which the rules can be defined according to which discovery takes place. Some of these rules will be "triggering rules", which provide the conditions under which a service object will be created. The trigger generally identifies certain service-supporting NE MOs, which we will call "master NE MOs", that are a certain indication that a service MO is present. It is important that master NE MOs are defined such that there is only one master NE MO per service MO, so not to inadvertently introduce too many service MOs. Redundant service MOs would be difficult to match and eliminate later. In many cases, there are different possible candidates that could serve as a master NE MO. Often, information models are "symmetrical" in that for instance all the service-supporting NE MOs are of the same type. In this case, a master NE MO can be identified not by the NE MO type but by another distinction, for instance through the system it is contained in. In the packet telephony case, the call controller is generally considered key to the configuration, accordingly the service-supporting NE MOs of the media gateway controller are considered the "master". In peer-to-peer cases, a convention which of the service-supporting NE MOs should serve as the master NE MO could be the NE MO whose NE's IP address is lower than that of the other NEs to which it points.

Other rules will be "completion rules", which identify other dependencies of the service object, i.e., the other service-supporting NE MOs that must be in place for a service object to be "complete", or consistent. When a service MO is first created, it is only related to the master NE MO that triggered its creation. Since other NE MOs typically support this service MO, the information contained in it is incomplete. Hence we also introduce a state concept to indicate the status of discovery, termed the "discovery state". When initially created, a service MO will typically have a discovery state of "incomplete". It moves to a completed state once the completion rules have been satisfied, respectively all NE MOs that the service MO depends on identified. The discovery state also helps deal with cases where services were misconfigured, for example where referential integrity is violated and other NE MOs that are needed for the service to be consistent and complete cannot be found. An incomplete discovery state can be an indication for such situations.

Finally, aggregation rules will define the computation of any derived attributes. All those rules could conceivably be specified separately and processed by a discovery inference engine. However, in our case we chose to simply encode those rules in the respective discovery algorithms of the systems that we implemented.

4.2 Steps during runtime

The steps that occur during service layer auto-discovery are accordingly as follows:

The first step is really outside the scope of service layer auto-discovery itself and is a prerequisite for the auto-discovery of service MOs that follows. It involves discovery of the NE MOs. This means management information is uploaded from the underlying element management system or network element, as part of initial upload or (later) of synchronization. (The network elements will generally be auto-discovered themselves; however, it is not a prerequisite as this information could also be populated using some other mechanism.) Internally, MOs representing those NE resources are created. This includes physical aspects (cards, ports) as well as logical aspects (protocol entities, termination points, etc.).

Next, NE MOs are scanned to identify service-supporting NE MOs, for instance NE MOs of certain classes. Examples are termination points (indicative of a connection), trunk group controls in a call agent (indicative of a trunk group), a cross-connect in an edge device that relates a DS0 with a trail termination point connecting to an aggregation device (indicative of a residential subscriber service, as per the earlier depicted example). A matching NE MO will in all likelihood exist on another NE, e.g., a matching termination point in the case of the connection or a DS1 in case of the trunk group control. If a matching NE MO is found, a service MO with those NE MOs should be created.

Master NE MOs are identified. For each master NE MO, a service MO is created as a result. This can occur before a matching counter piece is found. In the MGCP-based packet telephony case, the media gateway controller is generally considered key to the configuration, accordingly the service-supporting NE MOs of the media gateway controller are considered the "master". The service MO will be marked as "incomplete", as not all NE MOs that the network/service layer concept is composed of are identified. However, the service MO will have enough information to locate the "missing" NE MO. For instance, a trunk group control will indicate the port number, slot number and shelf name of the DS1 it is supposed to be controlling, or a termination point has the address of its counterpart on the other side.

Subsequent steps attempt to identify the other NE MOs that support the service MOs that have been created. This can be done as NE MOs are discovered, or in an extra pass scanning all the NE MOs. Matching NE MOs can be identified through the specific semantics of the underlying model. A simple example concerns network connections: a network connection service object could be initially created with the information contained in an MO representing a connection endpoint. This MO contains the far end's IP address and port number. NE MOs from the far end's systems can now be searched for another connection endpoint object, that has as far end the initial NE MO's NE's IP address (and corresponding port number).

As service MOs are "completed", they will be marked with a discovery status of “complete”. Also, information aggregated and abstracted from the supporting NE MOs can be computed. If conflicting information is found between the service supporting NE MOs, the service MO can be marked as “inconsistent”.

Finally, the service provider has the possibility to associate the identified service instances with other service-related information from the OSS, such as customer information.

The steps can be interleaved. For example, it is possible for NE MO auto-discovery (or discovery) to take place while auto-discovering service objects, performing analysis of service-supporting NE MOs as things go along. Also, multiple passes may be applied. In the first pass, information is extracted from the network and raw service MOs, based on master NE MOs are created. In the second pass, service MOs are refined and completed.

