Service Specific Control Architectures for ATM

J.E. van der Merwe and I.M. Leslie

Both authors are with the University of Cambridge Computer Laboratory

Manuscript received 10 January 1997; revised 28 August 1997

Abstract

The concept of a *Service Specific Control Architecture* (SSCA) in an ATM environment is presented. Here a Control Architecture denotes the out-of-band control and management mechanisms operational in a network or virtual network (or part thereof). An SSCA can use knowledge about the applications it serves to make better use of network resources and thus provide a more efficient service. Such a control architecture can be designed to provide only the subset of control functions required by the applications it serves, thereby greatly reducing its complexity. The *switchlet* mechanism, which allows more than one control architecture to be operational simultaneously within the same network, enables the use of service specific control architectures. Implementation work is presented to illustrate the use of switchlets and SSCAs. This includes both an environment which uses switchlets to dynamically create virtual networks, and an SSCA used to provide video and audio conferencing facilities in an ATM environment.

Keywords

Asynchronous Transfer Mode, Signalling, Control Architecture, Virtual Networks, Videoconferencing.

I. Introduction

The conventional approach to control in asynchronous transfer mode (ATM) networks has been to have a single monolithic general purpose control architecture[1] [1], [2]. This general purpose control architecture then tries to provide the functionality required by all current and anticipated future users of the network.

In contrast, this paper presents the concept of a Service Specific Control Architecture (SSCA). An SSCA utilises knowledge of the requirements of the applications it serves to control the network. In so doing it can make better use of network resources and provide a more efficient service. This concept is demonstrated by means of a relatively simple control architecture, which has knowledge about the applications it serves and can solve problems which are difficult to address with a general purpose control architecture.

Section II justifies the approach of SSCAs by comparing it to conventional general purpose control architectures, and by examining its relationship with such control architectures. A number of example environments, which stand to benefit from SSCAs, are discussed.

[1]In this paper, *control* is taken to mean all the signalling and management activities within a network, and the mechanisms that constitute a particular control and management approach are collectively called a *control architecture.*

Manuscript received 10 January 1997; revised 28 August 1997

Section III describes an implementation of a Dynamic Virtual Network environment which provides the enabling mechanisms to support SSCAs. The possibility of such a dynamic virtual network environment was first proposed in [3].

In Section IV, an example SSCA is examined in detail. The specific service considered is that of multi-party conferencing with high quality continuous media (audio and video). This is a service that is supposed to be ideally suited to the ATM environment, and yet very few (if any) solutions exist that will scale well into the wide area network environment. The design and proof of concept implementation of the VideoMan SSCA is presented.

Section V discusses related work and Section VI concludes the paper.

II. Motivation

In the traditional approach to control in ATM networks, an application is presented with a single User Network Interface (UNI), through which all communication with a single general purpose control architecture takes place [4]. As the requirements of users and applications change, modifications are made to the control architecture which are reflected in associated changes in the UNI. While this approach is understandable, and indeed desirable in order to provide interoperable implementations, it is built on the basic assumption that all applications require the same functionality from the network. Unfortunately, however, this "one size fits all" approach suffers from several drawbacks:

- The vocabulary of interaction across the UNI needs to be sufficiently large to deal with all ways in which users might want to communicate. Dealing with all such user requests in a single control architecture complicates its design and implementation.
- As new applications with new requirements are developed, both the UNI and the control architecture must change, or alternatively, applications must adapt to inadequacies of the control architecture.
- Because of its monolithic nature, any change in either UNI or control architecture, by necessity means network and/or application downtime, because the old architecture must be replaced by the new.
- Any such changes to the control architecture must take place in a synchronised fashion over the whole network[2].

[2]Backward compatibility is often attempted when smaller changes are introduced and can sometimes reduce the

In addition to the above shortcomings, a general purpose control architecture, by its very nature, cannot exploit characteristics of certain applications. Furthermore, in some cases, the functionality provided by a general purpose control architecture simply does not meet the requirements of some ATM capable devices. In these instances the only viable solution is the use of an SSCA. Before the mechanisms for using SSCAs are considered, a number of example environments that might benefit from this approach are presented.

A. Example Environments

Video conferencing in an ATM environment can benefit from an SSCA in at least two ways. First, the limited number of active participants in a conferencing session can be exploited to make more efficient use of network resources. This is particularly important in a high quality conferencing session, where network bandwidth can very easily become a scarce resource. Second, manipulation of connections *within* the network, under direction of the floor control function can avoid time consuming re-establishment of multicast trees, which would be the only alternative in a conventional ATM environment. The design and implementation of an SSCA for this environment is considered in detail in Section IV.

Because of its attractive quality of service (QoS) guarantees, the ATM environment has seen the development of several "custom made" ATM capable devices. In general these devices have varied processing capabilities, which means that the implementation of a general purpose control stack within the devices, is often not feasible. More important however, is the fact that often the functionality provided by a general purpose control architecture is not sufficient for their purposes.

One example of such a device is an ATM to SCSI[3] adapter [5]. These devices provide an adaption facility between a SCSI bus and an ATM network at a very low level. One way of using this adaptor is as a transparent extension of the SCSI bus across an ATM network. This means that commands and data sent across the SCSI bus are converted to ATM cells which are sent across an ATM network. A similar device on the other side of the network applies the inverse translation and converts the ATM cells into SCSI

need for network wide change.

[3]Small Computer System Interface (SCSI) is a peer-to-peer protocol for transferring data between up to sixteen devices connected to a common bus.

commands and data on a second SCSI bus. The adapters can be connected across the ATM network by means of permanent virtual connections (PVCs). The static nature of such a solution will however be unsatisfactory in all but the simplest applications. Clearly it will be difficult for these devices to use a general purpose control architecture because the connection setup rates required in this environment are much more demanding than in a normal ATM environment. But ATM is the ideal in-band transfer mechanism because of the QoS guarantees that it can provide.

The digital manipulation of high quality video and audio by broadcasters and the movie industry, is another example environment which would benefit from an SSCA. Again the QoS guarantees of ATM make it an ideal transfer mechanism for these applications [6]. A specific problem domain is the editing and playback of digitally stored video and audio clips, without excessive copying and duplication of data. This task can best be performed using an SSCA, which can schedule the distribution and delivery of these files across the ATM network according to current loads on particular servers and network nodes.

A different approach to the same problem of audio and video editing in a networked environment would be to have a large number of moderately powerful file servers each equipped with an ATM adapter and hard disc. Each of these file servers would then serve as a mini file server for a few audio and video files. A number of editing stations connected to the file server clustered by means of an ATM network completes the picture. The aim of this approach is to ensure that no video and audio files are ever copied, other than during their initial storage onto a file server. Instead an SSCA is used which keeps track of the editing of files, and then when clips of different files are "played back" as a single entity, establishes and switches between connections to ensure correct delivery of the final sequence of clips. We can view the SSCA as either interacting with, or being part of, the storage system.

The usefulness of the SSCA concept is not limited to the fairly esoteric examples above. For example, when carriers provide telephony services over ATM[4], it might be simpler to use existing telephony signalling, accounting and other software[7]. These can be applied to a separate ATM virtual network, rather than re-engineering it to fit standard ATM

[4]Such a service might use ATM right to the endpoints (telephones), rather than just using ATM in the backbone.

signalling [1], [2].

Finally, one can envisage the situation where a service provider provides a standard compliant service, but customises or enhances it to satisfy the special needs of its customers, and to differentiate it from services offered by its competitors. This approach has the advantage of being standards compliant if interaction with other service providers is required, but not being restricted by standards if services are only offered to the provider's own customers. For this example, the real strength of the approach presented in this paper comes from the possibility that a service provider can lease an empty virtual network and control it by means of its enhanced SSCA. The service provider can thereby provide the customised services to its customers even in places where it does not own the physical infrastructure.

B. Discussion

Other examples of SSCAs can be found in the areas of mobility and home networking. In almost all cases the potential usefulness of an SSCA stems from either or both of the following reasons:

- To perform some tasks efficiently, the need arises to manipulate connections within the network based on some service specific knowledge, or a standard control architecture simply does not provide the required functionality.
- ATM endpoints are not intelligent and are incapable of implementing a normal control stack.

Of course, all the functionality required by these examples can be pushed into a general purpose control architecture such as UNI/PNNI [1], [2]. However, this will only tend to make the already complicated general purpose control architectures even more complex, whilst the functionality is not required by all network users.

One could argue that the second reason given above will disappear as technology develops; that all end systems will be powerful enough to contain a complete general purpose signalling system. However, this places a limit on what an end system will be (e.g. a doorbell or an active badge detector), and adds gratuitous complexity to all end systems which greatly changes the dynamics of deployment.

It is possible to build dedicated ATM networks with special control architectures to achieve some of the same benefits described above. This is not sensible because, for example, an ATM workstation taking part in a video conferencing session could reasonably expect to use its ATM interface for functions other than video conferencing. Similarly, in the file server cluster approach to the continuous media editing problem, both file servers and editing stations might require the use of the same ATM network as a more general purpose network. There is thus a need to be able to run both a general purpose control architecture, as well as one or more SSCAs. The recently developed *switchlet* mechanism [3] solves this problem by facilitating the simultaneous operation of more than one control architecture within the same network and indeed within the same end system. In this manner it also obviates the need for the hard partitioning solution of a dedicated network approach.

An overview of the switchlet approach and mechanisms will be given in the next section.

