The Tempest — A Practical Framework for Network Programmability

Jacobus E. van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby
University of Cambridge

Abstract

The Tempest framework provides a programmable network environment by allowing the dynamic introduction and modification of network services at two levels of granularity. First, the switchlet and associated virtual network concepts enable the safe introduction of alternative control architectures into an operational network. The timescales over which such new control architectures can be introduced might vary from, for example, a video conferencing specific control architecture, which is active only for the duration of the conference, to a new version of a general purpose control architecture, which might be active for several months or longer. Second, the Tempest framework allows refinement of services at a finer level of granularity by means of the connection closure concept. In this case modification of services can be performed at an application-specific level. These attributes of the Tempest framework allows service providers to effectively become network operators for some well defined partition of the physical network. This enables them to take advantage of the knowledge they possess about how the network resources are to be used, by programming their own specially tailored control architecture. This, as our work with the Tempest shows, is a spur to creativity allowing many of the constraints imposed on operators and end users to be rethought and for new techniques to be quickly and safely introduced into working networks.

The need for a single network supporting a large number of diverse services is one of the major factors driving networking research. There are two important groups of functions to consider when building a multiservice network:

- The first concerns the different requirements of services in terms of data forwarding in the network. These functions include the scheduling, policing, and shaping mechanisms used within a node to forward packets and is often called *in-band control.*
- The second involves the structures involved in controlling the appropriate forwarding of data. These functions include the exchange of routing information, resource reservation and forwarding policies and is often called *out-of-band control.*

This article is concerned with the second set of functions in a multiservice network, which we collectively call a *network control architecture*. We believe that the control plane is the appropriate place to introduce flexibility and programmability in a network and present *The Tempest* as a practical framework to achieve this [1, 2].

The Tempest depends on our ability to simultaneously control a given ATM switch with multiple controllers by partitioning the resources of that switch between those controllers. We call such a partition a *switchlet* as to all intents and purposes the switchlet appears as a small, but complete, switch to the controller.[1] The set of switchlets that a controller or group of controllers possess forms its *virtual network*. The Tempest allows us to run both standard and non-standard control architectures on the same network. Third parties may lease a virtual network from the Tempest network operator and control it in the way best suited to their needs. The Tempest allows new control architectures to be introduced dynamically into the network and therefore provides network programmability at the granularity of control architectures. Our work [3] has identified the advantages of *service-specific* control architectures dedicated to one type of high-level service, e.g., video conferencing. The Tempest enables network programmability at an even finer level of granularity by allowing end users to dynamically associate code with network resources in a running control architecture in order to achieve *application-specific* control [4]. We call the association of the user defined control policy and the allocated network resources a *connection closure*.

In summary, the Tempest work is premised on the fact that there will be a requirement for many different control architectures on a widely deployed multiservice network. It demonstrates how this can be elegantly achieved through the

[1] *Switchlets is protected under British Patent Application number 9621248.5.*

switchlet concept. Further, it shows how end users can take advantage of switchlets to program their network either by writing dedicated service specific control architectures or by the use of control architectures which are capable of integrating application specific code dynamically. This article describes all components of the Tempest framework, all of which have been implemented at the University of Cambridge Computer Laboratory in various proof-of-concept implementations.

The Tempest

Motivation

The distinction between in-band and out-of-band control is inherent in ATM technology. Despite this fact, current commercial ATM switches are implemented as an integrated unit whereby both the packet forwarding and connection establishment functions are sold as one complete package. Furthermore, while both levels of control in ATM are subject to standardization, the interaction between the control architecture and the switching plane is not specified, allowing switch vendors to implement proprietary solutions.

Open signaling has proposed to formalize the separation between the switch controller and the switch fabric using a switch independent interface, thereby allowing both levels of control to be developed and to evolve independently. It is envisaged that this will allow more flexible ATM control and faster introduction of new services.

Our early work on open signaling led us to the realization that some of the basic assumptions regarding switch control could be rethought. By partitioning the resources of a switch it is possible to allow several switch controllers to manage distinct partitions of the switch simultaneously and in consequence to run multiple control architectures over the same physical network. This is shown in Fig. 1. We call the basic infrastructure for achieving this *The Tempest*. The Tempest allows us to define control policies not only at the packet and connection level, but also for individual virtual networks.

The Tempest frees network operators from having to define one single unified control architecture capable of satisfying the control requirements of all possible future services. It also allows operators the freedom to differentiate their service offerings in a proprietary fashion, while maintaining a base level of interoperability with other operators where this is required. At the same time it allows "end users" to become network operators, allowing them to make decisions not only about the resources allocated to their connections, but also the way that those connections themselves are established and managed. The Tempest changes the question from *How should I implement my service with this control architecture?* to *What is the best control architecture for my service?*

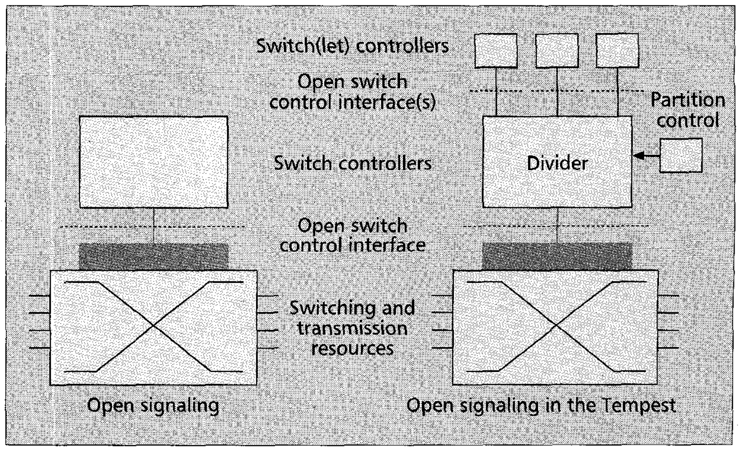
The rest of this section describes the core entities that make up the Tempest framework. The Tempest consists of:

- The Ariel switch-independent control interface
- The Prospero switch divider
- The Caliban switch management interface
- The Bootstrap Virtual Network
- The Network Builder
- The Hollowman Control Architecture

Each of the core Tempest entities are now described in turn.

