Application-Specific Policies: Beyond the Domain Boundaries

Herbert Bos
University of Cambridge, Computer Laboratory,
CB2 3DQ, United Kingdom
hjb1005@cl.cam.ac.uk

Abstract

We propose a network management and control architecture that is able to dynamically load application-specific code, which in turn is able to control resources at a very low level. Recognising the existing diversity of network management and control architectures, we then address the problem of interoperability between control architecture domains. Given such interoperability, we show how clients can install application-specific policies that span multiple domains.

keywords

Interoperability and cooperative control, agents, open control and management

1 Introduction

In this paper we concern ourselves with establishing resource management policies that span multiple domains. We call such policies *global*. Resource management in networks involves the reservation and allocation of resources to applications, for example in the form of connections. This is true regardless of the network technology. Resource Management and Quality of Service (QoS) guarantees have been a part of ATM from the outset but are now rapidly gaining ground in IP as well. The focus of this paper is on ATM but many of the issues are equally relevant in other network technologies. We observe that there may be many policies pertaining to resource management, each of which may be suitable in certain application areas, but not in others.

We define a network *management and control architecture* (MCA) as the set of protocols, policies and algorithms used to control and manage a network. *Sandman*, the MCA described in this paper implements commonly used operations efficiently and with simple interfaces, while allowing applications to extend this basic functionality to customise according to their needs. We call such a MCA *elastic*.

1.1 Contribution

The first problem we address is that the generic nature of high-level primitives prevents applications from exploiting application-specific knowledge. As an example, consider

Figure 1. The nature of an application $\mathcal{A}$ may be such that at any time only one of the endpoints is active as source and every endpoint C_{i+1} becomes the source exactly $T_i(t)$ seconds after its neighbour C_i became active as source (i.e. the source moves in clockwise direction, assuming addition is mod 4). For application $\mathcal{A}$, the connections to be made are determined by the following algorithm: if the last connection from C_i was a multicast, then a connection is made across switch S from C_i to C_{i+2}, else a multicast connection is made from C_i to C_{i+1} and C_{i+3}. For good performance, it would be useful for $\mathcal{A}$ to set up all connections to and from the central switch S so a change of source only requires changing the switch connection in S according to the algorithm. This is hard to do using high-level end-to-end primitives. While this example lies almost entirely within the *control* plane, a similar example for the *management* plane may consist of gathering and manipulating management information whenever some application-specific event occurs.

As a solution, we introduce a MCA which allows applications to load their own code into the network, i.e. to program the network. This, combined with the ability to reserve and allocate arbitrary collections of resources in the network, opens up the control and management architecture to incorporate application-specific behaviour. Although beyond the scope of this paper, we observe that advance reservations are also explicitly supported. The code is able to interact with the MCA at a very low level, enabling applications to have their own policies executed both in the reservation and allocation domain. Applications can even extend the MCA with new operations which are accessible to other applications as well.

This brings us to the second problem. The solution of loading application-specific code as described above allows applications to introduce application-specific policies, into the heart of the MCA controlling a certain MCA domain. Many applications, however, extend beyond the boundaries of a single control and management domain. We would like to support these applications in a similar way. The challenge here is twofold.

Firstly, different types of MCAs should be enabled to interoperate. Ideally, this should be possible without degrading the functionality of two communicating feature-rich MCAs A and B, only because an interconnecting MCA C (located between A and B) does not provide this rich functionality.

Secondly, clients should be allowed to take their policies across domain boundaries. This way clients can exploit their specific knowledge about the nature of their application throughout the network. We call these policies *global*. One problem here is that, although applications may be assumed to have knowledge about the local domain (e.g. about the topology), no such knowledge can be assumed for remote MCA domains.

1.2 Overview

We introduce an elastic MCA called *Sandman* that allows applications to specify their own policies which may span multiple MCA domains. Background information is given in section 2. Section 3 discusses basic operations of the Sandman. Light-weight virtual networks or *netlets* are introduced in section 4 and dynamic code loading in section 5. Techniques to achieve interoperability are the topic of section 6, while global policies are discussed in section 7. Implementation details are mentioned in section 8. Related work and conclusions, finally, are the topics of section 9 and 10 respectively.

2 Background

It seems unlikely that there will ever be a one-size-fits-all solution for MCAs. Many solutions for network control exist today (e.g. Q.2931, P-NNI, IP Switching, etc.) each serving its purpose, but it is not realistic to expect any of these to evolve into *The* MCA that will cater to all our needs, present and future.

Instead, we would like to enable users to control their networks with the MCA that suits their environment best. For this purpose, previous work in the Computer Laboratory has allowed us to partition physical networks into virtual networks, each of which can be controlled by its own MCA [1]. This is illustrated in Figure 2, where a switch divider process partitions the resources on a switch into *switchlets* and offers the same switch interface to the MCAs as found on the switch itself. It appears to the MCAs as if they are controlling a real (albeit smaller) switch. Clients C1, C2 and C3 each then request its own MCA to exercise management and control, e.g. to set up connections.

