— The Tempest — A Practical Framework for Network Programmability

J.E. van der Merwe, S. Rooney, I.M. Leslie and S.A. Crosby
University of Cambridge

Abstract— **The Tempest framework allows 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.**

I. Introduction

The need for a single network supporting a large number of diverse services is one of the major factors driving networking research. Some new services, e.g., video-on-demand, require precise guarantees from the network both in regard to the timeliness of the arrival of their data and its correctness. There is general agreement that other than by over provisioning, the only way to achieve this is through the reservation of resources on the set of network elements across which the data flows. These resources are usually termed a *connection* and the establishment of a connection is often termed signalling.

Two visions of multi-service networks can be distinguished: those which are extensions of telephony [1] and those that are extensions of data transfer between computers [2]. The latter are based on the Internet protocol suite, while the former assume ATM as the universal network protocol. IP is more flexible and robust in the face of network failure than ATM, while ATM allows more precise guarantees to be made about network resource usage. Recent work [3] has shown how compromises can be reached between the two and this trend can be expected to continue. Indeed our working assumption is that large scale multi-service networks will use both native ATM, when precise guarantees are required, and IP when only a best-effort service is needed.

The ATM standards have defined a succession of control primitives between the user and the ingress/egress switch [4], and between switches [5]. While the standards recognise that applications should be given as much freedom as possible in the specification of their resource needs, there is an implicit assumption that all applications can be satisfied by these same set of primitives. ATM signalling betrays its original inspiration in that it strongly resembles telephony. As [6] points out, the ATM control plane is necessarily complicated, controlling as it does a wide variety of services.

In this paper, we define a *network control architecture* as the set of policies, algorithms, mechanisms and protocols used to control and manage the various devices in a network. The distinction between management and control is blurred. In this paper management will be taken to refer to those functions concerned with the "well-being" of network devices, while control will refer to functions which try to manipulate network devices into doing something useful. In ATM and other connection oriented networks the control architecture is fairly well defined and encompasses all the functions that are not involved with the actual data transfer in the network, examples include SS7 [7] and UNI/PNNI [4], [5]. While similar functionality has recently appeared in connectionless networks such as IP, e.g. IFMP [8] and RSVP [2], the distinction between control and data transfer is not always clear in these networks.

Recently within the networking research community there has been a realisation that the control architectures for multi-service networks need to be more flexible. The open signalling concept [9], [10] attempts to loosen the association between the control plane and the switching plane allowing both to evolve independently. Open signalling control architectures can therefore be much more readily deployed and upgraded than existing ones. More importantly perhaps, open signalling allows some of the basic assumptions about the design of the control plane to be reexamined.

This paper describes the necessary infrastructure for allowing multiple control architectures to run simultaneously over the same physical ATM network. We call the set of required components *The Tempest* [11], [12]. The Tempest depends on our ability to simultaneously control a given ATM switch with multiple controllers by strictly 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; our work [13] has identified the advantages of *service-specific* control architectures dedicated to one type of high-level service e.g. video conferencing. At an even finer level of granularity we have shown how end-users can dynamically associate code with network resources in a running control architecture [14] in order to achieve *application-specific* control. We call the association of the user defined control policy and the allocated network resources a *connection closure*.

The running of multiple control architectures simultaneously is a new idea, as a consequence techniques for ensuring the general health of the network need to be rethought. On-going experiments have been looking at issues as diverse as the application of mobile software agents moving between network nodes in order to achieve more adaptive management [15], and the use of measurement based effective bandwidth estimation to perform resource management at various levels of granularity in our framework.

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

II. The Tempest

A. Motivation

Control in a packet forwarding network involves determining whether and to where a packet gets forwarded. Normally, the control policy is viewed at two different levels of granularity and time scales:

- at the network node level (also called in-band control);
- at the control architecture level (also called out-of-band control).

In an IP network all control is performed at the first level. Each IP router makes a control decision for each arriving packet individually. This has the great advantage of being both simple and robust, but as any attempt to use a service with temporal constraints such as the MBONE [16] at "rush hour" will demonstrate, it does not permit much in the way of resource guarantees. ATM and extensions to IP, such as RSVP, have the notion of a connection or a flow. A connection is an abstraction of a set of diverse resources across the network; transmitted packets are associated with connections. In such a control architecture each network node determines whether packets arriving on a connection can be forwarded using only the resources currently allocated to that connection (traffic policing) and on a longer time scale the control architecture determines if and across which nodes the connection should be established (connection routing and call admission control).