The Ariel Switch-Independent Control Interface

The *Ariel* switch-independent control interface is the means by which control architectures communicate with the switch.


Switch(let) controllers
Open switch control interface(s)
Partition control
Switch controllers
Divider
Open switch control interface
Switching and transmission resources
Open signaling
Open signaling in the Tempest

■ **Figure 1**. *Open signaling and open signaling in the Tempest.*

Ariel is as low-level as is consistent with it being independent of any given switch. Ariel assumes a simple client-server interaction between the control architecture and a server on the switch. A control architecture communicates with the Ariel server using a convenient transport, e.g., a well defined message passing protocol or a remote procedure call (RPC) mechanism, and the Ariel server translates the Ariel control operations into a form that can be understood by the physical switch. As such the Ariel control interface is not a protocol specification but can be considered an Abstract Data Type which specifies the *functionality* required by the interface rather than the way it should be implemented.

The operations in the basic Ariel interface are divided into the six groups explained below:

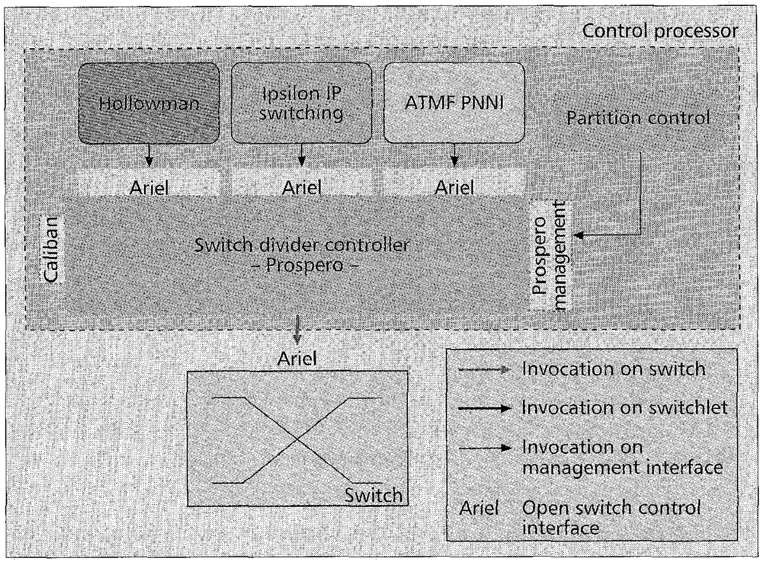
Configure Operations — The configuration operations allow an Ariel client to determine the switch configuration. The minimum amount of information the Ariel server should be capable of returning is: the type of the switch; the number of ports; the VPI/VCI range for each port; the ATM service categories that each port can support, e.g., ABR, VBR, etc.

Port Operations — The port operations allow a client to examine and change the administrative state of a port. For example, the state of a port can be toggled between active and inactive, or between normal and loop-back mode.

Resource Context Operations — The context operations allow a client to build resource contexts which can be associated with connections during connection creation. The parameters required by this interface follow those defined by the ATM forum in [5] and therefore should be supported by most commercial switches. This abstraction avoids having to know information about the precise implementation of the fabric, e.g., queuing algorithms, which are unlikely to be made public for commercial switches.

Statistics Operations — The statistics operations allow a client to examine information about virtual channels, virtual paths, and ports. These include the number of received and transmitted cells, the number of erroneous cells, the number of dropped cells, and — for virtual paths and virtual channels — the time elapsed since creation.

Alarm Operations — The alarms operations allow clients to register an interest in various exceptional conditions that the switch might generate, for example, changes in the state of ports.



■ Figure 2. *Creating switchlets.*

Connections Operations — The connections operations allow clients to create and delete virtual paths and virtual channels and to determine which virtual paths and channels are in place at any given moment. The resources associated with a connection are allocated by first creating a context by means of appropriate resource context operations. This context is then passed as a parameter to the connection operation when the connection is created. The separation between resource context and connection establishment, has the desirable side effect that best-effort connections can be established without the overhead of first creating a resource context.

The Prospero Switch Divider

Ariel is a switch control interface for open signaling. This basic functionality has been extended by a crucial component of the Tempest framework, namely the Prospero Switch Divider [1]. Prospero communicates with the Ariel server on the switch and as such is solely in control of the switch as a whole. It partitions the switch resources into subsets, each of which is called a switchlet. The Prospero Divider exports an Ariel control interface for each switchlet created in this fashion. Figure 2 depicts this arrangement.

Switchlet resources include a subset of the switch ports, VPI/VCI space, bandwidth, and buffer space. More than one switchlet can have the same switch port, but all switchlets need not have the same set of ports. Switch control software, of a particular type, controls its switchlet by invocations on the switchlet Ariel interface, in exactly the same way as it would control a physical switch with just a single Ariel interface. As an example, Fig. 2 shows three possible controllers, namely, for the Hollowman control architecture [2], and the IP Switching [6] and ATM Forum's UNI/PNNI [7, 8] control architectures. Partitioning of the switch into switchlets is controlled through a separate Prospero Management interface on the Divider Controller. Prospero also contains the Caliban server for general switch management in the Tempest framework, as discussed below.

The control operations performed by different controllers on their respective switchlet Ariel interfaces are intercepted by Prospero. The Divider ensures that the resources which an operation is trying to access belong to the client. If they do not, then the switch divider refuses the operation; otherwise, the switch divider forwards the operation to the Ariel server on the physical switch.