Figure 1 depicts the relationship between service specific and general purpose control architectures in an example ATM network. In this network two local area ATM networks are connected by means of an ATM backbone. A general purpose control architecture is assumed to be running on all switches and on all workstations, and is shown with a black circle. A continuous media editing SSCA provides services for a number of ATM capable fileservers and workstations, and is completely contained within one of the local networks. Finally, a video conferencing control architecture provides services for ATM video equipment and workstations on all three ATM networks.

As indicated, some SSCAs are expected to be contained within private networks, while others are expected to be particularly useful in the wide area. It would be possible to statically partition and configure networks to run different control architectures. However, a more dynamic approach whereby virtual networks with appropriate SSCAs are only created when needed, holds many advantages[5]. For example, in the wide area video conferencing case, it would be much more sensible to release the resources associated with a conference session when the conference is over. In this manner unused resources are not wasted, and can in fact be made available to other virtual networks. Such a dynamic

[5]The more general case where resources can be shifted between existing virtual networks has also been investigated, and is presented in [8].

approach is followed in the design and implementation of the video conferencing control architecture presented in Section IV. The next section therefore gives a short overview of the mechanisms involved with *switchlets* and dynamic virtual networks.

III. Dynamic Virtual Networks

A. Switchlets

In [3], a mechanism was presented which allows the physical resources on a switch to be partitioned. Each subset of resources, which is called a *switchlet*, is presented to a different controller forming part of a different control architecture. These control architectures can be different instantiations of the same control architecture, or could be completely different control architectures (or indeed service specific control architectures). A *divider entity*, which is implemented in software, is associated with each physical switch and creates the partitions to produce the switchlets. The divider also polices all invocations made by a controller to ensure that it will only influence resources in its allocated switchlet. Figure 2 depicts this arrangement.

Switches are nodes within networks, and the division of switch resources into switchlets means that virtual networks are effectively created. (Each of which has a different instance of a control architecture.) If the creation of switchlets and virtual networks is automated, it means that virtual networks can be created dynamically. The next sections present an implementation of such a dynamic networking environment, and Section IV shows how this can be used by an SSCA.

B. Creating the Virtual Network

In order to be able to dynamically create switchlets, the existence of a *bootstrapping virtual network* which spans all physical switches in the network is required. The divider entities, which create switchlets, are then all accessible from within this bootstrap network. By invoking appropriate commands on the divider entities (potentially remotely), human or software entities can dynamically create switchlets and virtual networks. The exact nature of the bootstrap virtual network (and control architecture) is not important, and an existing IP-over-ATM network is used for this purpose in the implementation. The use of IP in the bootstrap network also has the advantage that existing facilities from this environment can be used without modification[6]. For example, a distributed processing environment (DPE) which requires IP as the underlying transport mechanism is used in the implementation. The use of a DPE is not a requirement, and a solution based on well defined messages can be engineered easily enough. The facilities provided by a DPE are assumed in the rest of the paper.

The divider entities are encapsulated in a service called a *divider server*. The server exports management interfaces which allow the creation and destruction of switchlets. When a switchlet is created, the divider server also exports a switchlet control interface. The control interface allows a control architecture to manipulate resources on its allocated switchlet. These interfaces are shown in Figure 2.

Two other servers are required for dynamic network creation namely a *trader* and a *network builder*. In a DPE, trading is the way by which clients find out about the existence and capabilities of servers [10]. This functionality is normally provided by a DPE and is not considered in more detail. The role of the network builder is to create the switchlets, and to then associate appropriate switchlets with a particular virtual network.

Figure 3 shows the interactions involved in creating a virtual network. Some requesting entity (e.g. an SSCA) contacts a *trader* to find a *network builder*, **(1)**. The requesting entity then contacts the network builder with a specification of the required virtual network, **(2)**. The network builder has knowledge about existing virtual networks within a specific domain, and coordinates the creation of virtual networks so that, for example, the virtual channel identifier (VCI) and virtual path identifier (VPI) allocations for interconnected switchlets are performed in a consistent manner. The request to the network builder can either be to create a new virtual network, or can be a request to modify an existing network in some way. For example, in the case of video conferencing, switchlets can be added piecemeal to the virtual network as the control architecture receives requests from endpoints which are not connected to any of the current switchlets.

The network builder now consults the trader to find a divider server which can create switchlets according to the supplied specification, **(3)**. (The divider servers are assumed to have registered with the trader at an earlier time.) It then performs appropriate invo-

[6]Note that the use of IP in the bootstrap network does not necessarily imply the complexity of schemes such as classic IP-over-ATM. Indeed the use of IP switching [9] as the bootstrap network is currently being investigated.

cations on (potentially) a subset of the divider servers, in order to create the switchlets, **(4)**, and returns the result to the requesting entity. The result returned will, amongst other things, contain information about the topology of the virtual network, as well as interface references to the switchlets in the virtual network[7]. This enables the requesting entity to control and manage the created virtual network.

Following the creation or modification of a virtual network the divider servers in question have to update their current state to the trader, **(5)**. The exact nature of the interfaces exported by the different services are considered in the next section.

Dynamically creating ATM virtual networks in the manner described above, results in three time scales of interest in the network: Cells are manipulated at very short time scales, normally in hardware. At a longer time scale connections are created within a switchlet or virtual network, by a control architecture. These control invocations are subject to policing by the divider entity in the switchlet environment. Third, at an even longer time scales, virtual networks are created and modified by the network builder entity. In order to make efficient use of the available resources, scheduling and access control mechanisms are required at all three time scales.

C. Implementation

An implementation of the dynamic virtual networking environment has been performed using the switchlet implementation as reported in [3]. The only switch resources considered by this switchlet implementation were switch ports, and VPI/VCI space. This was a deliberate implementation decision to avoid dealing with QoS issues on switches which implement QoS in different ways.

The ATM network used for the implementation forms part of a departmental service ATM network. In particular, the subset of switches used in this work consists of three Fore Systems ATM switches (one ASX-100 and two ASX-200s). The ATM network interconnects a number of file servers, workstations and ATM video adapters (Fore Systems AVA-200s). An early implementation of the Distributed Interactive Multimedia Architecture (DIMMA) was used as the DPE [11]. DIMMA is a framework Object Request Broker

[7] In a DPE an interface reference provides the means for a client to invoke an operation on an interface exported by a server.

(ORB), which provides a common base for the construction of domain specific ORBs.

Figure 3 shows three divider servers each of which corresponds to a single physical switch. The divider server Manage interface consist of the following two sub-interfaces:

• `Dynamic`: This interface has two methods which allow the dynamic creation and destruction of switchlets. The current implementation does not allow the characteristics of a switchlet to be specified, instead a limited number of "default" switchlets can be created. A side effect of creating a switchlet is that the VCI space allocated to the switchlet will be "cleaned up", i.e., all connections falling in the range allocated to the switchlet are released, so that the VCI space is presented to the control architecture in a known state. Clearing the VCI space in this way dominates the time it takes to create a switchlet in the proof of concept implementation: To create a switchlet with 7 ports and 1 VPI with 40 VCIs per port takes less than 40 ms.

• `Configure`: Allows a network administrator to find out what the current configuration of the divider server is. Methods are provided to show the current switchlets, as well as the resources used by each.

As shown in Figure 2, the divider server also exports a composite Open Control switchlet control interface for each switchlet that is created. The Open Control interface contains the following sub-interfaces and methods:

• `Configuration` - The `Configuration` interface is primarily used to *find out* what the configuration of a switchlet is. It contains methods which allow the controller to obtain the capabilities of the switchlet in a per port list. The capabilities include the VPI and VCI ranges and the ATM Forum service categories supported.

• `Connections` - The `Connections` interface is responsible for basic VPI/VCI mapping. Methods are provided to create and delete connections at VP and VC level, as well as to obtain lists of all current connections.

The implemented switchlet control interfaces and methods presented above are a subset of those which will be required in a fully functional implementation. In particular interfaces will be required for per port management, statistics and alarms. If methods for manipulation of QoS related resources on the switch are not part of the `Connections` interface, then a separate interface for these will have to be provided. (See [3] and [8] for more detailed discussion of a switch(let) control interface.)

As mentioned above the network builder keeps track of all current virtual networks in the system. The network builder is also the only entity which is allowed to create and destroy switchlets using the `Dynamic` interface on the divider server. This allows the network builder to perform garbage collection on virtual networks for which the control architecture malfunctions. Because users are allowed to supply "their own" control architectures, this is an important function to ensure the integrity of the network as a whole. In the current implementation a trivial access control mechanism is applied, whereby a list is kept of users who are currently allowed to create virtual networks. This list is updated by the "system administrator" who also starts up the divider servers and the network builder. This list is consulted by the network builder when a user requests creation of a virtual network as described below[8].

The network builder implementation is depicted in Figure 4. The network builder exports two interfaces namely, `Dynamic` and `Topology`. The `Dynamic` interface facilitates the manipulation of virtual networks, while the `Topology` interface provides topology information to control architectures that do not have their own topology services.

The methods provided by the `Dynamic` and `Topology` interfaces are as follows:

• `Dynamic`: This interface allows for the creation and removal of virtual networks. In addition it allows the modification of existing (active) virtual networks:

– *createNamedNetwork*: If successful, this method returns a network reference number to the requester. The network reference number is unique within a particular network builder, and is used as an argument to subsequent invocations on the network builder to identify the appropriate virtual network. A virtual network created with this method contains no switchlets, and switchlets have to be added by appropriate invocations on the methods below.