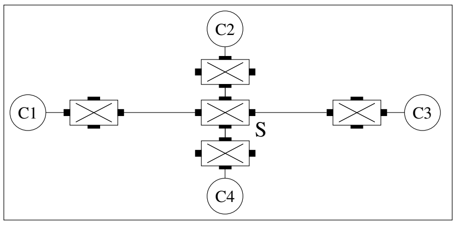


Figure 1 Client-specific knowledge

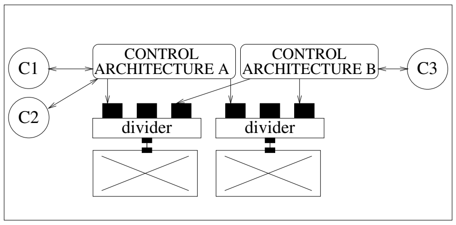


Figure 2 Partitioning switches

As mentioned before, we think that the infeasibility of a one-size-fits-all solution applies also to the MCA itself. Therefore, we would like to extend the idea of switchlets into the MCA by enabling individual applications to specify their own policies for reserving and allocating resources. In this way, we can really speak about *open control*: flexible control that is not dictated by any one standard, organisation or network operator. Even so, the MCA proposed here is not intended to replace any existing MCAs. Instead it is expected to run alongside them as illustrated in Figure 2.

3 Basic operations

The MCA proposed here supports a few basic operation classes. For this paper, the relevant ones are:

1. *Unicast connection*. Probably the most common operation is the connection from source to sink for a particular time interval with particular characteristics. If the admission control accepts accepts a request for such a connection, the client is guaranteed that the connection will be set up in that time interval. Traditional, immediate, connections simply leave out the interval in which case it defaults to $[now, \infty\rangle$.

2. *Multi-source, multi-sink connections*. A small number of more complicated types of connections exist, such as a connection that is time-shared by multiple sources and which may have multiple sinks each with its own and possibly overlapping time interval. These are described in [2].

3. *Information gathering.* A rather wide-ranging class of operations to discover the state of the network, the topology, routes, available capacity, etc.

4. *Reservation of arbitrary sets of resources.* The reservation of arbitrary sets resources is described in section 4.

5. *Loading application-specific code.* Allowing applications to load their own code into the MCA allows them to exploit application-specific knowledge on a very low level. We will discuss loadable code in section 5.

The first two of these operation classes allow for reservation in advance, so that guarantees about the availability of resources at some time in the future can be given. These operations are very common and can be expected to be sufficient for the majority of applications. We call these the *primary* operations. All other operations mentioned above are called *secondary* operations.

4 Recursively partitioning networks

The *primary* operations are expected to be sufficiently expressive for a large class of applications. Some applications, however, have very specific needs so, in order not to restrict them, we propose to give these applications a number of resources which are theirs to use as they please (i.e. without imposing on them connections of any predetermined type). This is also useful for certain network management tasks. For example, it has been suggested in [3] to partition resources in the (virtual) network, so that immediate reservations are shielded from advance reservations (and vice versa). For this purpose the Sandman allows a client to make (possibly advance) reservations for something called a *netlet*, which is a small virtual network in a larger virtual network.

Netlets consist of (a share of) an arbitrary set of resources within the encompassing virtual network (VN). For example, for a switch port we specify a netlet element consisting of the switch name, port number, direction (i.e. in or out), number of channels (e.g. VCIs in ATM) and bandwidth. These elements need not be adjacent as one netlet may consist of multiple unconnected sub-partitions (see Figure 3).

Netlets can be created recursively, so it is possible to create netlets in netlets. This enables applications to repartition resources almost unrestrictedly. In fact, the encompassing VN of section 2 can itself be thought of as a netlet (the *null-level* netlet). Repartitioning network resources merely extends the idea of switchlets into the MCA. This has a number of advantages. We briefly mention two:

1. *Policing differentiating.* VNs must be policed, because misbehaviour in one VN (null-level netlet) $\mathcal{N}_0$ should not affect any of the other VNs. Given null-level policing, however, we can decide not to police at a higher-level netlet, because even if connections in the netlet misbehave, the problems will be limited to $\mathcal{N}_0$ only and not propagate to the outside world. (Note that the final responsibility of shielding different domains from each other, lies with the switch divider–this is why recursive partitioning in the MCA does not obsolete the partitioning by the divider.)

   We can now differentiate the policing policy in the network, e.g.: given that there is hard (in-band) null-level policing, we can decide to police specific netlets only very loosely (e.g. by periodically taking measurements from switches to see if they have exceeded their allocated bandwidths) and certain other netlets not at all. In fact, the *looseness* may vary from netlet to netlet. In other words, netlets are light-weight VNs (in this sense, the relation between a higher-level netlet and a VN is similar to that between a thread and process).