The Caliban Switch Management Interface

Maintaining the "health" of the network includes functions such as fault management and performance monitoring. A very successful and widely deployed mechanism employed in this context is the SNMP network management protocol [9]. Currently most, if not all, commercial LAN ATM switches support SNMP. SNMP's lack of fine-grained access control means that it is not suitable for the management of a Tempest virtual network. For example, many commercial switches allow connections to be created and modified by means of the appropriate SNMP operations on the switch. A control architecture could subvert the partitioning imposed by the Tempest if no restrictions are placed on the SNMP operations it can perform. This could be problematic in an environment where the Tempest is used to supply virtual network services to many distinct non-cooperating companies, each of which wants to perform its own network management.

The chosen solution in the Tempest framework is to define an interface containing a small set of primitives which can be mapped onto a variety of underlying management protocols. This interface is called *Caliban.* Caliban is a switch management interface which allows a network administrator to verify and ensure the overall health of their virtual network within the Tempest environment. Control architectures communicate with a Caliban server using this interface across some transport, and the server translates the request into the appropriate format for communicating with the switch. Control architectures do not address the switch directly, therefore fine-grained access control can be implemented within the Caliban server if the switch itself cannot support it. Within the Tempest, the Prospero switch divider is already required to know the resource allocation of each control architecture, so it is natural to run the Caliban server as a part of the switch divider as depicted in Fig. 2.

Bootstrapping

The switchlets created by invocations on the management interface of the Prospero divider naturally combine to form virtual networks. The creation of switchlets on a network of switches must be performed in such a fashion that the resulting virtual networks are fully connected over the resulting topology. For example, care must be taken that the VC-address space of two adjacent switchlets in the same virtual network overlap. While this function can be performed manually by a network administrator, it is best automated. The *Network Builder* considered in the next section is such a Tempest service which is used to create, modify and release virtual networks.

Regardless of the exact way in which virtual networks are created, remote access to and interaction with the Management interfaces on the Prospero dividers is required. Since the Tempest framework defines and provides the means for network interaction, this presents a bootstrapping problem: *How are Tempest components to interact, for example, to create virtual networks, if the Tempest itself provides the means for communication?*

The easiest way to solve this is by designating a default switchlet in each physical switch to be part of a *Bootstrap Virtual Network.* This bootstrap virtual network implements a

Bootstrap Control Architecture, the combination of which provides the platform for the required initial communication infrastructure.

The Bootstrap Virtual Network — The bootstrap virtual network and control architecture have to be "switchlet aware," in that they must start up using a default switchlet while at the same time allowing switchlet allocation for other virtual networks to be performed. Although it is possible to design and build a special control architecture to fulfill these requirements, a more general purpose control architecture will suffice. In our proof-of-concept implementation an existing IP-over-ATM network was used as bootstrap virtual network. The use of a ubiquitous protocol such as IP in the bootstrap virtual network has proved very useful. Note that this does not necessarily imply the use of the Internet Engineering Task Force (IETF) IP-over-ATM mechanism [10], which in turn requires the fairly heavyweight ATM Forum control architecture [7, 8]. For example, more lightweight control architectures providing an IP service, such as IP Switching [6], might be more appropriate.

Distributed Processing Environment — One generic service provided in the bootstrap virtual network is a Distributed Processing Environment (DPE). Indeed most of the remaining Tempest components have been implemented using a DPE and assume its existence in the bootstrap virtual network. A Distributed Processing Environment (DPE) is the framework in which distributed entities cooperate in order to achieve some objective [11]. The DPE used in the described version of the Tempest framework is Dimma [12], which is an experimental CORBA-Orb implementation.

One of the common services that a DPE can provide is a *Trader* function. Trading is the means by which clients in a DPE find out about services and the properties of those services. Within this article, we define an *Interface Reference* to be an entity containing the information required by a client to communicate with a server. An interface reference typically contains information about the service type and the possible transports that can be used to communicate with the emitting server.

The Network Builder

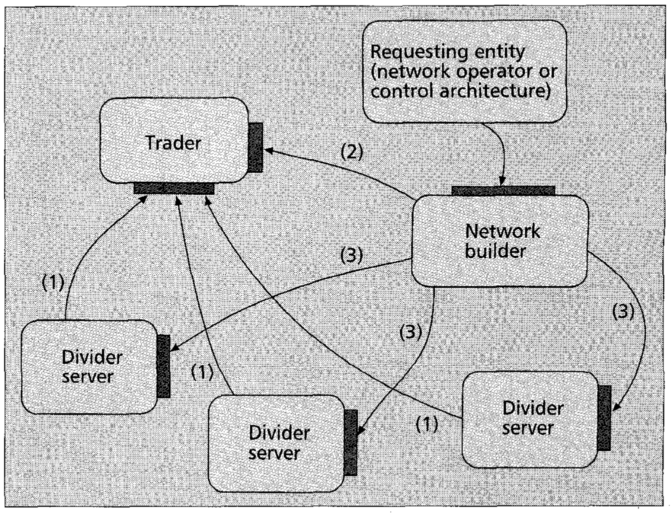
The *Network Builder* is a Tempest service involved with the creation, modification, and maintenance of virtual networks. In order to create a virtual network, the network builder service is provided with the "specification" of the desired network. The network is specified in terms that hitherto could only be contemplated during the network design phase, and could only with great difficulty be changed after network equipment had been commissioned. An example network specification could be:

- ATM Forum control architecture
- CBR capacity of 15 Mb/s
- Between A and B
- With redundancy
- For three hours

Or it could be as simple as a request to create "A cheap network between A and B."

The network specification could be the output of another service, for example a control architecture, or be provided by a human being. Figure 3 shows the interaction between the services that are involved in the process of requesting the network builder to create a virtual network.