– *getNamedNetwork*: In this case a set of switchlets for a predefined virtual network topology is created, and returned to the caller together with a network reference number.

[8] This trivial access control mechanism, useful though it is in the current experimental development environment, would be completely inadequate in a commercial implementation. The ultimate objective is that "any" authorised network user could obtain a virtual network, and apply a control architecture to it, and thereby effectively become (or enter the role of) a network operator. It would therefore seem that role based access control mechanisms, such as those implemented in [12], would be a natural fit for this environment.

This method is for control architectures which cannot make use of the facility to add switchlets to their virtual network one at a time.

– *addSwitch*: Requests the network builder to add a switchlet for a named switch to a previously created virtual network.

– *removeSwitch*: Requests the network builder to remove a previously allocated switchlet from a virtual network.

– *removeNetwork*: Requests the destruction of a virtual network. The network builder will remove all switchlets in the virtual network which have not been removed already.

• `Topology` - In the current implementation topology information is entered into a topology database, which the network builder reads at startup time. (In a commercial implementation, this static procedure would be replaced with a dynamic topology discovery service.) In the database, switches and endpoints are identified with a human readable name or string. A switch is given a system wide unique identifier, while an endpoint is identified within the network by the switch identifier and switch port to which it directly connects. This topological information is made available to control architectures through the methods described below. This information can be used by control architectures which lack their own topology discovery services. The VideoMan SSCA presented in Section IV is an example of such a control architecture.

– *getSwitchTopology*: Provides complete switch topology information according to the network builder database. A switch topology information element consists of a switch identifier, and a list of (remote switch, local port, remote port) tuples. A sequence of these elements is returned by the network builder.

– *getEndpointAddress*: Provides a (switch identifier, port number) pair, which together constitute an “endpoint address”.

– *getSwitchName*: Returns the switch name given a switch identifier.

– *getSwitchId*: Returns the switch identifier given a switch name.

As indicated in Figure 4, the network builder maintains an internal database of the current active virtual networks. This per virtual network information include the switchlets in each virtual network. A control architecture can therefore simply invoke the *removeNetwork* method, upon which the network builder will release all resources currently held by the virtual network.

The next sections show how this dynamic virtual network environment can be used by an SSCA to create, modify and control a virtual network based on the requirements of the virtual network users.

IV. VideoMan - An SSCA for Continuous Media Conferencing

Having motivated the need for and usefulness of SSCAs in Section II, and having explained the switchlet and dynamic virtual network environment in the preceding sections, this section explores an example SSCA in detail, and presents a proof of concept implementation. The SSCA presented is called VideoMan and provides a control architecture for a continuous media conferencing environment.

A multi-party conferencing environment has several characteristics which can be exploited by an SSCA to provide a scalable solution that efficiently utilises network resources. These include:

- Most participants are data *consumers* for the complete duration of the conference.
- Some participants become audio data *producers* for fairly short periods of time.
- At any particular moment a limited number of audio data producers should be active within an orderly conference.
- Of all the potential video data producers (all participants), only a limited number will actually be of interest. (For example the video of the current and previous speaker.)

The VideoMan control architecture drastically reduces the network resources required for video conferencing by creating novel connection structures in the ATM network. Connection structures and endpoints are also manipulated by the control architecture as directed by the *floor control* function to ensure that QoS guarantees can be maintained for all media flows. Floor control is the process of deciding which set of participants should be producers and can be achieved in a variety of ways, such as chaired conferences or audio triggered conferences [13], [14]. The following subsections describe the VideoMan architecture in detail: Section IV-A explains the novel tree structures created in the network, and Section IV-B describes how VideoMan fits into the switchlet environment. The section ends with a comparison of VideoMan with other approaches to video conferencing.

A. Gather and Distribution Trees

VideoMan uses the inherent multicasting and inverse multicasting[9] capabilities of ATM switches, as well as the previously mentioned application specific knowledge to control the network.

In particular the control architecture creates sets of *gather* and *distribution trees* as depicted in the example network in Figure 5. For simplicity the figure only shows a single set of tree pairs. The top part of the figure shows the example topology, while the bottom part "unfolds" the network into two mirror images for clarity. Similar tree structures have been proposed to do multicasting in the IP based networks [15], and indeed more recently in the ATM environment [16]. The interaction required within the control architecture to create the tree pairs is considered in a later section[10].

In Figure 5 it is assumed that all endpoints are potentially producers and consumers of data. As indicated the gather tree uses the inverse multicasting, and the distribution tree the multicasting capabilities of the switches. The final requirement is that the control architecture must ensure that at any moment in time only one producer is active in a particular set of trees to avoid the interleaving of cells from different producers on the same inverse multicasting VCI. The control architecture can achieve this by either having some control over the endpoint, or by disabling the first branch of all but one endpoint in a particular gather tree. This ability of the control architecture to directly manipulate network resources is a key feature of the VideoMan control architecture, and distinguishes it as an SSCA.

Note that this is not meant to be a general solution to the abstract problem of many-to-many multicast. It is a system to avoid solving the general problem while solving the specific problem of the application in hand, i.e. continuous media conferencing. At the

[9] Inverse multicasting is the ability of some switches to map several incoming virtual channel identifiers (VCIs) onto the same outgoing VCI. The architecture is not critically dependent on this capability, but inverse multicasting does allow for a symmetric solution which requires fewer network resources than if it is not available.

[10] It is worthwhile to note if an ATM switch separates the functions of "allocating resources" and "using resources", and further allows different virtual circuits to share the same resources (i.e. police them as a group rather than individually), then it would be possible to create a mesh that would not use excessive switch resources, and yet still provide QoS guarantees under some conditions. Such switches, however, are uncommon, and a different approach is required.

same time, the VideoMan resource management provides a functionality similar to the general merging operations in RSVP [17]. Indeed, if layered encoding is used to effect the merging facility of RSVP, then the gather and distribution tree structures would be the natural way to implement this in an ATM environment. The use of similar tree structures have recently been proposed to solve the general multipoint-to-multipoint problem in ATM [18], [16]. Unlike the VideoMan, which requires no changes to physical switches, these approaches both require changes to the ATM data path, and will be discussed in Section V.

The same basic scheme outlined above can be used to scale the architecture across various sites. A control entity at a particular site will be responsible for creating gather and distribution trees locally. The control entities at different sites cooperate to create distribution trees between them, with the incoming branch of a remote site being connected into a local distribution tree. Floor control is performed by interaction between the control entities at different sites[11]. Looping back a remotely produced stream to its originating site is prevented by obeying the following simple rule: If the active source is within the local site, then patch in the relevant remote distribution tree into the local distribution tree, otherwise disconnect all remote distribution trees.

Figure 6 shows a two site example, where endpoint **2** at site **II** is the current active producer. The remote distribution tree whose source is at switch **C** at site **II** is enabled, while the reverse remote distribution tree is disabled. (The intersite distribution trees in this simple example constitute a single branch each.) Disabling a tree involves only the invalidation of a single VCI table entry at the source of the tree while the bulk of the tree stays intact, i.e. it does not involve the tearing down of a complete end-to-end connection, or multicast tree in the traditional sense.

In the discussion above the function of a single set of trees was described. Clearly a set of trees will be required for each media stream that needs to be consumed by an application. For example, a conferencing application might require an audio stream from the current speaker, a high quality video stream from the current speaker (main video window), and a lower quality video from the previous speaker (secondary video window), each of which

[11] In a chaired conference this interaction can be either peer-to-peer or hierarchical. It is unlikely that any distributed floor control implementation will be fast enough for voice activated conferencing.

will require the creation of both a gather and a distribution tree.

If switches do not support the inverse multicasting capability, it is always possible to create unicast connections from each producer to the core of the tree, and have the control architecture patch the appropriate unicast connection through to the distribution tree at the appropriate time.

Note that in the case of a chaired conference the activation and distribution of audio from the current speaker can be handled by means of an appropriate set of gather/distribution trees. However, when audio triggered floor control is used all audio sources must by necessity be active all the time. These streams are sinked and analysed at a single point by the floor control function to determine which video source should be activated. At the same time the different audio streams can be mixed into a single combined audio stream, which can be sent to audio sinks by means of a distribution tree. The analysing and mixing functions need not be performed by the same entity, in which case a two-branched multicast can be created from each audio source, with one branch going to the floor control function and the other going to the mixing function. If the format of the audio streams can be controlled so that audio samples are sent in self contained cells, a gather tree, rather than several unicast connections, can be used to transport audio streams to the analysing and mixing entities.

It is important to note that the control architecture utilises application specific knowledge in its manipulation of network resources. This is crucial to its correct functioning, and would be impossible to achieve using a general purpose control architecture.

B. VideoMan within a Virtual Network

The previous subsection explained the tree structures that the VideoMan control architecture maintains within the network. In this subsection the creation and maintenance of these tree structures are explained in the context of a video conference in the switchlet environment.

Having obtained a virtual network, as described in Section III, a service specific control entity, which is an instantiation of the control architecture, can start to provide services to applications. Interface references to switchlet control interfaces, returned by the network builder, are used to manipulate the virtual network resources.

The interaction between users and the control architecture, as well as the interaction between the control architecture and switchlets is depicted in Figure 7. To simplify the diagram, the interaction with the network builder to instantiate and modify the virtual network is not shown. Similarly, the existence of a trader is assumed to allow clients (applications) to find out about servers (the control entity), but is not shown in the figure.