2. *Partitioning.* Using netlets, it is easy to separate immediate and future reservations as proposed in [3].

At the start of the reservation interval, the netlet resources are allocated to the application which requested them. Using simple operations the client can set up and tear down end-to-end connections in netlets. At the end of the interval, the MCA automatically tears down all connections belonging to the netlet and releases its resources. However, since the resources of a netlet are said to *belong* to a specific application (and nobody else), the application should be able to manipulate these resources in *any* way it wants to, not just by setting up connections between endpoints. For this, applications need control at a finer level of granularity than end-to-end. For example, we want to enable applications to set up a connection across an individual switch from a specific input (port, vpi, vci) to a specific output (port, vpi, vci). This allows applications to build their own connection types and setup mechanisms. We call such low-level operations *tertiary* operations.

5 Loading application-specific code

One problem with giving fine-grain control over network resources is that because of the distributed nature of the interaction between client and Sandman, it takes a long time to do simple things such as setting up a connection across a large number of switches (each low-level switch connection request travels across the network). An elegant solution for this problem is to enable the application to push its own management and control policy (limited to resources owned by the application) into the MCA and have it interact locally with the low-level control operations (in our implementation this interaction is very fast as it takes place within the same address space).

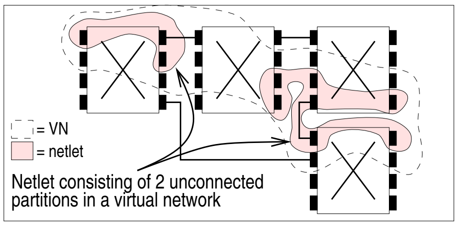
In other words, we enable applications to program the network using dynamically loadable *agents* (DLAs). Note that this is different from what is commonly called *active networks* [4] in the sense that it keeps a clear distinction between control and data path, while *active networks* are generally understood to interpret the packets on the data path.

5.1 Code and available operations

To enable clients to load code into the network, there is an operation in the Sandman's secondary interface which takes as arguments a policy and a start time t_{start} at which time the application wants the policy to be run. This is illustrated in Figure 4. At start of day, the implementation of the Sandman has two public interfaces: one for the primary operations and one for the secondary operations. The tertiary interface need not be publicly accessible. Note that restrictions may be placed on the number of DLAs allowed in the MCA as well as on the amount of CPU time each DLA gets. An interface to the DLA enables remote applications (e.g. the parent) to communicate with it. This allows DLAs to make arbitrary extensions to the core functionality of the Sandman. The operations that are made available to the DLA range from the usual operations that are available to normal applications (i.e. the primary and secondary operations) to the low-level tertiary operations.

5.2 Security

Running foreign code in the heart of the MCA introduces risks that range from the risk that the code steals or manipulates sensitive information, to the risk that a DLA uses up too much resource capacity. The former could be handled by careful shielding between the code and the rest of the Sandman while the latter can be dealt with by using an operating system such as Nemesis [5]. Another issue concerns the question of access restriction, i.e. which applications do we allow what sort of access to the MCA's functionality. The current implementation uses a capability-based access control scheme. In this document we will not address security issues any further.



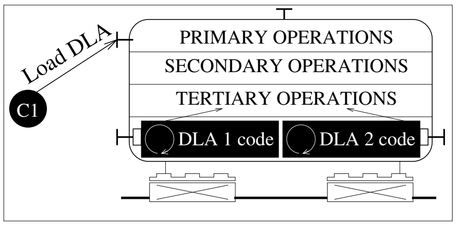


Figure 3 Netlet in virtual network

Figure 4 Partitioning switches

5.3 Combining resources with policies

Summarising, given the notions of *netlet* and *loadable code*, it is possible to associate application-specific behaviour with particular (sets of) netlets. Netlets and loadable code together enable applications to implement any resource allocation and management policy within the MCA domain that suits their environment. Note that the loadable code is also able to create new netlets (if need be recursively), which it can control itself as separate netlets, or associate with a DLA other than itself. Our experience with the loadable code feature of the Sandman has shown that it is extremely useful in prototyping and testing. We will demonstrate next how it can be used for implementing interoperability between multiple MCA domains as well as for establishing global policies.

6 MCA interoperability

As mentioned before, we do not intend the Sandman to replace any existing MCAs. Instead, we expect it to run alongside instantiations of Q.2931, IP switching, other instantiations of Sandman, etc. This makes interoperability an important issue. In this section we show how to achieve this. We stress that the issues are not specific to the Sandman MCA. Instead, they apply to interoperation between any two MCA domains. Consider Figure 5. In the figure, we see four different MCA domains, three of which are controlled by instantiations of the Sandman, while the one in the middle is controlled by some other MCA, e.g. P-NNI. We call this the MCA-X domain. The figure illustrates all four types of inter-domain interaction:

1. Sandman to Sandman, direct;
2. Sandman to MCA-X, direct;
3. MCA-X to Sandman, direct;
4. Sandman to Sandman, via MCA-X.