With reference to Fig. 3 the creation of a virtual network can be summarized with the following steps:



■ Figure 3. *Dynamic virtual network services.*

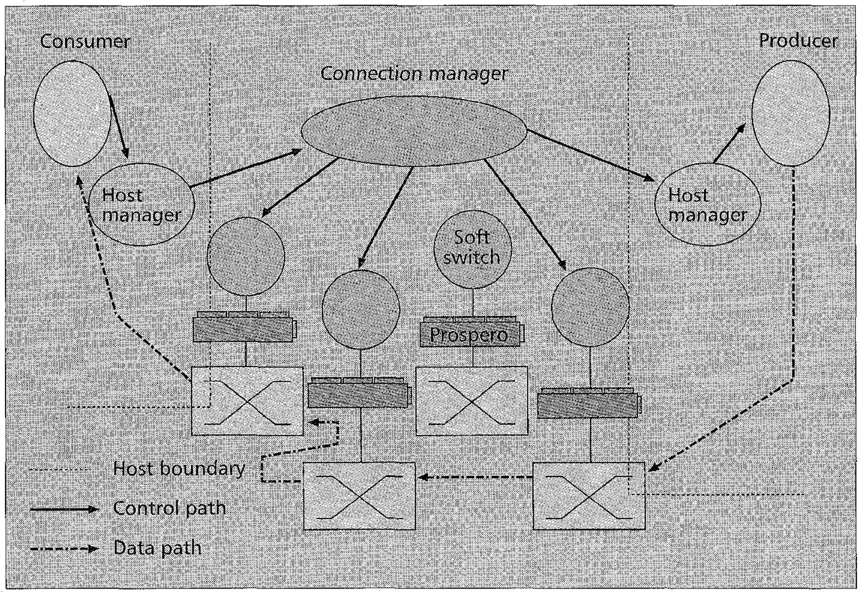
- At startup the Divider Servers, each of which is associated with a physical switch, registers with the Trader its configuration and capacity. This process is repeated whenever any of these capabilities change.
- The Network Builder, on receipt of a virtual network specification, consults the Trader to obtain interface references to all Divider Servers which satisfy the criteria of the request.
- The Network Builder invokes appropriate methods on the Divider Server(s) to create the switchlets and resulting virtual network.

Two possibilities exist in terms of the type of the control architecture which will be instantiated on a newly created virtual network:

- A predefined control architecture from a well known set can be started up when the virtual network is created. An example would be the creation of an ATM Forum UNI/NNI compliant virtual network. In DPE terminology this would be called a *traded typed* virtual network. In this case the appropriate software entities will be started up by the Network Builder as soon as the virtual network has been created.
- Alternatively, it is possible to create a virtual network without a control architecture. In this case the control architecture is supplied or "filled in" by the entity that requested its creation. This would be called an *anytype* virtual network in DPE terms. In this case the Network Builder will not start up the control architecture, but rather will return an interface reference for each switchlet to the entity that requested creation of the virtual network. This entity may itself be the control architecture which will exert control on the newly created virtual network. Since interaction with the switchlet interface involves the services of the DPE, these control architectures will normally be required to be DPE-aware.

In this case the control architecture either builds its network by specifying the amount of resources it requires and the switches on which it requires them (a process which can be done in a piecemeal fashion), or it is allocated a predefined virtual network. The network builder, after allocating a virtual network, informs the control architecture about the resources that have been allocated to it and passes it the Ariel interface references through which they can be accessed.

In all cases, when a control architecture terminates, it informs the network builder that liberates its virtual network. The resources which comprise that virtual network, then become available for use by other control architectures.



■ Figure 4. *Hollowman control path.*

The Hollowman Control Architecture

The Hollowman control architecture serves three distinct purposes:

- It is an example of Tempest control architecture
- It serves as a flexible platform on which experiments related to ATM control can be carried out
- It acts as support for a number of applications that require advanced control. Those applications in turn have served as a means of testing the Hollowman.

In the longer term, the Hollowman will become a set of basic components with which network operators can build their own control architectures within the Tempest environment. The Hollowman implements the full set of UNI 4.0 mandatory capabilities, e.g., point-to-point connections, and optional capabilities, e.g., multicast and proxy signaling.

The core Hollowman functions enable the creation and deletion of ATM connections between ATM-capable workstations and devices. The Hollowman allows applications a great deal more flexibility than standard-based signaling systems in the management of their resources, while the Application Programmers Interface (API) is minimal and simple to use. The Hollowman is a realistic ATM control architecture in terms of both the functions that it offers and the efficiency with which it performs those functions; the desire to keep the Hollowman simple was an overriding concern in its design.

The Hollowman contains a small set of entities, these are:

- The soft switch — the entity through which the rest of the Hollowman interacts with its switchlet
- The host manager — the entity which manages the resources of a host
- The connection manager — the entity which synchronizes the creation, modification and deletion of connections
- Traders — the means by which applications learn of services

Applications use traders to obtain offers for services that have been advertised by service providers. Having obtained an offer, the application communicates with the host-manager running locally on its host through a simple API to establish a connection. The host-manager, after some local book keeping, forwards the request to the connection-manager. The connection-manager determines the location of the service provider and calculates a route between the provider and demander. The connection-manager synchronizes the actions of the host-managers and a set of soft switches in order to establish the end-to-end connection. Figure 4 shows the entities involved in the creation of a Hollowman connection. In the current implementation there is one connection manager per Hollowman domain, this is purely for reasons of convenience and is not a constraint of the model.

Experimental Results

All components of the Tempest framework described in this article have been realized in a proof-of-concept implementation on a variety of ATM switches, workstations, and other ATM end devices at the Computer Laboratory. More details of the various implementation efforts can be found in [13, 14].