Finally, it is possible to discover service MOs on an ongoing basis, even after the initial auto-discovery pass. Changes to NE MOs will trigger auto-discovery rules to be re-evaluated, based on whether the NE MO was associated with a service MO to ensure that service integrity is still met), whether the change should trigger creation of a new service MO, or whether the change implies that the NE MO can be newly associated with an existing service MO

To identify network-layer inconsistencies and misconfigured services, a user should check for service MOs with a discovery state of incomplete or inconsistent. Also, service-supporting NE MOs that are not related to any service MO are indicative of "orphaned" resources in the network that lie idle and should for management purposes be garbage collected.

5. APPLICATION EXAMPLES

As mentioned earlier, we have realized the presented service and network layer auto-discovery concepts in two systems addressing two very different domains. One (PTC) concerns the management of packet telephony networks, the other (ISC) management of metro Ethernet, specifically Transparent LAN Service (TLS). This is an indication to us that the concepts are indeed generic and will be applicable to other areas as well.

Packet telephony. The fundamental ideas underlying the system for packet telephony management have been described in [2]. Central to it is the notion to hide the distributed nature of a packet telephony network by projecting virtual entities onto it that provide a logical management wrapper around the physical network. This greatly simplifies its management, as management complexity that results from the distribution is largely abstracted away. Examples for virtual entities are a virtual switch that represents a media gateway controller (MGC) and the media gateways (MG) that it controls along with the various signaling and control connections between them in an MGCP network, or a virtual zone representing a gatekeeper and a set of associated gateways sharing the same dial prefix in an H.323 network. The virtual entities constitute a mix of network and service management layer objects to which auto-discovery can be applied. Provisioning of packet telephony networks can be fairly error prone, therefore the ability to detect network-layer configuration mismatches using auto-discovery of the virtual entities proved to be a very attractive side aspect of the system.

An analysis of the packet telephony information model [3] yields the service-supporting NE MOs, basically the NE MOs that the various virtual entity MOs are related to. Most of the virtual entities are based on "symmetrical" relationships between a virtual entity (service layer) MO and two NE MOs of the same type. We declare the NE MOs contained by the media gateway controller the master NE MOs. Key to discovering the various virtual entities is discovering the top-level virtual entities, i.e., the virtual entities that contain the other virtual entities, for example the virtual switch in an MGCP-based network. The virtual switch can be easily identified by identifying the MGCP connections, derived from the MGCP termination point maintained by a media gateway controller and the MGCP termination point of the media gateway that it points to. In a first pass, the virtual switches are identified this way. In a second pass, the other virtual entities are identified, i.e., the various aspects contained in the virtual switch, such as trunk groups, trunks, or backhaul connections. Matching up of the related NE MOs is fairly straightforward once the virtual switch itself is known, as it is clear where to look for the counterparts of the MGC's NE MOs.

The algorithm as described is of course somewhat simplistic. In reality, some aspects turned out to be quite tricky. For example, there can be multiple MGCP associations between an MG and several MGCs in a failover configuration in order to provide fault tolerance. At the same time, an MGC can have MGCP associations with multiple MGs, for which however the same failover configurations will need to apply. This leads to very sophisticated construction principles for the virtual entities and to fairly complex configuration constraints whose integrity must not be violated, respectively many rainy-day scenarios. Another challenge was the interdependence between service layer objects. Service objects representing PRI signaling backhaul, trunk groups, and DSx lines are dependent on creation of MGCP association and virtual switch objects. Unless MGCP association as well as virtual switch objects are discovered first, PRI backhaul and other service objects cannot be discovered because they use the association knowledge between MG and MGC. Thus the complete discovery process involves multiple phases where the initial phase discovers those MOs that have no dependency on other MOs, and subsequent phases discover those MOs that are dependent on already discovered MOs.

Metro Ethernet. Metro Ethernet management [1] deals with Transparent LAN Service. TLS could constitute a multipoint to multipoint connection or a single point to point connection and is used to transparently connect LANs at multiple customer sites as one single LAN. TLS is abstracted as a service object and is formed by grouping multiple network layer objects. Refer to Figure 5 for a TLS example. In the example, TLS connects three customer sites LANS together to form one virtual LAN. The end user is interested in only the endpoints of the TLS and does not care which intermediate nodes the circuit goes through. However, the TLS is comprised of multiple segments (6, in this case) at the network layer. As a part of auto-discovery, the information is read from multiple devices, individual segments are formed, and then these segments are grouped together to form a TLS service.