Note that it is sufficient to consider only the cases where communication originates in Sandman-1 and MCA-X. We assume that the Sandman MCA has only partial domain-level knowledge about the topology (or at least about that part of the total network that is of interest to it). By this we mean that Sandman-1 knows that endpoint 2 is connected to Sandman-2, but not what the exact topology within Sandman-2 is. Similarly, it knows that endpoint 6 is connected to the network controlled by Sandman-3 and that it can be reached through MCA-X.

The dashed line L3 between the Sandman-3 and the MCA-X domain indicates that there might be other domains between Sandman-3 and MCA-X of which Sandman-1 has no knowledge. When communication originates in MCA-X, we do not require the MCA to have even this knowledge. We will show that to MCA-X, Sandman-1 exhibits exactly the same behaviour as another instantiation of MCA-X, so that MCA-X can use its own proprietary signalling to set up connections to endpoints in Sandman-1.

6.1 Simple interoperability between domains

We first discuss a simple solution for interoperability between domains which resembles the one proposed in [6], which is to associate gateway code with a *pseudo-endpoint* that corresponds to the link connecting the two MCAs. A *pseudo-endpoint* is a *control gateway* that translates signalling messages from one MCA into those of another MCA. In Sandman domains it takes the role of an endpoint, while to a neighbouring MCA-X domain, it may look like a native MCA-X switch controller.

Under normal operation, where both endpoints lie in the same domain, the Sandman MCA sets up a connection from one endpoint to another (or a number of others) and then notifies the endpoints that the connection is in place. Things are different if one (or more) of the endpoints lie outside the local domain. Without loss of generality, we take the example of a point-to-point connection connecting A, a local endpoint (i.e. within the Sandman's domain), with B, a remote endpoint (outside the domain), as illustrated for two interoperating Sandman domains in Figure 6. When a request for such a connection arrives at the Sandman MCA, the pseudo-endpoint C of the appropriate outgoing link automatically takes on the role of the remote endpoint B. In other words, whenever a Sandman MCA tries to set up a connection to a remote endpoint, it really sets up a connection within its own domain, to the pseudo-endpoint corresponding to the outgoing link and then notifies the pseudo-endpoint that the connection is in place (and which vpi-vci values are associated with it).

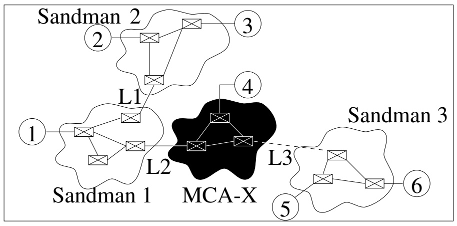


Figure 5 Multiple MCA domains

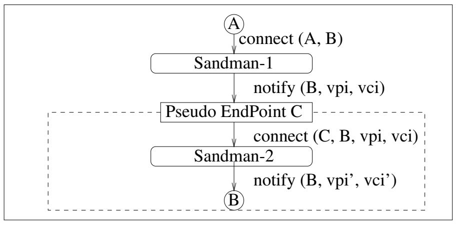


Figure 6 Control gateway

Upon receiving the notification, the pseudo-endpoint translates the setup request to whatever signalling protocol is used in the neighbouring domain (this includes address translation if necessary). If the neighbouring domain is another Sandman domain, it simply repeats the connection request, this time assuming the role of endpoint A. If the neighbouring domain succeeds in setting up the rest of the connection, the pseudo-endpoint returns an acknowledgement to the first Sandman MCA. If not, the connection set up has failed and all actions taken so far in the first Sandman MCA are rolled back also (the resources are released and the client is notified).

Connections from MCA-X to the Sandman could be set up in the same way if such a control gateway is implemented in the MCA-X domain. Alternatively, it is possible to let the Sandman offer the same sort of interface to the MCA-X domain that would have been offered by another MCA-X domain. This is illustrated in Figure 7. In this case, the MCA-X domain cannot tell that its is actually communicating with a different MCA. For example, many MCAs have well known channels for signalling. For example, ATMF UNI signalling uses a dedicated VC with VCI = 5 and VPI = 0. It is not difficult to direct this VC to the control gateway which then translates incoming signalling messages into Sandman requests.

This solution covers all four cases of interoperability mentioned in section 6. We call this the *hop-by-hop solution* for interoperability because each MCA only communicates with its immediate neighbour, translating each control message from its own domain directly into that of the neighbouring MCA.