A variety of Ariel switch control interfaces were implemented based on both RPC mechanisms and well defined messages. Only a subset of the operations presented earlier were implemented to allow comparison between different implementations. The time taken to perform a connection setup was measured to be between 2.3 ms and 4.5 ms depending on the implementation. The cost of having the Prospero divider in the control path was evaluated by having Prospero use the most efficient Ariel implementation to communicate with the switch, and then measuring the connection setup time through a switchlet Ariel interface. Having Prospero in the control path increased the average connection setup time to approximately 5.2 ms in the worst case. These figures are comparable with numbers published elsewhere [6].

Creating a switchlet in the current implementation has the side effect of "cleaning up" the VC-space so that it can be presented to a control architecture in a known state. That is to say, all existing connections falling in the range allocated to a switchlet are released. This operation completely dominates the time taken to create a switchlet in the current implementation and was measured to be in the order of 40 ms for a switchlet with seven ports and 1 VP with 40 VCs per port.

The measurements above were taken on an HP 9000 series workstation with the Ariel server implemented on an ASX-100 ATM switch from FORE Systems.

Experiments have been carried out using the Hollowman in regard to parallelizing the connection establishment over a number of switches in order to reduce the signaling latency and "caching" certain connections in the expectation that they will be reused, also with the objective of reducing latency. [15, 16] and more recently [17] have proposed similar techniques. Even without these optimizing techniques the Hollowman compares quite favorably in terms of performance with more conventional signaling systems [18, 19], for example, point-to-point connection creation over a single switch with the Hollowman running on Sun Ultras under Solaris 2.5 requires approximately 11 ms. This is equivalent to the best published figures for standards based implementations [19].

The results of our experiments indicate that the Tempest approach does not compromise efficiency while greatly enhancing the flexibility of network control.

Using the Tempest

Being able to deploy multiple instances of the same or different control architectures in the same physical network is the most obvious use of the Tempest framework. The topology of the resulting virtual networks can be dynamically modified and indeed resources can be moved between virtual networks depending on demand, which makes the Tempest framework much more flexible than any other virtual network offering. In addition to these uses the Tempest framework enables more advanced uses, a number of which are outlined below.

Service-Specific Control Architectures

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 [7]. As the requirements of users and applications change, modifications are made to the control architecture which are reflected in associated changes to 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 have to change, or alternatively, applications have to adapt to the inadequacies of the control architecture.
- Any changes to the control architecture have to take place in a synchronized fashion over the whole network.

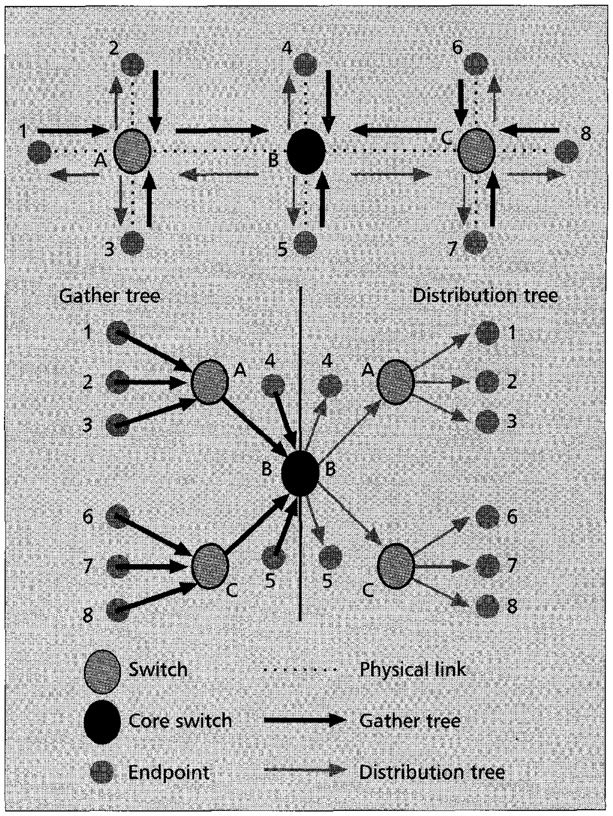
In addition to the above shortcomings, a general purpose control architecture, by its very nature, cannot exploit characteristics of certain applications. Furthermore, in some cases the functionality provided by a general purpose control architecture simply does not meet the requirements of some ATM capable devices. The Tempest framework offers a viable solution to these problems by allowing Service-Specific Control Architectures (SSCA) to be deployed. An SSCA utilizes 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.

We now motivate the need for and usefulness of service-specific control architectures by describing one that has been implemented in the Cambridge Computer Laboratory.

Videoman — A multiparty video conferencing environment has several characteristics which can be exploited by an SSCA to provide a scalable solution that efficiently utilizes network resources:

- 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 in 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 speakers.)

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 manipulated by the control architecture as directed by the *floor control* function to ensure



■ Figure 5. *Gather and distribution trees for single site media distribution.*

that resource 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.

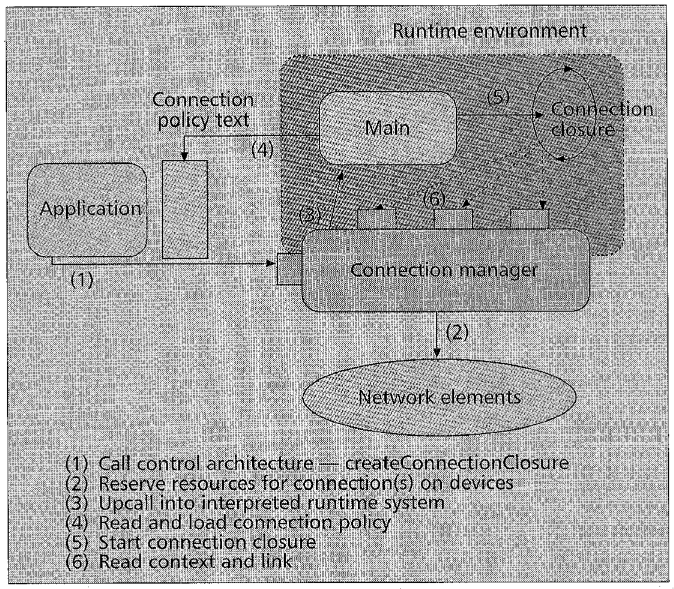
Videoman uses the inherent multicasting and inverse multicasting capabilities of ATM switches, as well as application-specific knowledge to control the network. Figure 5 depicts the tree structures created by the Videoman control architecture for a single site video conference. The top part of the figure shows three switches, with the core of the tree located at switch B in the center. The arrows show the direction of data flow in the two trees. The bottom half of the figure "unfolds" the same network topology to better show tree structures.