When the management system is connected to a live network where some TLS circuits are already provisioned, the management system auto-discovers all the TLS services and makes them visible to the user. This is also useful for further resource allocation in the network for the services provisioned later. The management system can keep track of resources such as ports, VLAN IDs which have already been used by the auto-discovered services and does not let operator use those resources.

Similar to the case of packet telephony management, auto-discovery can also serve to detect network layer inconsistencies. For example, a segment belonging to TLS1 can be misconfigured to belong to TLS2. When NMS uploads the network objects from individual devices and correlates these objects, the algorithm can detect the inconsistency and flag it to the user through its GUI.

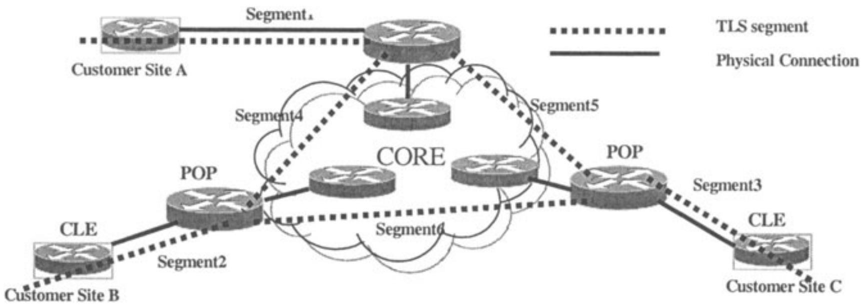

Figure 5: TLS in a Metro Ethernet

6. CONCLUSION

In general, auto-discovery focuses on the network element management layer. However, there are legitimate and important reasons to extend auto-discovery to higher management layers as well. The concepts discussed in this paper have been utilized by management systems for Open Packet Telephony and for Metro Ethernet, with convincing results. The most important aspect perhaps is a side effect of the auto-discovery itself, namely the ability to detect network-layer inconsistencies in the network where service objects are not able to reach a discovery state that indicates they are consistent and complete. Some of our practical experiences have been that service layer auto-discovery is however an expensive operation that should be used sparingly. It is applied once during initial cold start of the system and subsequently on operator request; essentially in situations where the service provider has reason to believe that the service information is no longer accurate. An important feature of our systems is the ability to restrict service-layer auto-discovery to a certain scope, such as in packet telephony an H.323 zone. The scope should of course be defined such that the service-layer concepts within it are self-contained, so that unnecessary and erroneous flagging of seeming inconsistencies – where service-layer aspects extend to information outside the scope is avoided.

Future work could aim at deriving auto-discovery rules automatically from service definitions used to provision the network, such as the ones defined in and applicable to systems and methodologies such as the ones described in [4, 5, 7, 10]. Another area for further research concerns matching auto-discovered data with information contained in the service provider OSS. This would allow for instance to automatically associate subscriber information with service instances identified in the network, so really complete service auto-discovery with service aspects that are not to be derived from the network side.

ACKNOWLEDGMENTS

The authors would like to thank Prakash Bettadapur, Poon Leung, and Petre Dini for fruitful discussions on this subject.

REFERENCES

[1] Barry, D.: Metro Ethernet Management. Packet Magazine 3/2002, softcopy at http://www.cisco.com/warp/public/784/packet/jul02/p45-cover.html, 7/2002.

[2] Clemm, A., P. Bettadapur: Building Management Solutions for Open Packet Telephony Networks. IEEE/IFIP IM 2001, Seattle, WA, 5/2001.

[3] Clemm, A., P. Leung: Model-Driven Open Packet Telephony Management. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.

[4] Dreo Rodosek, G., L. Lewis: Dynamic Service Provisioning: A User Centric Approach. IEEE/IFIP DSOM 2001, Nancy, France, 10/2001.

[5] Garschhammer, M., R. Hauck, B. Kempter, I. Radisic, H. Rölle, H. Schmidt: The MNM Service Model – Refined Views on Generic Service Management. Journal of Communications and Networks (JCN) Vol. 3 Nr. 4, 12/2001.

[6] Jacob, B.: Service Discovery: Access to Local Resources in a Nomadic Environment. OOPSLA'96 Workshop on Object Replication and Mobile Computing, San Jose, CA, 10/1996.

[7] Kong, Q., I. Rose, D. Cameron: Towards Technology Independent and Automated Service Activation and Provisioning. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.

[8] Lewis, L: Managing Business and Service Networks. Kluwer Academic / Plenum Publishers, New York, NY, 2001.

[9] Preuss, S.: JESA Service Discovery Protocol (SDP). Proceedings Networkers 2002 (Springer), Pisa, Italy, 5/2002.

[10] Shen, F., Clemm, A.: Profile-Based Subscriber Service Provisioning. IEEE/IFIP NOMS 2002, Florence, Italy, 4/2002.