6.2 Shortcomings of hop-by-hop solution

The hop-by-hop solution provides very basic interoperability between multiple MCA domains. The solution is attractive because of its simplicity but for the same reason limited in its usefulness.

The main problem is that the signalling gateways reduce all possible interconnection to the lowest common denominator in terms of MCA functionality. Consider, for example, the case of two Sandman domains, connected by one or more P-NNI domains, such as between Sandman-1 and Sandman-3 in Figure 5. Although both Sandman MCAs support the use of future reservations, it is impossible to make use of this functionality in an inter-domain connection. This is because at the control gateway between Sandman-1 and MCA-X (P-NNI), the future reservation request is translated into the type of request that P-NNI understands, e.g. immediate setup. After that it will never be 'promoted' to future reservation again. Instead, at the boundary between MCA-X and Sandman-3, the immediate setup request is translated into a Sandman immediate setup request. In other words, all functionality is reduced to the simplest common service on the path between Sandman-1 and Sandman-3. We call this the problem of *functionality degradation*.

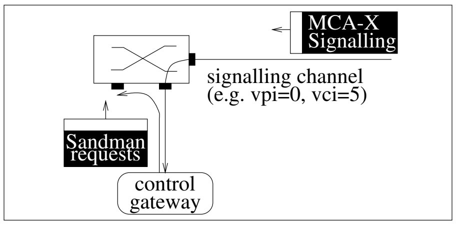
An additional problem is that the nature of the interoperation between two domains is fixed. This makes it hard to exploit application-specific knowledge. Again taking the example of future reservations, consider the case where endpoint 1 in Sandman-1 wants to reserve in advance for a connection from itself to endpoint 4 in MCA-X. The Sandman domain first makes all local future reservations for an interval $[T_{start}, T_{end}]$ and then injects the request into the MCA-X domain via the control gateway.

The control gateway has to translate the request into control operations that MCA-X understands. One option would be to simply allocate the resources (i.e. setup the connection) in the MCA-X domain immediately and keep it in place, so that at least the future reservation is guaranteed. This is the right solution if the guarantees regarding the availability of resources in $[T_{start}, T_{end}]$ are important and the resources in MCA-X are scarce. Alternatively, it may decide not to allocate any resources in MCA-X at all and simply *try* to set up the connection when it is needed at T_{start}. This may be the right solution if there is little risk of some other application using the required resources in the meantime. The point is that the gateway has to choose how the request is translated, while the chosen solution may be optimal in certain situations but not in others. It would be preferable if the application itself was able to specify the nature of the interoperation between two domains, allowing it to exploit application-specific knowledge that is impossible to support otherwise. We call this the problem of *fixed interaction*.

6.3 Sandman control channels and tunnels

We first address the problem of functionality degradation. We stress again the fact that the Sandman serves only as an example MCA—the exact same issues need to be addressed when interconnecting other types of MCA. Observe that functionality degradation occurs when multiple Sandman domains are on the paths between the endpoints and when these Sandman domains are separated by non-Sandman MCAs. If no other Sandman domains are involved, we can't do better than the hop-by-hop solution.

When multiple Sandman domains are involved, however, we propose to implement an inter-Sandman signalling channel (ISSC) between each two adjacent Sandman domains (possibly separated by a number of MCA-X domains). This is illustrated in Figure 8. As indicated in the figure, there is no need to dedicate a well-known vpi-vci value for the signalling channel. The channel can be set up simply using hop-by-hop inter-domain communication as described in section 6.1 (lowest common denominator is good enough for setting up simple signalling channels). All intermediate domains simply pass on the Sandman control messages without even looking at them (tunnelling). As usual, the ISSC finds its endpoints in the control gateways of both Sandman domains (in other words, the control gateways are the entities that signal to each other).



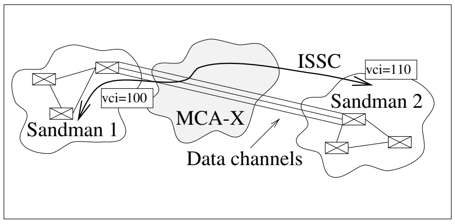


Figure 7 Signalling translation **Figure 8** Inter-Sandman signalling

Now, when Sandman-1 wants to communicate with Sandman-3 (Figure 8), it sets up a data connection between the two MCA domains. Ostensibly, the data connections also find their endpoints in the pseudo-endpoints described in section 6.1, so that the pseudo-endpoints (i.e. the control gateways) get notified when connections are set up which allows them to handle these connections further (outside the local domain). The pseudo-endpoints take care of the administration and maintenance of these connections. Inside the two Sandman domains however, these inter-domain data channels can be connected in any way the MCA wants to. So the data channels are really data tunnels connecting two Sandman domains. The further connection of these data channels on the remote side is controlled by signalling over the ISSC. Note that it is still not necessary for one Sandman domain to have precise knowledge of the topology of the remote domain: all routing is local to the individual domains. Note also that it is possible to set up data channels in advance or leave them in place after a certain application is done with them. We call this *tunnel caching*.