Application Specific Control Architectures

Service-specific control architectures give service providers the ability to program their own control, but requires them to write and deploy a complete control architecture; this technique is therefore best suited to well defined complete services such as those mentioned in the previous section.

At a finer level of granularity we can allow individual applications to determine the control policy for a set of connections, without having to write a complete control architecture. When an application creates a connection it can pass an application-defined control policy as well as a connection description into a specially designed control architecture. The control policy is a dynamically loadable program in some appropriate programming language which the control architecture can read, load, and run.

After the control architecture has successfully allocated the resources on the diverse network devices specified in the con-



Runtime environment
Connection policy text
Main
Connection closure
Application
Connection manager
Network elements
(1) Call control architecture — createConnectionClosure
(2) Reserve resources for connection(s) on devices
(3) Upcall into interpreted runtime system
(4) Read and load connection policy
(5) Start connection closure
(6) Read context and link

■ Figure 6. *Closure creation.*

nection description, a handle on these resources is combined with the application-provided control policy to form a connection closure. This closure is then executed within the context of the control architecture.

The connection closure interacts with the network only via the handles on the resources it is allocated. Connection closures are free to manipulate the resources in whatever way they see fit; the role of the control architecture is simply to:

- Allocate resources to applications
- Provide an interface to the resources
- Provide the context in which connection closures execute

The value of the connection closure concept is best explained by a specific example application of the technology in the context of mobile ATM.

Connection Closures in Mobile ATM — One of the key challenges of mobile ATM is the necessity to distinguish between different types of applications within the mobile control architecture [20]. Certain applications are keen to learn about changes in network conditions in order to be able to adapt their behaviors whilst others wish to operate transparently of the fact that the control architecture supports mobility. One way to address this problem is to allow applications to register an interest in events — such as handoffs — at the medium access level [20].

Connection closures offer an alternative solution. The closure can take the responsibility within the control architecture for modifying its application's connections. Since the connection closure may be executing within the same address space as other parts of the control architecture, this can be done very efficiently. In this way, the application rather than the control architecture can decide the policy to use for mobile control operations, such as handovers.

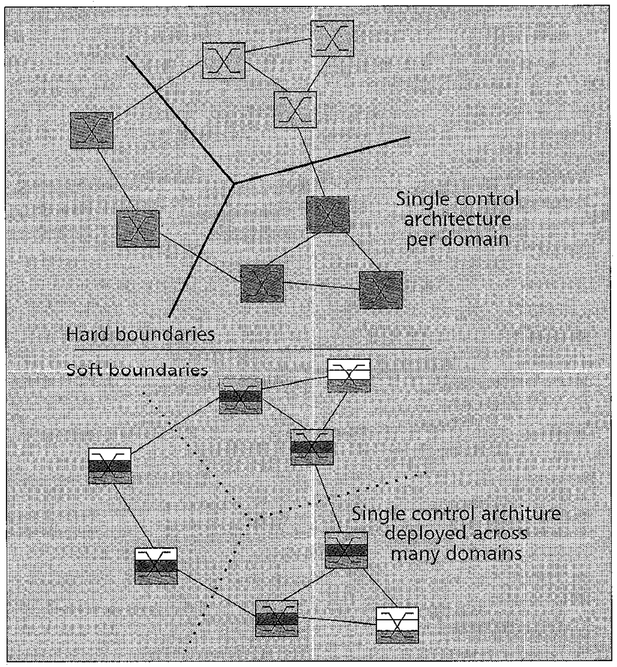
The following proof-of-concept implementation shows how the loading of a connection closure into the Hollowman allows it to functions as a flexible mobile ATM control architecture without having to modify it in any way.

Proof-of-Concept — The Hollowman control architecture was extended in order to support the loading of code into the Hollowman connection-manager. Figure 6 shows a schema for the reading, loading, and instantiating of connection closures. An application signals to the Hollowman informing it of its required resources and the location of the control policy that should be used to control those resources (1). In our current implementation this policy is defined in Java byte code. The connection manager reserves resources for the connection on the switch (2) and performs an upcall into the interpreted runtime environment (3). The interpreted runtime environment reads the policy over the network, loads it in to the runtime environment and allocates it the required resources on the network devices (4). It then combines handles on the resources with the control policy in order to form the connection closure, which it starts to execute (5). Finally the connection closure can start executing the downloaded code which manipulates the connections as specified by the application (6).

The mobile connection closure is about 200 lines of Java and compiles to 4 kilobytes of Java byte code. After the closure is loaded and resources are allocated, the time to establish and modify connections is mainly determined by the time taken to communicate between the mobile entity and the closure. Although the Java code is the top layer, it is a very thin layer and does not have much influence on connection set up time.

Networks Without Boundaries

A more subtle use of the virtual networking facilities provided by the Tempest framework relates to the crossing of administrative boundaries. In conventional virtual private networks (VPNs), the creation of a virtual network is handled by means of bilateral agreements and standardized vertical interfaces between domains. Within a particular domain the service is, however, provided and controlled by the network operator in that domain and is limited in terms of functionality to that provided by the local operator. In contrast, because it provides protection between virtual networks, a Tempest virtual network can cross administrative boundaries, with a single service (or control architecture) employed throughout the virtual



Single control architecture per domain
Hard boundaries
Soft boundaries
Single control archititure deployed across many domains

■ Figure 7. *Conventional hard administrative boundaries versus Tempest soft administrative boundaries.*

network and potentially controlled by a single operator. In this manner the vertical interfaces between domains are replaced by horizontal (switchlet) interfaces which could reside in various domains and be controlled by a single control architecture. Figure 7 depicts the difference between these hard and soft administrative boundaries.