Applications register with the control entity as either producers or consumers of data or both, shown by invocation **(1)** in Figure 7. An application also provides the control architecture with the name of its host or endpoint. This name is then turned into an endpoint address and used by the control architecture to determine whether appropriate switchlets already exist within its virtual network to accommodate the requesting application. If not, appropriate invocations are made on the network builder to add switchlet(s) into the virtual network.

For the sake of simplicity it will be assumed that only a single set of gather/distribution trees are required, although as indicated in the previous section several sets of trees will typically be required.

If an application registers as a consumer of data, the control entity determines where in the distribution tree the endpoint should be added, carries out the appropriate invocations on the switchlet(s) to do so, **(2)**. Depending on the situation, appropriate invocations might be adding a branch to a multicast tree, or the creation of a new tree. The control architecture then returns to the application a *service access point* on which it can start receiving[12].

Similarly, when an application registers as a producer of data, the control architecture performs the required invocations on the switchlet control interface(s) to add the endpoint into the gather tree. Additionally when an application registers as a producer, it is required to provide the control entity with a *control interface* reference. This control interface reference is used by the control entity to grant or invoke the right of a producer to send into the gather tree, **(3)**. This is used to ensure that only a single producer is active at any moment in time. In order to protect the conference from misbehaving producers, or to

[12]In a DPE which supports both stream and operational interfaces, a service access point would correspond to a bound stream interface. In the proof of concept implementation in the next section, a service access point is simply the value of a VPI/VCI pair on which the application can start receiving.

cater for less sophisticated devices, the control entity might want to disable all gather tree endpoints except the branch which ties the current active producer into the tree. Again this is a simple local operation, and the bulk of the gather tree can be kept intact.

C. Implementation

This section presents a proof of concept implementation of the VideoMan control architecture. The implementation used the same ATM environment as described in Section III-C. The control architecture again used the DIMMA framework [11] as DPE to provide an implementation environment. This is not a requirement however, and the control architecture could have been implemented with any other message passing, or client server implementation[13].

The current implementation creates a single distribution/gather tree pair, which is used to distribute a single video stream from any number of AVA devices (sources), to any number of workstations (sinks). This can trivially be extended to create any number of trees in a fully functional implementation. The control architecture ensures that only one of the AVA devices is active within the gather tree by directly controlling the device (in fact, a manager of the device). The operation of this system is described below.

At startup the control architecture obtains the current switch topology by invoking *getSwitchTopology* on the `Topology` interface of the network builder. Based on this information it selects a core switch. The current implementation simply selects the switch with the most neighbours. The control architecture then proceeds to determine the shortest path from the core switch to all other switches. A breadth first search algorithm is used to calculate shortest paths. At this point the control architecture invokes *createNamedNetwork* on the `Dynamic` network builder interface, to create an "empty" virtual network, and awaits requests from applications to join the video conference.

The VideoMan control architecture exports a single `ConferenceUser` interface with the following methods:

• *registerAppl*: Invoked by an application to join a conference. The control architecture passes back to the caller a conference-unique application identifier which is used in all

[13] Interaction with the dynamic virtual network world would still require DIMMA interaction, however this is provided in the form of a set of libraries.

other invocations.

• *unregisterAppl*: Signals to the control architecture the intention of this application to leave the conference. The control architecture uses this information to prune any connections associated with this application from the trees.

• *sinkJoin*: Joins a conference as a sink. The control architecture is also provided with a reference to a `MediacontrolSink` interface on the application (see below).

• *sourceJoin*: Joins a conference as a source. The control architecture is also provided with a reference to a `MediacontrolSource` interface on the application (see below).

The `MediacontrolSink` and `MediacontrolSource` interfaces provide the following methods:

• `MediacontrolSink`: The methods provided by this interface allow the control architecture to control consumers of information, for example based on who the current active speaker is:

– *receiveOn*: Provides the application with a service access point (a VPI/VCI pair) on which it can start sending.

– *stopReceiving*: Instructs the application to stop receiving on a particular service access point.

• `MediacontrolSource`: The methods provided by this interface are used by the floor control function of the control architecture to ensure that a single producer is active in any tree pair:

– *sendOn*: Instructs the application to start sending on a particular service access point.

– *stopSending*: Stops the application from sending on particular service access point.

The current implementation provides a simple test "floor control function", whereby the control architecture sequentially steps through the list of producers at four second intervals, and selects one to be the current active producer. It invokes *stopSending* on the `MediacontrolSource` interface of the current active producer, and then invokes *sendOn* on the `MediacontrolSource` interface of the newly selected producer. In this proof of concept implementation the time it takes to switch to a new producer is completely dominated by the communication overhead involved with making the two remote procedure calls, which varies between 2 and 5 ms for the pair, depending on the speed of the workstations involved.

The emphasis of this implementation was to develop a working prototype which illustrates the power of an SSCA. In particular it was shown that a fairly simple SSCA can create and manipulate novel connection structures in the network, thereby greatly decreasing the network resources required for video conferencing in an ATM environment. Since application specific knowledge is used to do this, it is impossible to do the same with a general purpose control architecture. These points are illustrated in the next section by comparing VideoMan with other possible video conferencing solutions.

D. Comparison

This section compares the VideoMan approach to more conventional video conferencing solutions which can be engineered by using a general purpose control architecture. The comparison is done on the basis of resource requirements, response time and scalability. The comparison in this section is of an architectural nature, and a discussion of related work is presented in Section V.

To simplify the comparison, it will be assumed that similar (or identical) floor control functionality is available in all cases. It is also assumed that a single high quality video stream of the current speaker needs to be displayed to each participant in a multi member video conference.

- At one end of the spectrum, connections for the video stream can be created on demand as dictated by the floor control function. A multicast connection is created from the current speaker to all other participants. When the speaker changes, a new multicast connection is created, and the previous one torn down. This approach is very efficient in terms of network resources, in that a single multicast connection is created. (If an old connection is only dismantled after the new one is created, there will often be more than one connection present in the network.) However, the time taken to create and dismantle the multicast connections will mean that changing between speakers will be unacceptably slow[14]. Since this problem will be exaggerated by larger networks and more participants,

[14] Commercially available control architectures report connection setup times of several milliseconds per call per switch [19]. Since current specifications [1], [20], [21] construct point-to-multipoint connections one endpoint at a time, the time taken to set up a multicast tree in a large network can be significant.

this solution does not scale very well.

• At the opposite end of the scale, a full mesh can be created between all participants, and the floor control function can simply inform participants on which connections to send or receive. In practice, a multicast connection will therefore be created from each member to all other members. Because a multicast connection is required for each participant, this solution uses a huge amount of network resources, (N multicast connections are required), and therefore does not scale well. Since all connections are left in place once they are created, the response time of this solution should be very good.

• One way of avoiding the problems outlined above is to make use of a special hardware unit which is often called a multipoint control unit (MCU) [13]. In this solution a unicast connection is created from each participant to the MCU and a single multicast connection is created from the MCU to all participants. Under direction of the floor control function, the MCU switches the appropriate unicast connection from the current speaker through to the multicast connection. The number of connections are reduced to one multicast and N unicast connections in the worst case, which means less network resources are required. (Worst case resource usage will occur in the switch directly connected to the MCU.) Switching between unicast connections should be very efficient, so good response times can be expected. Correct placement of the MCU can however be problematic, and might lead to network resources being tied down in a remote part of the network, i.e. removed from where the conference is taking place.

• The VideoMan approach is logically similar to the MCU solution presented above, however it requires less network resources and is more flexible: The core switch performs a similar function to that provided by the MCU, however no additional hardware is needed. Since the core switch is a logical entity in the sense that any switch can be the core switch, the problem of correct placement of the MCU is alleviated. If inverse multicasting can be used only two multicast connections (a gather and a distribution tree) are required, which significantly reduces the network resources required and therefore improve scalability. (If inverse multicasting cannot be used, the resources requirements are the same as that of the MCU approach.) To switch between speakers involves disabling one source and activating another and is therefore very efficient.

Although the above comparison rightly shows the similarity in functionality provided by the MCU approach and VideoMan, the two are in fact radically different. The MCU solution is probably the best that can be achieved using a general purpose monolithic control architecture. In contrast, VideoMan presents an idea of what is possible with an SSCA, which utilises service specific knowledge "inside" the network.

V. Related work

As indicated earlier, the approach adopted by standards bodies to deal with the diverse requirements of users and applications, is to add more signalling elements into a general purpose control architecture making it increasingly complex and ever evolving [1], [20], [21], [2]. The problems associated with its monolithic nature, lack of extensibility and poor support for multi media applications have been identified and addressed by many. In [22] an object based approach to signalling is proposed, whereby new services can be derived from a set of generic objects. In [23] a generic signalling protocol is presented, where seven basic connection services are used to construct more complex supplementary services. The use of a binding architecture based on a distributed processing environment is proposed in [24] to serve as a seamless control architecture between networking and multimedia devices. In [25] a control architecture, based on a distributed processing environment, was removed from the physical switching equipment and implemented on general purpose computing platforms. (See [8] for a more comprehensive discussion on control architectures.) The main emphasis of these different approaches has been to make *one* general purpose control architecture more open and extendible by various means. Although valuable in itself, this is different from the work presented in this paper, which propose the simultaneous use of *different* control architectures which cater for different requirements.

While the main purpose of the gather and distribution tree pairs introduced in Section IV was to illustrate the power of an SSCA, it is worthwhile comparing this with other approaches involving ATM. In UNI 3.1 [20], multicasting was limited to root initiated multicast trees. In [26], the UNI 3.1 facility was used to map IP multicasting onto ATM multicasting. The result was that either a multicast server was used, or a mesh of multicast trees were created. UNI 4.0 [21] extended multicast tree creation to also allow leaf initiated joins.