So to take up the example again of a reservation in advance for a connection from endpoint 1 in Sandman-1 to endpoint 6 in Sandman-3 (in Figure 5), this now becomes a matter of grabbing a data channel between the two Sandman domains and then making a local reservation in advance in Sandman-1 which is transferred over the ISSC to Sandman-3. Sandman-3's control gateway picks up the request and tries to make an advance reservation from link L3 to the eventual destination. If successful, it returns *true*. If not, the actions in Sandman-1 are also rolled back. At the start of the reservation interval, the connections are set up locally on both sides and connected to the VC of the chosen data channel.

We now have full interoperation between islands of Sandman domains, providing the full functionality of the MCA, while being interconnected by simple connections that act as tunnels. Note that we do not have to set up ISSCs from an originating Sandman domain to all other Sandman domains on a path between a source and destination. Instead, we again use hop-by-hop interconnectivity, albeit of a somewhat coarser granularity. Each hop is now a Sandman domain. Setting up connections end-to-end is done by sending the appropriate control message along the ISSCs from one Sandman hop to the next.

6.4 Loadable interoperability

We still have to address the problem of *fixed interaction*. For example, in the example of making a future reservation from a Sandman-1 endpoint to a Sandman-3 endpoint in section 6.3 it was assumed that the data channel between the two domains was set up immediately. This may be the right solution in certain cases but not necessarily in others (as shown in section 6.2). We now propose a very simple solution to this shortcoming.

Essentially, we allow applications to define their own pseudo-endpoints (if necessary with their own ISSC and data channels). For this we use DLAs, as discussed in section 5. So, users are allowed to load up their own gateway code dynamically. Of course, there have to be restrictions on this, as we don't want application-specific and maybe faulty pseudo-endpoint code to become the only available option for *all* applications. A simple solution is to enable users to associate their own pseudo-endpoint with a particular *netlet* (assuming the netlet owns capacity on the outgoing link). Now, whenever a connection to a remote endpoint is made in this netlet, the netlet gateway is used to communicate with the neighbouring MCA (as well as with the remote Sandman, using the netlet ISSC).

This allows applications to specify exactly the mapping between Sandman operations and the operations supported by the neighbouring domain. For example, a netlet gateway may decide not to map a future reservation onto an immediate connection in the neighbouring domain, choosing instead to wait until the start of the reservation interval (e.g., because it knows that bandwidth is unlimited in MCA-X).

7 Global policies

Section 5 described how applications can push DLAs into their local MCA, while section 6 showed how multiple MCAs can cooperate without suffering from functionality degradation. All this, however, is not sufficient for implementing truly global application-specific policies. In the interaction between domains, we have so far only been able to use the basic operations of the MCA. In other words, we may have been able to prevent functionality *degradation*, but we still haven't enabled clients to implement their own functionality *upgrades* across multiple domains.

We will first look at a property of DLAs which enables policies to spread over or migrate through a network. After that we will briefly look at operations that support migrated or replicated policies that have no knowledge about their new local domain. Finally, we will discuss very briefly an example to demonstrate the usefulness of global policies.

7.1 Policy migration and replication

An interesting property of the DLA support as discussed in section 5 is that it allows the DLA itself to push loadable code into other MCAs. This follows from the fact that starting loadable code is part of the secondary interface of the Sandman MCA where the secondary interface is publicly accessible. So, in a multi-domain network, the code is capable of sending DLAs across the wire which will then be run in the remote MCA. Using this mechanism the DLA is also able to migrate or replicate *itself* across a larger network (see Figure 9). Note that the various incarnations of the DLA distributed over the network, as well as different DLAs are still able to communicate.

It can be argued that a network operator running a MCA in a particular administrative domain will probably not want to allow code from applications in very different administrative domains to be loaded inside the heart of its MCA. Nevertheless, we feel that there are advantages in doing precisely this and that there is no intrinsic risk in doing so (provided the security issues described in section 5.2 are addressed).

But even if we accept that DLAs are not allowed to spread across multiple administrative domains, this does not mean that they are not allowed to spread over multiple MCA domains, as these are very different things. MCA domains only consists of an instantiation of the MCA together with one or more switches they control. In fact, the most common MCA domain consists simply of a traditional switch controller on a single switch. Therefore, there will generally be multiple MCA domains in a single administrative domain (which could be as large as a department). In this respect, the Sandman MCA is only different from traditional MCAs in that it offers a choice of how many switches one wants to associate with the MCA domain. This could be a single switch as in traditional systems, or small clusters of three or four switches. This is illustrated in Figure 10. Within the (fairly large) administrative domain, it is then perfectly permissible to have DLAs cross MCA domain boundaries.