Current experimental work has been limited to small scale implementations within the Cambridge University Computer Laboratory.[2] Recent initiatives such as the Learnet Project [21], which is based on the Tempest, will allow experimentation to be extended to a wide area network covering multiple administrative domains. In particular, this will enable the boundless networks concept to be further explored.

Related Work

Signaling System No. 7 (SS7) [22] is the signaling network and suite of protocols used in the Public Switched Telecommunication Network (PSTN). Intelligent Networks (IN) [23] uses SS7 for the introduction of services other than basic telephony into the PSTN. The speed of new service introduction is measured in years.

The ITU-T [24] and the ATM-Forum [7, 8] have used SS7 as their model for ATM signaling. Using telephony signaling as the basis for general multiservice networks is questionable. The limitations of this approach have been witnessed by various alternative control architectures that have been proposed and implemented including: the TINA connections management architecture [25], the Xbind binding architecture from Columbia University [17], the Distributed Call Processing Architecture from Bell Labs [16], and the UNITE lightweight signaling protocol from AT&T Labs [26]. Many of these control architectures use the services of a distributed processing environment such as CORBA, but none has formalized the current need for a primal network such as IP in their model.

The attractiveness of applying different control architectures to the same switching platform has also found favor within the IP community. Initial work on the IP Switching approach by Ipsilon [6] has lead to the creation of an IETF working group on *Multiprotocol Label Switching* [27] which aims to broaden the scope of this approach to protocols other than IP and switching technologies other than ATM.

Despite all this work on network control, the Tempest approach is unique in allowing a framework in which various control architectures, potentially of different types, can be simultaneously operational each contained in its own virtual network. This represents a significant shift in thinking because creating the one control architecture which will fulfill all needs is now no longer an issue.

Indeed, it is the ability of the Tempest framework to allow virtual networks of arbitrary topology to be created and to then be controlled by *any* control architecture which sets it apart from other virtual network approaches. Virtual network offerings in the Internet are normally of an "overlay" type using some sort of address mapping function and the capability to encapsulate virtual network traffic in regular IP packets [28, 29]. Similarly, ATM-based virtual networks normally assume a single control architecture and provide limited flexibility in terms of topology changes and resource management [30, 31].

Several proposals which can be loosely grouped under the umbrella of "active networks" have recently been made [32]. Most originate from within the Internet community, or at least assume that the unit of transfer within the network is datagram based [33, 34]. This work is motivated by reasons similar to our own. However, because traditional datagram-based networks lack a clear distinction between in-band and out-of-band control, the impact of this approach on any guarantees that the network could provide is not clear. This is aggravated by the fact that their current implementations seems to rely on the ability to handle data transfer in software.

Conclusion

The Tempest framework, by allowing multiple control architectures to coexist, addresses problems of migration and upgrading that all network operators are familiar with. It also allows completely different widely deployed control architectures such as UNI signaling and IP Switching to run simultaneously.

If these were the only problems that it solved it would still be a significant contribution, but more importantly, in our opinion, is the liberating of network operators from the task of defining a single, unified, best, control architecture. As anyone who can obtain a virtual network will effectively be a network operator, we hope to see an increase in the creativity that can be brought to bear upon the problem of network control. We have demonstrated that the Tempest framework provides this flexibility while permitting comparable efficiency to current solutions.

New techniques mentioned in this article, such as the use of software mobile agents, dynamically incrementable control architectures, and control architectures dedicated to a single service, can all be experimented with safely in existing, deployed, networks. It is difficult to countenance how the concept of active and programmable networks could ever be realized without a Tempest-like infrastructure.

Acknowledgments

The development of a framework such as Tempest represents a significant effort. The authors are indebted to all who worked on the Tempest framework, in particular: Rebecca Isaacs, Dave Halls, Brian Cowe, and Herbert Bos.

References

[1] J. van der Merwe and I. Leslie, "Switchlets and Dynamic Virtual ATM Networks," *Integrated Network Management V*, May 1997, pp. 355–368.

[2] S. Rooney, "An Innovative Control Architecture for ATM Network," *Integrated Network Management V*, May 1997, pp. 369–380.

[3] J. van der Merwe and I. Leslie, "Service Specific Control Architectures for ATM," *IEEE JSAC*, vol. 16, Apr. 1998, pp. 424–436.

[4] S. Rooney, "Connection Closures: Adding application-defined behaviour to network connections," *Computer Commun. Rev.*, vol. 27, no. 2, Apr. 1997, pp. 74–88.

[5] S. S. Sathaye, "ATM Forum Traffic Management Specification Version 4.0," *ATM Forum Technical Committee – Contribution 95-0013*, 1995.

[6] P. Newman, G. Minshall, T. Lyon, and L. Huston, "IP Switching and Gigabit Routers," *IEEE Commun. Mag.*, vol. 35, no. 1, Jan. 1997, pp. 64–69.

[7] ATM-Forum, "ATM User-Network Interface Specification – Version 4.0, (UNI 4.0)," The ATM-Forum: Approved Technical Specification, July 1995, af-sig-0061.000.

[8] ATM-Forum, "Private Network-Network Interface Specification – Version 1.0 (P-NNI 1.0)," The ATM Forum: Approved Technical Specification, Mar. 1996, af-pnni-0055.000.

[9] J. D. Case, "A Simple Network Management Protocol," Internet RFC 1157, May 1990.