In [18], many-to-many ATM connections are proposed in a demand sharing environment: A many-to-many multicast tree is assumed to be created by appropriate extensions of the standard control architectures. An in-band cell level access protocol, called SMART, then grants access to the trees, merges trees and generally ensures that there is only one source active in any tree. The in-band access control protocol relies on the special treatment of resource management (RM) cells to exchange messages.

More recently, the use of core based trees [15], was proposed as a general multipoint-to-multipoint multicast architecture for ATM [16]. This approach, called SEAM, relies on the ability of some switches to do per-VC queueing and per-packet forwarding. The latter requires a switch to detect the first cell on a VC as well as the end-of-packet cell, and is used to prevent cell interleaving at merging points. Per-VC timers cater for the loss of end-of-packet cells. A store-and-forward mechanism of switching is proposed to prevent slow links from effectively locking outgoing ports for a particular VC during packet forwarding. The paper successfully illustrates the power of a single shared tree approach, and proposes a short-cutting mechanism which prevents unnecessary transmissions to the core switch.

The SMART and SEAM proposals achieve similar functionality to that of the VideoMan approach. Their proposals, aimed at a general purpose control architecture, try to provide a general multipoint-to-multipoint architecture. While the VideoMan architecture can benefit from a more generic approach, it is not clear that this can be achieved to the satisfaction of all applications. For example, in the case of the SMART approach, will a single access control mechanism (hardcoded in a general purpose control architecture) be sufficient for all requirements ? More importantly, these approaches also require modifications to the data transfer capability and/or in-band control mechanisms of ATM switches.

In the area of video conferencing numerous solutions have been proposed and implemented. Many of these have been on networks that do not provide QoS guarantees [27], [14]. Despite this fundamental drawback, and the associated problems [28], the inherent openness of this environment allows a significant amount of experimentation and innovation.

A multipoint conferencing system built around Integrated Services Digital Network

(ISDN), is presented in [13]. Single user workstations with a range of peripheral equipment provide the audio visual terminals (AVTs), and a hardware MCU connected to a node in the ISDN network allows multiple parties to partake in a single conference. Conferences are chaired and a distributed floor control function is used. Control data is carried in-band and is distributed to participants by the MCU.

In the ATM environment few solutions exist, mainly because of the lack of adequate support in standard control architectures. In [29], a sophisticated multimedia environment for the local area is presented. Because it was developed for the local area environment, network connections (and bandwidth) were not considered a scarce resource, and were not treated in any special way. While true for the local area, this is certainly not currently true for the wide area.

The systems described above provide more feature rich facilities than the VideoMan proof of concept implementation described in this paper. A full implementation based on the VideoMan concepts, will however provide a more scalable service which fully utilises the potential of an ATM environment. The VideoMan also proved that the switchlet approach provides an open ATM environment, in which experimentation and innovation is as easy as in the Internet environment.

While the current openness of the Internet can only be achieved in an ATM environment by means of systems such as the one described here, the Internet is considered too restrictive by some in that community. This led to several recent proposals which can be loosely grouped under “active networks” [30]. Active networks vary from an overlay, or glue layer, approach in [31], to capsules replacing packets in [32], and active packets in [33][15]. Some of the motivations for the active network approach parallel that of the work presented here, and are to some extent complimentary. However, their current work [30], [34] seems to depend on the ability to handle data transfer in software. While this is acceptable for experimental work, it is not clear how these approaches could be extended to networking elements where data transfer is primarily performed in hardware.

[15] The active packets in [33] are unfortunately also called switchlets, but have nothing in common with the switchlets in this paper and in [3].

VI. Conclusion

The paper has motivated the use of service specific control architectures in an ATM environment. Although the needs of many applications can be met by a general purpose control architecture, several example environments were presented in which this is not the case. The requirements of these environments can best be met when the control architecture can utilise service specific knowledge to manipulate connections within the network. By its very nature this is something that a general purpose control architecture is ill equipped to do.

The paper described the implementation of an environment in which virtual ATM networks can be dynamically created and modified, which allows for a very elegant way of providing SSCAs. Finally, an example implementation of an SSCA was presented. The VideoMan SSCA allows the creation of scalable video conferences by utilising the dynamics of user interaction in a conference to manipulate network resources and to manage communication.

A possible criticism of the concept of SSCAs in general, is that one ends up developing a great deal of mechanism which is used only in particular services when it might be of more general use. For example, the VideoMan resource management could be used in many multipoint communication environments. Such functionality might find its way into a general purpose control architecture, but it is not clear how such functionality would be developed in the first place without an environment such as the one presented here.

SSCAs should also not be seen as a limited set of controls for a pre-defined set of services, but rather as an enabling technology which will facilitate the development of new applications and services in a natural way. Indeed we believe that the dynamic virtual network environment, together with service specific control architectures are the enabling technologies, which will allow an openness in the ATM environment similar to that of the Internet, without compromising the attractive features of ATM.

References

[1] ATM Forum, *ATM User-Network Interface Specification - Version 3.0*, Prentice Hall, 1993.

[2] ATM Forum, "Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)," ATM Forum document: af-pnni-0055.000 , March 1996.

[3] J.E. van der Merwe and I.M. Leslie, "Switchlets and Dynamic Virtual ATM Networks," in *Integrated Network*

Management V, A. Lazar, R. Saracco, and R. Stadler, Eds. IFIP & IEEE, May 1997, pp. 355–368, Chapman & Hall.

[4] Burkhard Stiller, "A survey of UNI Signalling Systems and Protocols for ATM Networks," *ACM Computer Communications Review*, vol. 25, no. 2, pp. 21–33, April 1995.

[5] i-cubed, "SASCiA (the SCSI ATM switch): product overview," i-cubed Ltd., Rustat House, 62 Clifton Road, Cambridge, UK, 1996.

[6] Paul W. Jardetzky, *Network File Server Design For Continuous Media*, Ph.D. thesis, Cambridge University, Computer Laboratory, UK, 1992, Available as Technical Report No. 268.

[7] Abdi R. Modarressi and Ronald A. Skoog, "Signalling System No. 7: A Tutorial," *IEEE Communications Magazine*, vol. 28, no. 7, pp. 19–35, July 1990.

[8] J.E. van der Merwe, *Open Service Support for ATM*, Ph.D. thesis, Cambridge University, Computer Laboratory, UK, 1997.

[9] Peter Newman, Greg Minshall, Tom Lyon, and Larry Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 64–69, Jan 1997.

[10] APM, "The ANSA Model for Trading and Federation," Tech. Rep. AR.005.00, APM Limited, Castle Park, Cambridge, UK, 1993.

[11] Guangxing Li, "DIMMA Nucleus Design," Tech. Rep. APM.1551.00.05, APM Limited, Castle Park, Cambridge, UK, 1995.

[12] Richard Hayton, *OASIS An Open Architecture for Security Interworking Services*, Ph.D. thesis, University of Cambridge, Computer Laboratory, UK, 1996, Available as Technical Report No. 399.

[13] William J. Clark, "Multipoint multimedia conferencing," *IEEE Communications Magazine*, vol. 30, no. 5, pp. 44–50, May 1992.

[14] Hans Eriksson, "MBONE: The Multicast Backbone," *Communications of the ACM*, vol. 37, no. 8, pp. 54–60, August 1994.

[15] Tony Ballardie, Paul Francis, and Jon Crowcroft, "Core based trees (cbt), an architecture for scalable interdomain multicast routing," *Computer Communications Review (SIGCOMM 93)*, vol. 23, no. 4, pp. 85–95, 1993.

[16] M. Grossglauser and K.K. Ramakrishnan, "SEAM: A Scheme for Scalable and Efficient ATM Multipoint-to-Multipoint Communication," in *INFOCOM*. IEEE, April 1997.

[17] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, vol. 7, no. 5, pp. 8–18, Sept. 1993.

[18] Eric Gauthier, Jean-Yves Le Boudec, and Philippe Oechslin, "Shared Access to Many-to-Many ATM Connections," in *IEEE Global Telecommunications Conference*. IEEE Globecom, November 1996, pp. 2123–2127.

[19] FORE, "*ForeRunner* asx-200 atm switch users's manual," FORE Systems, Inc., 1000 FORE Drive, Warrendale, PA 15086-7502, USA., 1995.

[20] ATM Forum, *ATM User-Network Interface Specification - Version 3.1*, The ATM Forum, 1994.

[21] ATM Forum, *ATM User-Network Interface Specification - Version 4.0*, The ATM Forum, 1994, ATM Forum document: af-sig-0061.000.

[22] Steven Minzer, "A Signalling Protocol for Complex Multimedia Services," *IEEE Journal on Selected Areas in Communication*, vol. 9, no. 9, pp. 1383–1394, December 1991.

[23] Patrick A. Miller and Petre N. Turcu, "Generic Signalling Protocol: Switching, Networking, and Interworking," *IEEE Transactions on Communications*, vol. 40, no. 5, pp. 967–979, May 1992.

[24] A. A. Lazar, K. S. Lim, and F. Marconcini, "Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture," *IEEE Journal on Selected Areas in Communication*, vol. 14, no. 7, pp. 1214–1227, Sept 1996.