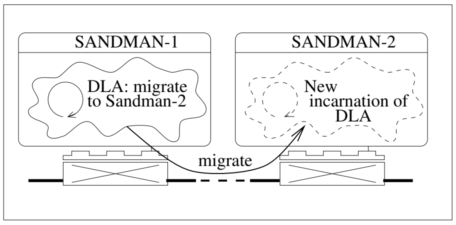


Figure 9 Policy migration

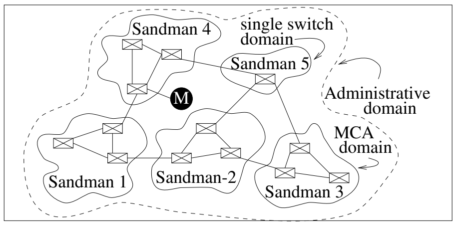


Figure 10 Domains and boundaries

7.2 Environmental awareness

It may be assumed that a DLA has rather extensive knowledge about the domain where it was created. For example, the topology of the network may be known to the application injecting the DLA and hence programmed into the DLA. Such domain knowledge can no longer be assumed when DLAs migrate through the larger network in order to implement global policies. A DLA may know that certain endpoints are connected to a particular MCA domain, but it generally has no knowledge of the switches and interconnections on the paths between these endpoints. This makes it impossible to exercise low-level control over the resources in this domain (using netlets), unless we provide the netlet with operations to learn about its new environment.

For this purpose, the tertiary interface contains a number of operations to allow DLAs to acquire knowledge about the new domain. One of these operations returns to the DLA (upon request) a detailed description of a path between two endpoints. Using this operation, a DLA can learn about the part of the MCA domain that is of interest to the DLA. The DLA can now start creating netlets consisting of exactly these paths, allowing it the low-level control that it may require to implement application specific policies in the new domain.

7.3 Example: mobile agents for mobile computing

We have shown how application-specific policies spanning multiple MCA domains can be introduced into the network. This allows applications to make use of application-specific knowledge that would otherwise be difficult if not impossible to exploit. To demonstrate the usefulness of these global policies, consider the case of mobile computing. A client may have very specific knowledge about the route followed by, or the communication pattern associated with a particular mobile system M (see Figure 10). We assume that there are multiple MCA domains and that the mobile system roams among these domains. We are now able to install the client's application-specific knowledge across the entire administrative domain. It is even possible to have a DLA 'follow' the mobile system as it travels from domain to domain (which means that it does not burden those parts of the network that M is not even close to). The DLA sets up connections for the mobile system, works out the routes for them and also collects data about the communication (e.g. for billing, etc.). This is an example of application-specific control using roaming policies of which a proof-of-concept implementation has been achieved.

8 Implementation details

The Sandman MCA has been implemented as a distributed system running over an implementation of CORBA. The DLAs are currently specified in the form of Tcl8.0 code. Experiments were also conducted with code specification in Java, but in our implementation the interaction between C and Tcl bytecode was slightly faster. Also, it is easier to quickly write and modify Tcl scripts (even on the fly) than Java programs. Even so, this is all just an implementation detail: there is no reason why the DLA should not consist of compiled C programs, so that there is no noticeable performance penalty to be paid when using loadable code.

The testbed consisted of a number of Sun UltraSparcs and ATM cameras, connected by Fore switches. The switches were partitioned and controlled using the switch dividers as described in section 2.

9 Related work

9.1 Programming the network

Several solutions to make network nodes programmable have been proposed over the last few years. We distinguish between solutions that maintain a clear separation between control and data path (such as the solution proposed here) and solutions where control and data path are the same. An example of the latter is what has come to be known as *Active Networks* [4]. Active Networks are packet-switched networks where each packet may carry executable code. In [4] these packets are called *capsules*, i.e. little programs with embedded data that are evaluated in a transient execution environment, allowing network nodes to process the data in an application-specific way. The execution of the packets in the data path is strongly related to the speed with which these packets can travel through the network. Note that the Operation, Administration and Maintenance (OAM) cells in ATM offer a similar, albeit much more restricted functionality (the OAM cell effectively carries one of a finite number of pre-defined programs).

Intelligent Networks (IN) [7] allows the introduction of new services by associating them with signalling endpoints. Basic calls are separated from IN-based calls. For example, dialling an 0-800 number will trigger a temporary suspension of call-processing and initiate a series of transactions between the local switching point (in IN terminology: the Service Switching Point or SSP) and the so-called Service Control Point (SCP), which is essentially a real-time database. A lookup in this database (e.g. for a 0-800 number) tries to find the corresponding application-specific *service logic*, i.e. the code which is then executed. The code sends back instructions to the SSP on how to process the call. The bulk of current IN transactions consist of translating the number dialled by the caller into another number depending on the needs of the service.