[10] M. Laubach, "Classic IP and ARP over ATM," Internet RFC 1577, Jan. 1994.

[11] OMG "The Common Object Request Broker: Architecture and Specification Version 2.0 (CORBA 2.0)," tech. rep., The Object Management Group (OMG), 1995.

[12] G. Li, "Dimma Nucleus Design," Tech. Rep. 1553.00.05, APM, Poseidon House, Castle Park, Cambridge, CB3 ORD, UK, Oct. 1995.

[2] *The University of Cambridge has made a version of the Tempest available under terms of a royalty free license for research purposes only. The Tempest work has also led to the establishment of a start-up company.*

[13] J. van der Merwe, Open Service Support For ATM, PhD thesis, Cambridge University, Computer Laboratory, UK, Sept. 1997.
[14] S. Rooney, The Structure of Switch Independent ATM Control, PhD thesis, Cambridge University, Computer Laboratory, UK, 1997, to be available as a technical report.
[15] M. Veeraraghavan, T. L. Porta, and W. S. Lai, "An Alternative Approach to Call/Connection Control in Broadband Switching Systems," *IEEE Commun. Mag.*, vol. 33, no. 11, Nov. 1995, pp. 90–96.
[16] M. Veeraraghaven, "Connection Control in ATM Networks," *Bell Technical Journal*, vol. 2, Winter 1997.
[17] A. Lazar, "Programming Telcommunication Networks," *IEEE Network*, vol. 11, no. 5, Sept./Oct. 1997, pp. 8–18.
[18] A. Battou, "Connections Establishment Latency: Measured Results," ATM Fogrum T1A1.3/96-071, Oct. 1996.
[19] D. Niehaus *et al.*, "Performance Benchmarking of ATM Networks," *IEEE Commun. Mag.*, vol. 35, no. 8, Aug. 1997, pp. 134–143.
[20] P. Agrawal *et al.*, "SWAN: A Mobile Multimedia Wireless Network," *IEEE Personal Commun.*, Apr. 1996, vol. 3, no. 2, pp. 18–33.
[21] S. Crosby *et al.*, "Pearl: Proposal for Experimental Academic Research using LEANET," Available from http://www.cs.ucl.ac.uk/staff/jon/pearl/pearl.htm, 1997.
[22] ITU-T, "Specification of Signalling System No. 7," ITU Recommendation Q.709, ITU publication, 1993.
[23] ITU-T, "Recommendation I.312/Q.1201, Principals of Intelligent Network Architectures," ITU publication, 1992.
[24] ITU-T, "Draft Recommendation Q.2931, Broadband Integrated Service Digital Network (B-ISDN) Digital Subscriber Signalling Systems No. 2, User-Network Interface layer 3 specification for basic call/connection control," ITU publication, 1994.
[25] J. Bloem *et al.*, "TINA-C Connection Management Components," *Proc. TINA Conf.*, Melbourne, Australia, Feb. 1995.
[26] G. Hjalmtysson and K. Ramakrishnan, "UNITE—An Architecture for Lightweight Signalling in ATM Networks," *Proc. INFOCOM*, San Francisco, Apr. 1998.
[27] R. Callon *et al.*, "A Framework for Multiprotocol Label Switching," Internet Draft: draft-ietf-mpls-framework-01.txt, July 1997.
[28] M. Macedonia and D. Brutzman, "Mbone provides Audio and Video across the Internet," *IEEE Computer*, vol. 27, Apr. 1994, pp. 30–36.
[29] N. Doraswamy, "Implementation of Virtual Private Networks (VPNs) with IP Security," Internet Draft: draft-ietf-ipsec-vpn-00.txt, Mar. 1997.
[30] V. Friesen, J. Harms, and J. Wong, "Resource Management with Virtual Paths in ATM Networks," *IEEE Network*, vol. 10, no. 5, Sept./Oct. 1996, pp. 10–20.
[31] M. Chan, A. Lazar, and R. Stadler, "Customer Management and Control of Broadband VPN Services," *Integrated Network Management V*, May 1997, pp. 301–314.
[32] D. Tennenhouse, S. Garland, L. Shrira, and M. Kaashoek, "From internet to activenet," Request for Comments, Jan. 1996. Available from: http://www.sds.lcs.mit.edu/activeware
[33] D. L. Tennenhouse *et al.*, "A Survey of Active Network Research," *IEEE Commun. Mag.*, vol. 35, no. 1, Jan. 1997, pp. 80–86.
[34] D. S. Alexander *et al.*, "Active Bridging," *Computer Commun. Rev. (SIGCOMM '97)*, vol. 27, Oct. 1997, pp. 101–111.

Biographies

JACOBUS E. VAN DER MERWE (kobus@research.att.com) received B.Eng. and M.Eng. degrees from the University of Pretoria, South Africa, in 1989 and 1991, respectively. He completed a Ph.D. at the University of Cambridge Computer Laboratory, Cambridge, U.K. in 1997 and has since joined AT&T Labs Research in Florham Park, New Jersey. His research interests are in network control and management, network resource management and the Internet.

SEAN ROONEY received B.Sc. and M.Sc degree in Computer Science from The Queen's University Belfast in 1990 and 1991, respectively. After three years at the research center of Alcatel Alsthom at Marcoussis working in the field of network management, he started working for his Ph.D. degree at Cambridge University. His current research interests are in ATM network control and self-regulating control systems.

IAN LESLIE received a B.A.Sc in 1977 and an 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.

SIMON CROSBY holds a B.Sc in mathematics and computer science from the University of Cape Town, South Africa, an M.Sc in computer science from the University of Stellenbosch, South Africa, and a Ph.D. in Computer Science from Cambridge University, where he is currently a lecturer in the Computer Laboratory and a Fellow of Fitzwilliam College. His research interests include performance of communications systems and operating systems, protocol design, and distributed systems.