[25] S. Rooney, "An Innovative Control Architecture for ATM Networks," in *Integrated Network Management V*, A. Lazar, R. Saracco, and R. Stadler, Eds. IFIP & IEEE, May 1997, pp. 369–380, Chapman & Hall.

[26] G.J. Armitage, "Multicast and Multiprotocol support for ATM based Internets," *Computer Communications Review*, vol. 25, no. 2, pp. 34–46, April 1995.

[27] Stephen Casner and Stephen Deering, "First IETF Internet Audiocast," *Computer Communications Review*, vol. 22, no. 3, pp. 1–6, July 1992.

[28] S. Keshav, "Experience with Large Videoconferences on Xunet II," in *INET'94/JENC5*, 1994.

[29] Stuart Wray, Tim Gluart, and Andy Hopper, "Networked Multimedia: The Medusa Environment," *IEEE Multimedia*, pp. 54–63, 1994, Winter edition.

[30] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, January 1997.

[31] Yechiam Yemini and Sushil da Silva, "Towards Programmable Networks," *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1996.

[32] David L. Tennenhouse and David J. Wetherall, "Towards an Active Network Architecture," *Computer Communications Review*, vol. 26, no. 2, pp. 5–18, April 1996.

[33] J.M. Smith, D.J. Farber, C.A. Gunter, S.M. Nettles, D.C. Feldmeier, and W.D. Sincoskie, "*SwitchWare*: Accelerating network evolution," Tech. Rep. MS-CIS-96-38, CIS Dept. University of Pennsylvania, 1996.

[34] D. Scott Alexander, Marianne Shaw, Scott M. Nettles, and Jonathan M. Smith, "Active bridging," *Computer Communications Review (SIGCOMM 97)*, vol. 27, no. 4, pp. 101–111, October 1997.

Kobus van der Merwe received the B.Eng. and M.Eng. degrees from the University of Pretoria in 1989 and 1991 respectively, and is currently working towards the Ph.D. degree at the University of Cambridge Computer Laboratory, Cambridge, U.K. His current research interests are in network control and management, network resource management, and the Internet.

Ian Leslie received the B.A.Sc in 1977 and M.A.Sc in 1979 both from the University of Toronto and a Ph.D. from the University of Cambridge Computer Laboratory in 1983. Since then he has been a University Lecturer at the Cambridge Computer Laboratory. His current research interests are in ATM network performance, network control, and operating systems which support guarantees to applications.