In [8] a solution to network management using *delegated agents* is proposed, where the agents are dynamically loadable code that can be dispatched using a so-called delegation protocol to an executing elastic (extensible) server. This helps prevent the explosion of management traffic from all over the network to a central site, which results from using management models that were designed when management was still a relative simple task and the traffic generated by it was minimal. Delegating management also makes the control loop (from managing code to managed device) smaller, decreasing the probability of failure at times when there are problems in the network (and management is needed the most).

Connection closures [6] are related to first-level netlets with associated behaviour. They provide for a way to open up the specification of an application's resource allocation behaviour. It is not possible to recursively partition the resources. Also, as indicated by the name, the connection closure associates application policy with a set of resources owned by the application. It is not possible to extend the MCA with new "public" operations (i.e. operations that can be used by any application in the system).

9.2 Interoperability

Efforts within the ATM Forum and the ITU-T have led to the definition of signalling interfaces between switches called the Network-to-Network Interfaces (NNIs). Of these, the ATM Forum's Private Network-to-Network Interface is intended for private networks and contains interfaces both for the exchange of routing information and for connection control [9]. The public NNI developed by the ITU-T serves as a demarcation point between two public networks. It is based on a modified version of Signalling System 7 and uses preassigned VCIs for signalling.

Closely related to this and the sort of interoperability discussed in this paper are the ATM Forum's efforts regarding the broadband inter-carrier interface (B-ICI). B-ICI is similar to a NNI except that NNI is really a switch-to-switch interface, designed to makes switches from different vendors work together, so that if these switches are located in the same network, the NNI needs to exist inside the network as well. B-ICI on the other hand is only concerned with internetworking between public carriers and need never exist within the network. B-ICI specifies a wide range of physical layers over which the ATM layer can run and also particular adaptation layers for common inter-carrier services such as Frame Relay.

In [6] a mechanism very similar to the simple hop-by-hop solution of section 6.1 is described. It is used to provide interoperability between a home-grown MCA and other MCAs.

10 Conclusions

In this paper we discussed the design and implementation of an elastic management and control architecture (MCA) called Sandman which enables applications to inject their own policies into the network in the form of DLAs. It allows very fine grain control over the resources in the network (up to the level of individual switch connections) and allows extensions by means of new user-defined operations accessible to any application. Also, here as elsewhere, we have maintained a strict separation between control path and data path. The resources in the network can be recursively (re-)partitioned at an almost arbitrarily fine granularity using netlets. A netlet can be associated with application-specific code, if this is so desired. A general solution for interoperability across multiple domains has been presented. The solution consists of a simple hop-by-hop mechanism in addition to dedicated signalling channels between Sandman islands (separated by other MCAs) that does not suffer from functionality degradation at the domain boundaries. Finally, we have shown how we can achieve application-specific policies spanning multiple domains by allowing dynamically loadable agents migrate from one MCA to the next, installing application-specific policy in each domain.

We believe that such an open approach to network control and management allows applications to exploit application-specific knowledge that would be difficult to capture by a fixed set of high-level primitive operations. This in turn allows for a class of operations that hitherto have been impossible to implement.

11 References

[1] K. van der Merwe, *Open Service Support for ATM*. PhD thesis, University of Cambridge Computer Laboratory, Pembroke Street, Cambridge CB2 3QG, U.K., Feb. 1998.

[2] H. Bos, "Efficient reservations in open atm network control using online measurements," *International Journal of Communication Systems*, vol. 11, pp. 247–258, Aug. 1998.

[3] A. Schill, S. Kuehn, and F. Breiter, "Resource reservation in advance in heterogeneous networks with partial atm infrastructures," in *Proceedings of INFOCOM'97*, Apr. 1997.

[4] D. Tennenhouse and D. Wetherall, "Towards an active network architecture," *ACM Computer Communication Review*, Apr. 1996.

[5] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The Design and Implementation of an Operating System to Support Distributed Multimedia Applications," *IEEE Journal on Selected Areas In Communications*, vol. 14, Sept. 1996.

[6] S. Rooney, *The Structure of Open ATM Control Architectures*. PhD thesis, University of Cambridge Computer Laboratory, Pembroke Street, Cambridge CB2 3QG, U.K., Feb. 1998.

[7] ITU-T, "Recommendation M.3010. Principals for a Telecommunications Management Network," *ITU publication*, 1992.

[8] G. Goldszmidt and Y. Yemini, "Delegated Agents for Network Control," *IEEE Communications Magazine*, vol. 36, pp. 66–70, Mar. 1998.

[9] ATM Forum, "Private network-network interface specification version 1.0 (p-nni 1.0)," *ATM Forum Document: af-pnni-0055.000*, 1996.