The distinction between the two levels of 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 standardisation, the interaction between the control architecture and the switching plane is not specified, allowing switch vendors to implement proprietary solutions.

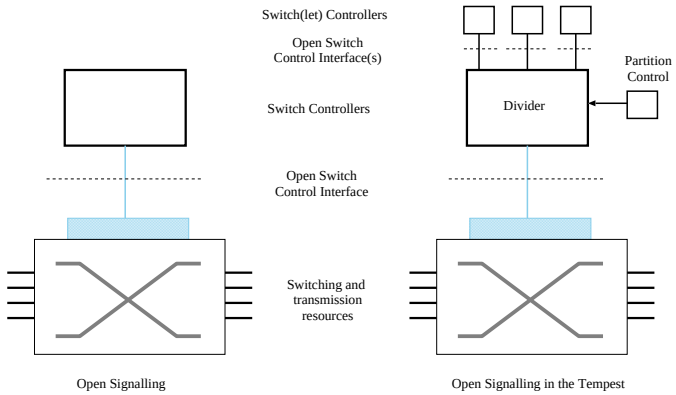

Fig. 1. Open Signalling and Open Signalling in the Tempest

Open signalling has proposed to formalise 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 evolved independently. It is envisaged that this will allow more flexible ATM control and the faster introduction of new services.

Our early work on open signalling led us to the realisation that some of the basic assumptions regarding switch control could be rethought. By strictly 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 Figure 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 decision not only about the resources allocated to their connections,

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

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.

B. The Ariel Switch Independent Control Interface

The *Ariel* switch independent control interface is the means by which control architectures communicate with the switch. 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 context contains four delay and loss parameters: Cell Delay Variation; Maximum Cell transfer delay; Mean Cell transfer delay; Cell Loss Ratio, and six traffic parameters: peak cell rate; sustainable cell rate; cell delay variation tolerance; maximum burst size; minimum cell rate; feedback algorithm for flow control. Different combinations of these parameters will be significant for the different ATM service categories of the connection.

The parameters follow those defined by the ATM forum in [17] 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. queueing 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.

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.

C. The Prospero Switch Divider

Ariel is a switch control interface for open signalling. This basic functionality has been extended by a crucial component of the Tempest framework, namely the Prospero Switch Divider [11]. 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, Figure 2 shows three possible controllers, namely for the Hollowman control architecture [12], and the IP Switching [3] and ATM Forum's UNI/PNNI [4], [5] control architectures. Partitioning of the switch into switchlets is controlled through a separate Prospero Management interface on the Divider Controller, the use of which is considered in more detail in later sec-

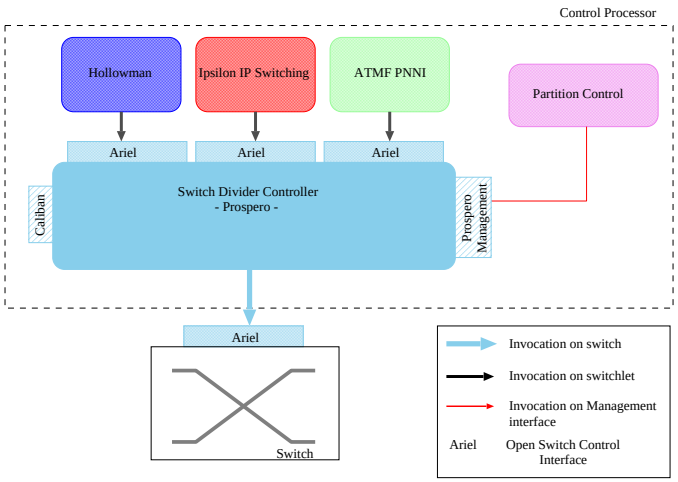

Fig. 2. Creating switchlets

tions. Prospero also contains the Caliban server for general switch management in the Tempest framework, 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 that 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.