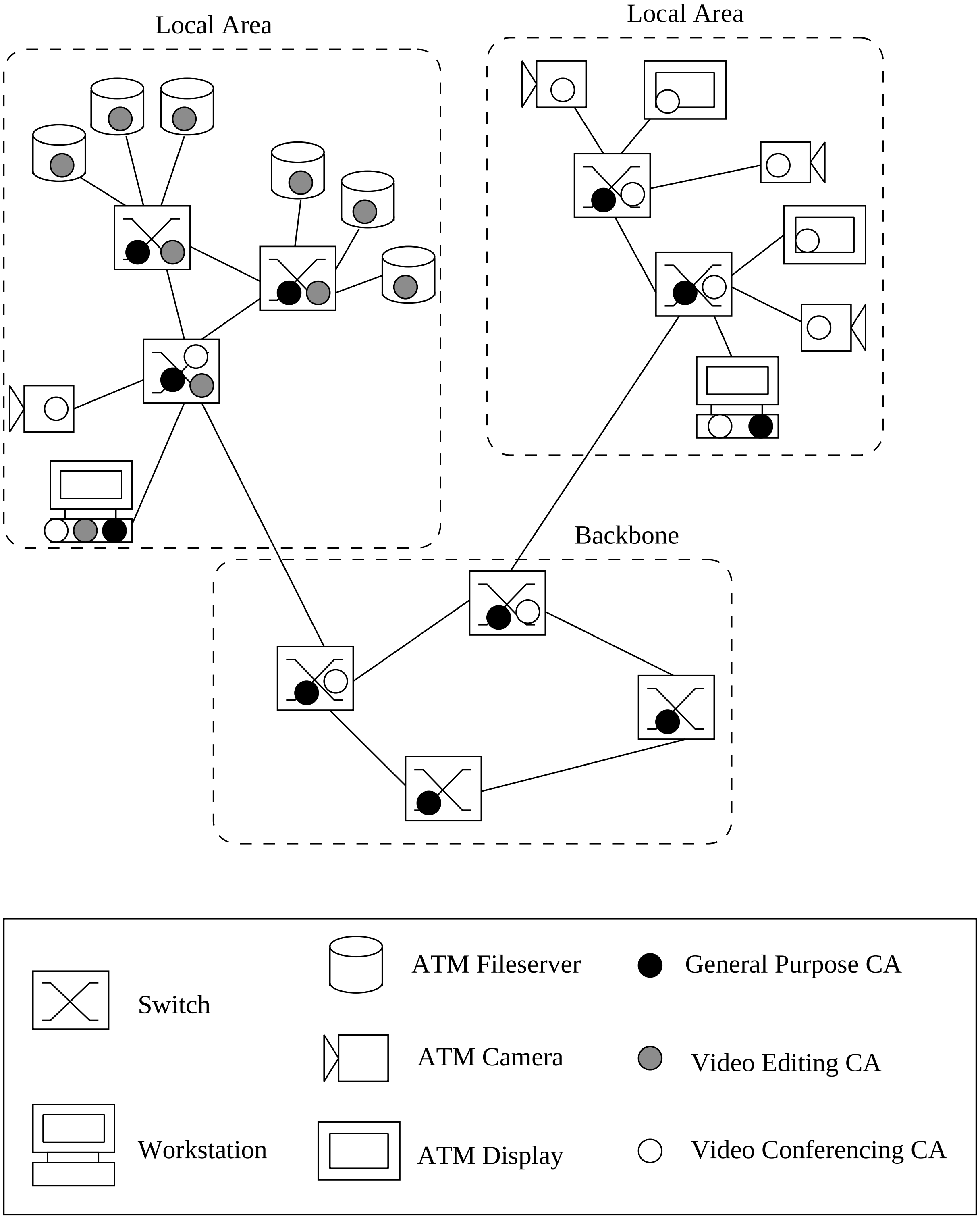

Fig. 1. Control Architectures in a Physical Network

Manuscript received 10 January 1997; revised 28 August 1997

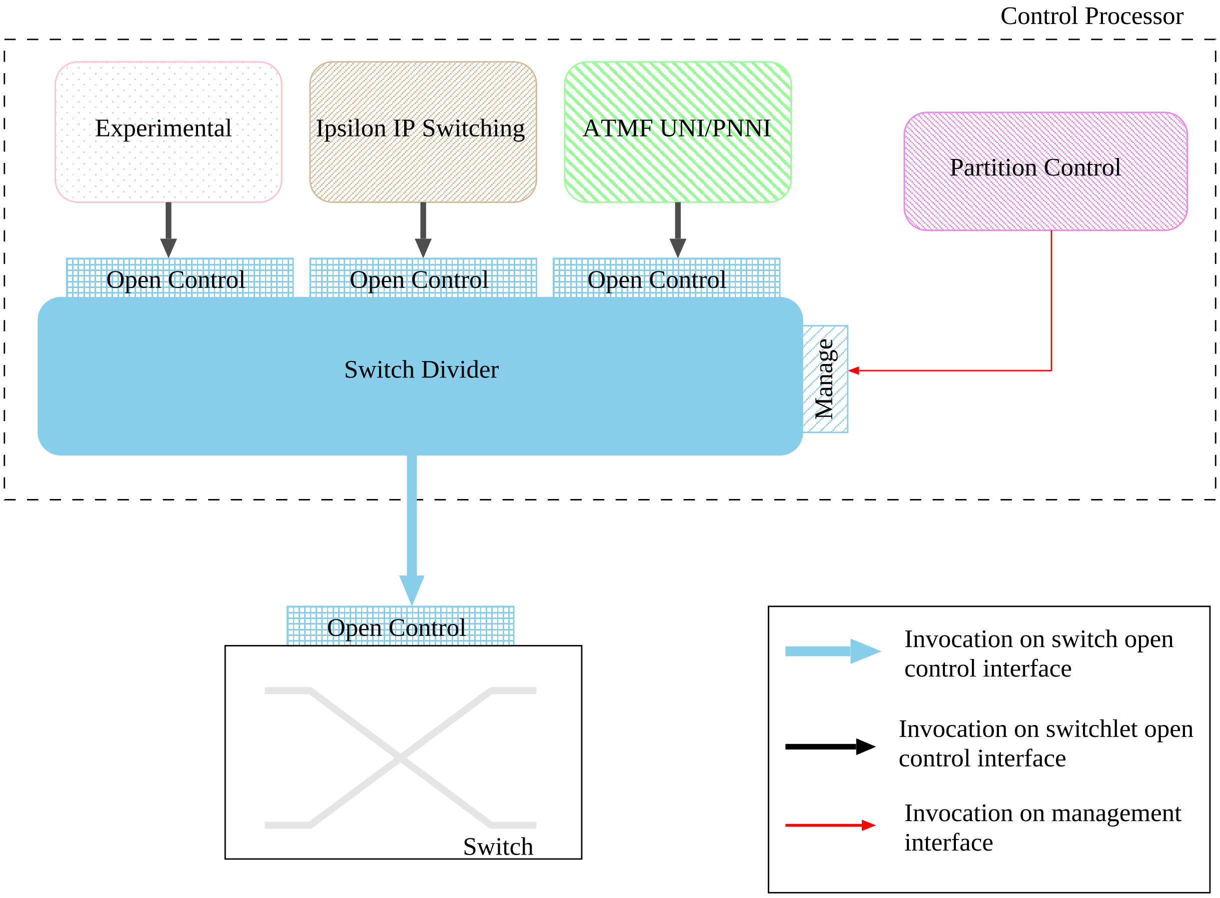

Fig. 2. Creating switchlets

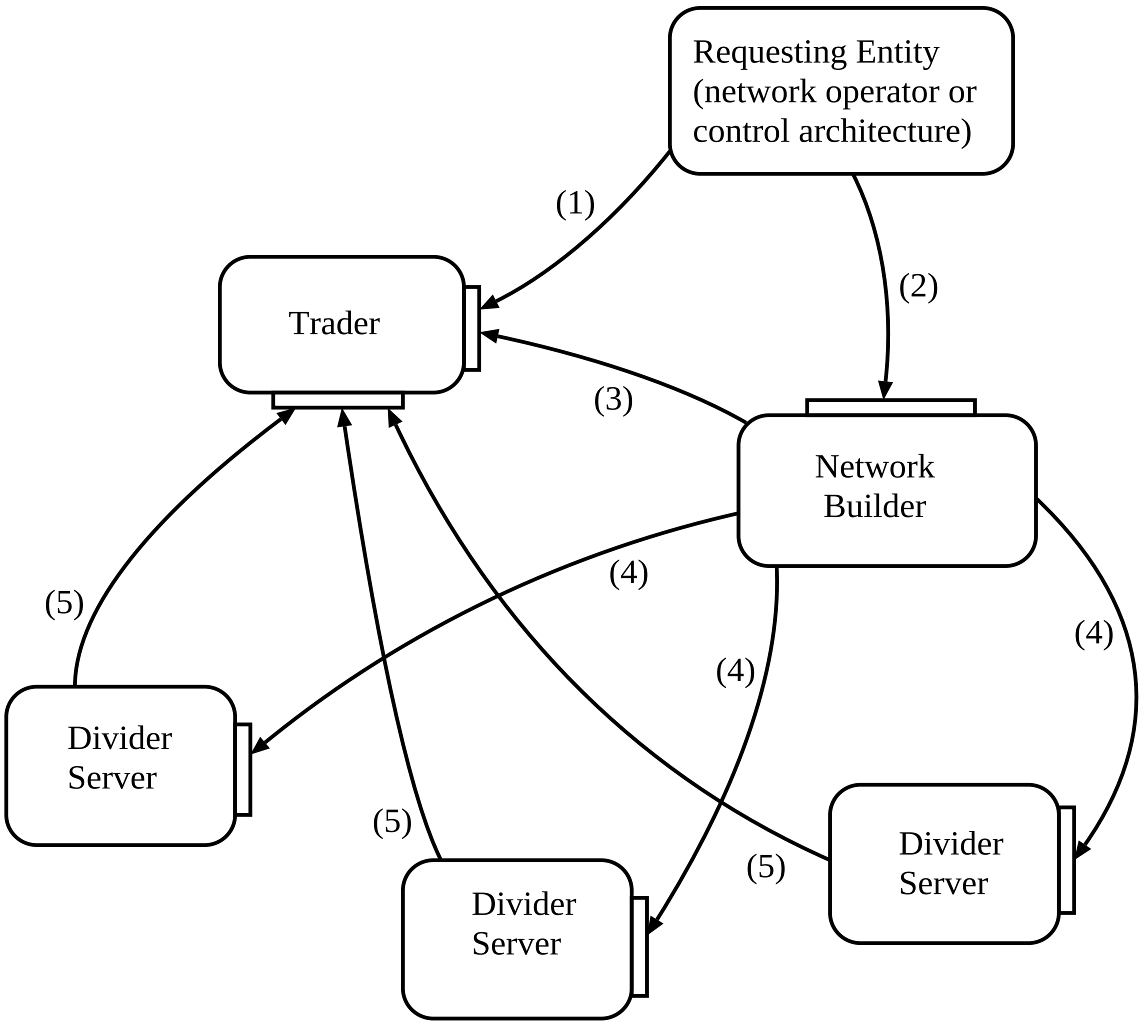

Fig. 3. Creating a Virtual Network

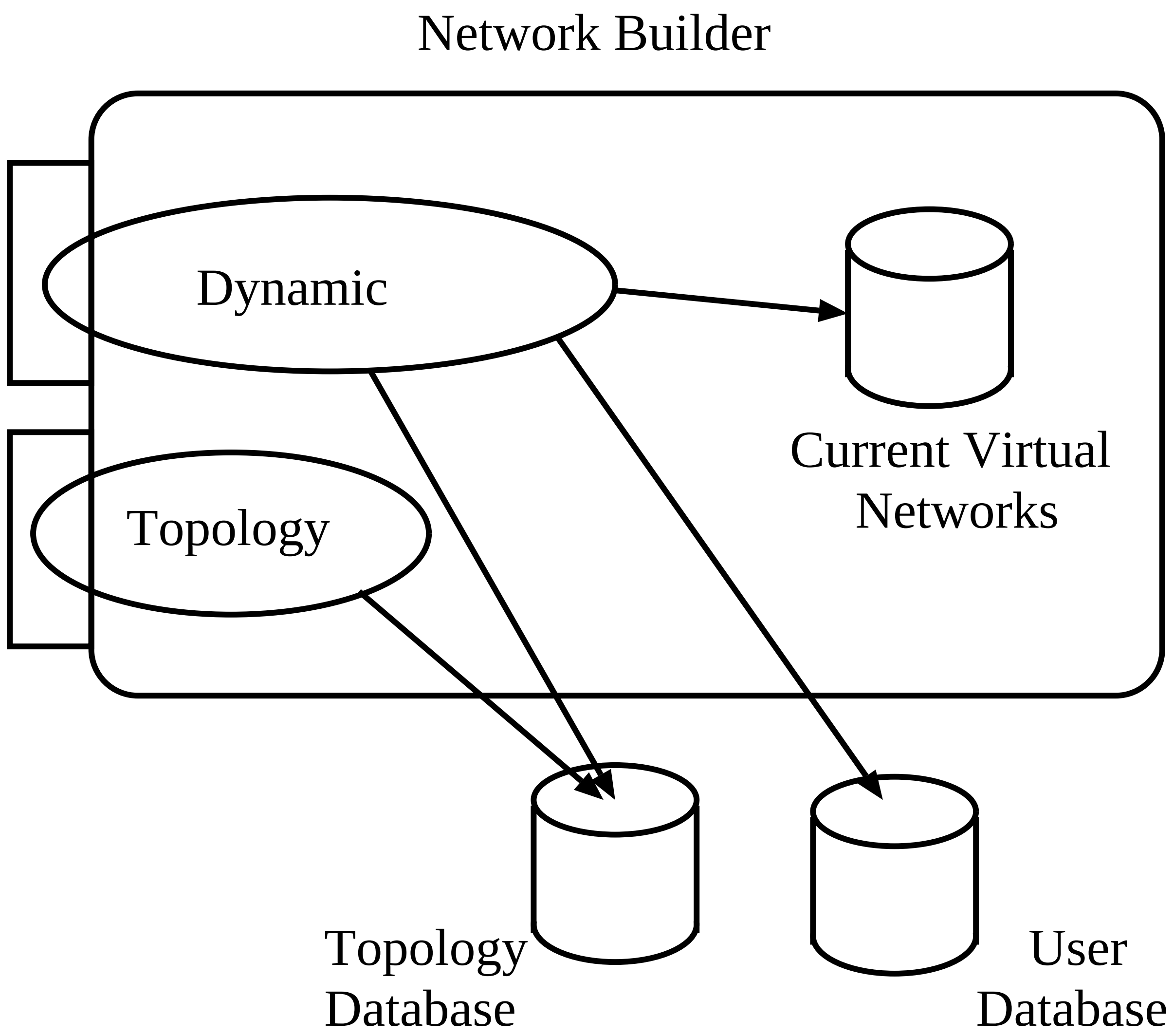

Fig. 4. Network builder implementation

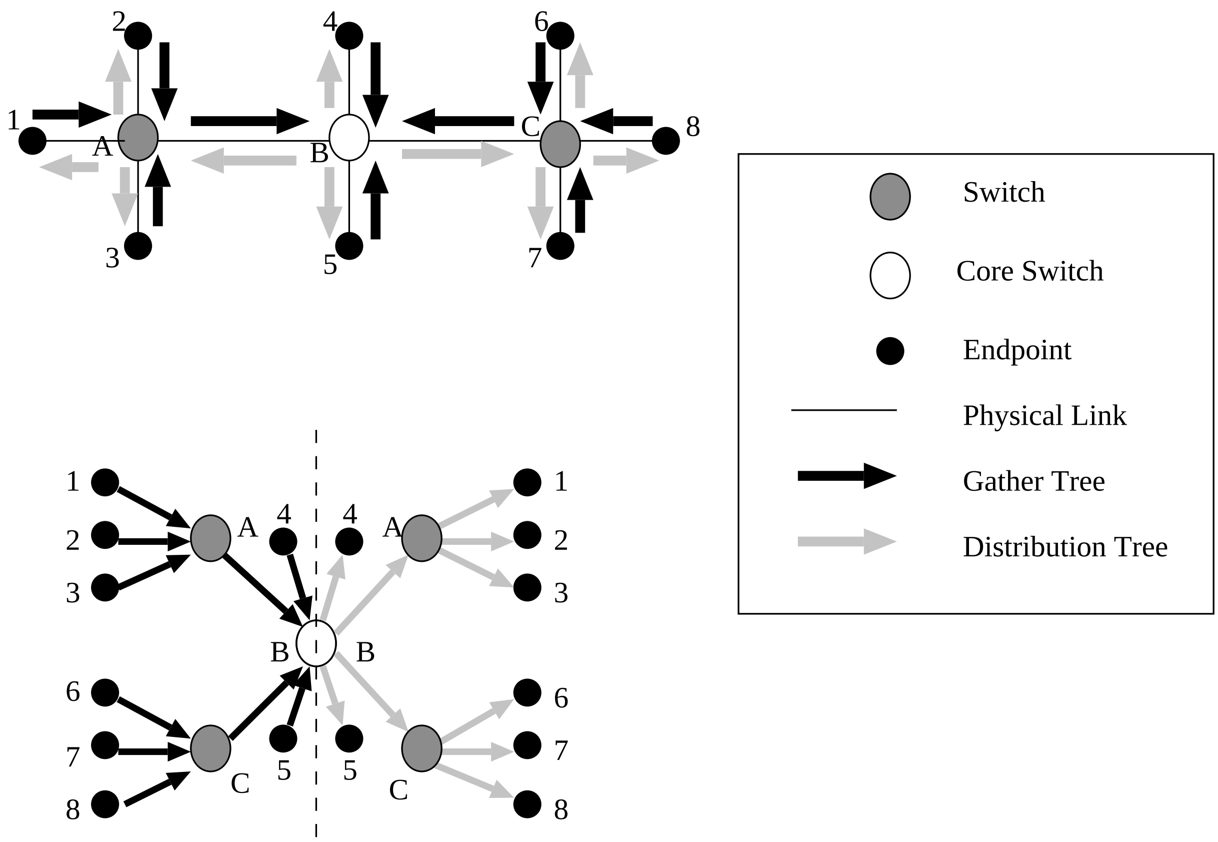

Fig. 5. Gather and Distribution trees

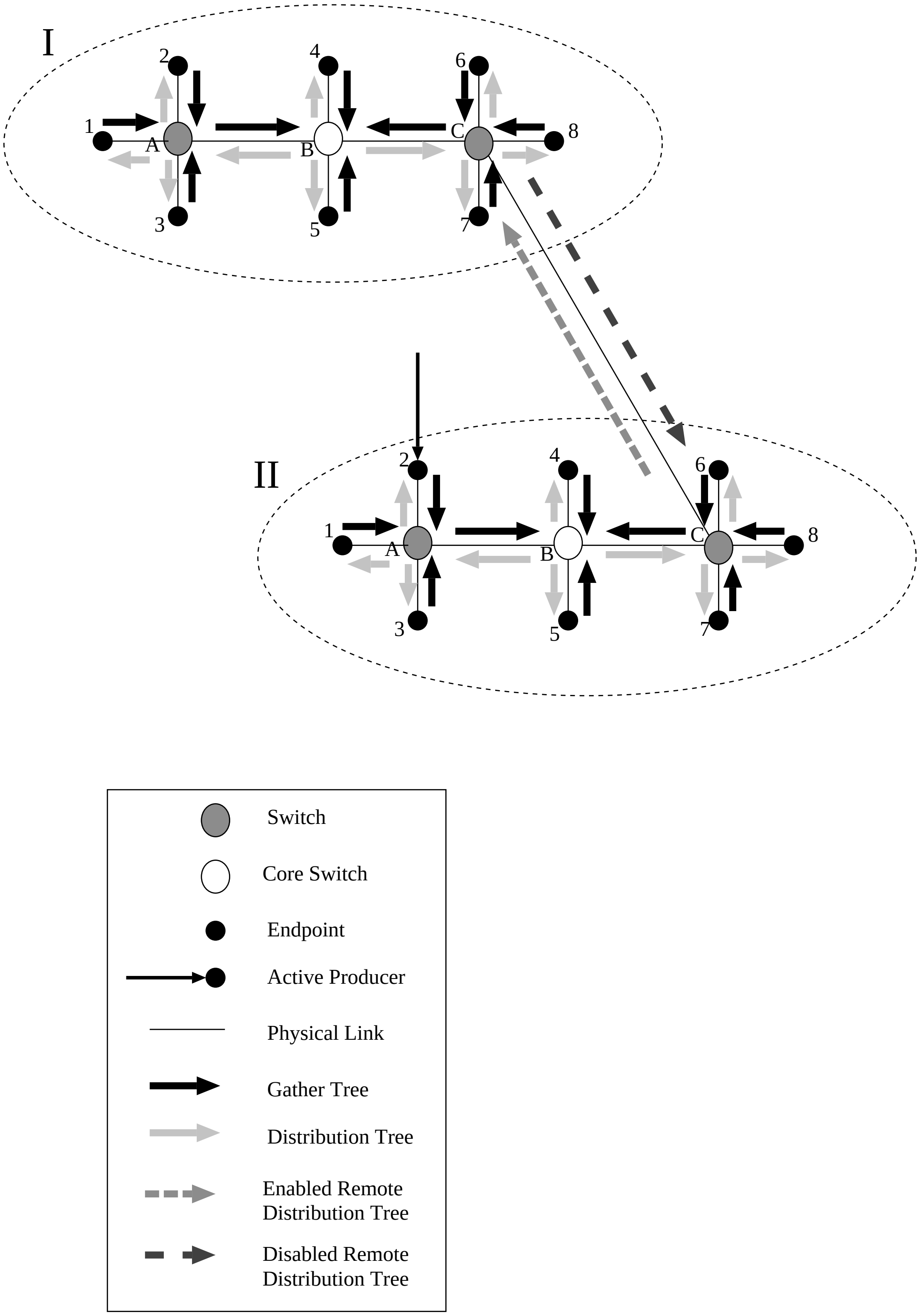

Fig. 6. Inter site distribution

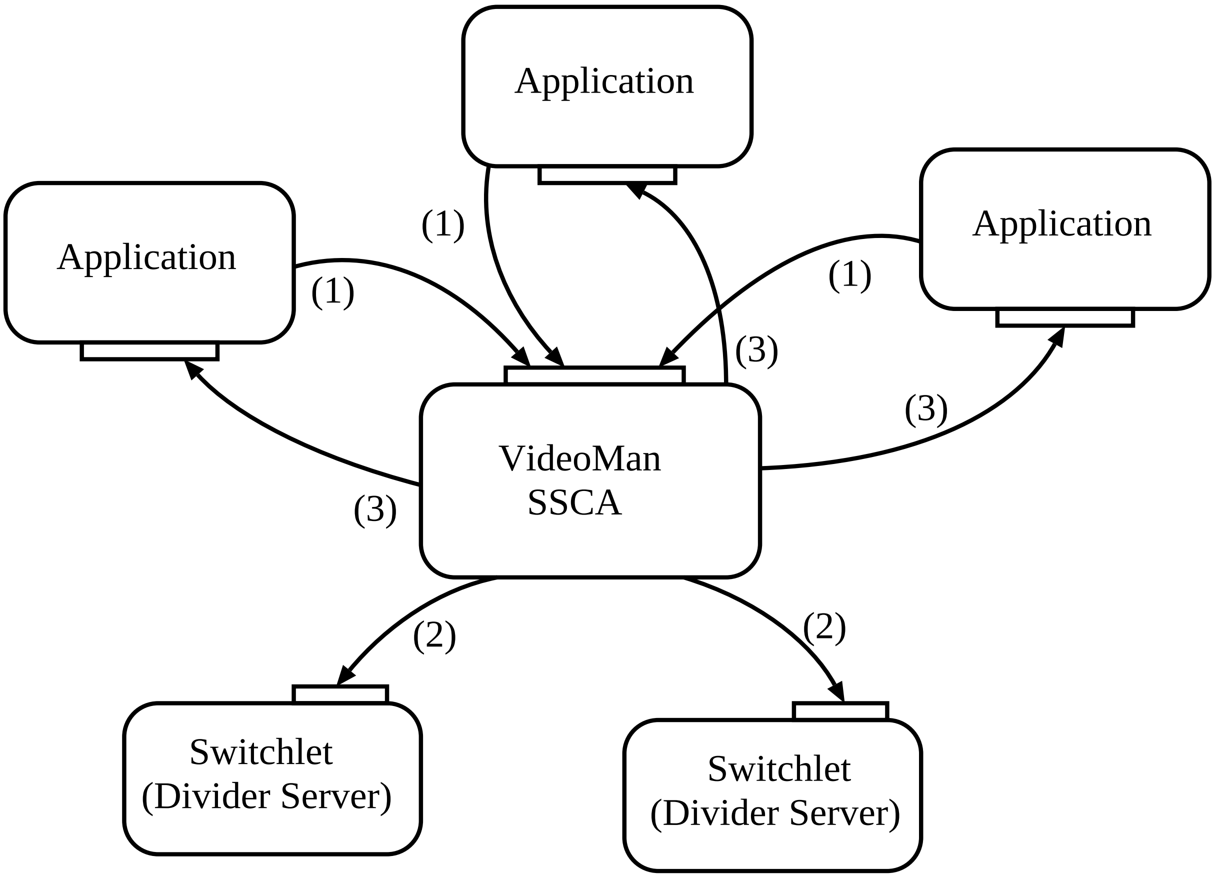

Fig. 7. Conferencing Control Architecture