D. The Caliban Switch Management Interface

Health functions, such as fault management and performance monitoring permit the well being of the network to be verified and maintained. The Ariel switch interface allows the full range of ATM control functions to be implemented within a logically distinct control layer.

Health functions require more information than control functions about the switch implementation in order to achieve their goal. For example, it may be necessary to access any piece of information contained in the memory of the switches in order to determine why it is not functioning as expected. For this reason management protocols have tended simply to be abstractions of memory with some restricted means of accessing and modifying memory locations rather than high level operations.

SNMP is a very successful, widely deployed management protocol [18]. 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 strict 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 Figure 2.

Caliban may be thought of as an indirection which allows some independence from the precise switch communication protocol and permits the required fine grained access control. The cost of this is that the client is restricted to using only those primitives in the Caliban interface, rather than the potentially richer management protocol, as well as some overhead in communication time.

E. 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.

E.1 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 [19], which in turn requires the fairly heavyweight ATM Forum control architecture [4], [5]. For example, more lightweight control architectures providing an IP service, such as IP Switching [3], might be more appropriate.

The bootstrapping problem is present in all ATM (and other) networks, and is not unique to the Tempest environment. For example, the ITU-T specify the use of meta-signalling in order to create signalling channels and thereby bootstrapping the network [1]. By means of the bootstrap virtual network, the bootstrapping facility has, however, been generalised into something that can provide more sophisticated services. For example, in addition to providing a means of managing virtual networks, the bootstrap virtual network can provide a set of generic network services which can be used by other control architectures and Tempest components.

E.2 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. CORBA/OMG [20] specifies things such as the language for defining interfaces, transports between clients and servers, formats for interface references and common services required by most applications. CORBA, due to its position as a standard environment and the large number of implementations both freeware and commercial, is a convenient DPE for the Tempest. So for example, the interfaces of the Tempest services are defined in CORBA Interface Description Language (IDL). The CORBA implementation used in the described version of the Tempest framework is Dimma [21].

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 paper, 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.

F. 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 Mbps
- 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.

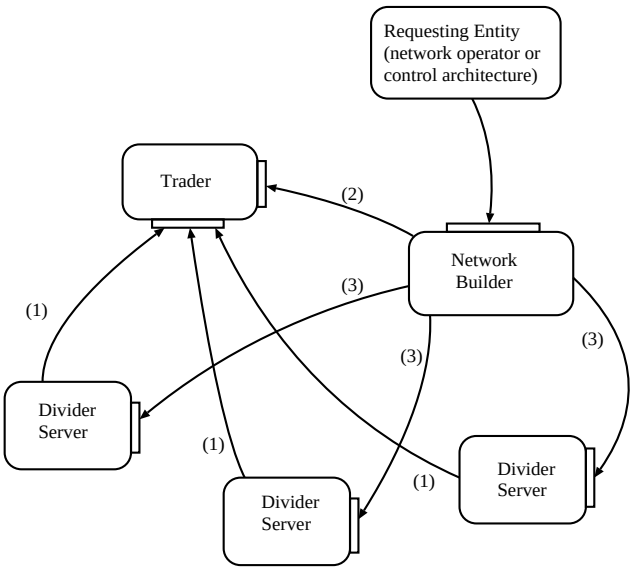

Fig. 3. Dynamic virtual network services

With reference to Figure 3 the creation of a virtual network can be summarised with the following steps:

1. At startup the Divider Server registers with the Trader its configuration and capacity. This process is repeated whenever any of these capabilities change.
2. 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.

3. The Network Builder invokes appropriate methods on the Divider Server(s) to create the switchlets and resulting virtual network.

These interactions are described in more detail in the remainder of this section.

The Network Builder has knowledge about existing virtual networks, and coordinates the creation of virtual networks so that, for example, the address space allocated to two adjacent switchlets in the same virtual network overlaps. When requested to create a virtual network, the Network Builder contacts the Trader and asks for all switches with the required capabilities and capacity according to the supplied network specification. The result of the query, invocation (2) in Figure 3, is a list of interface references to Divider Servers matching the criteria.

Using the supplied network specification together with topology information obtained from some topology service (which is not discussed in this paper) the Network Builder determines which switches should form part of the virtual network. The Network Builder then invokes the required operations on appropriate Divider Servers to create the switchlets, invocation(s) (3) in Figure 3.

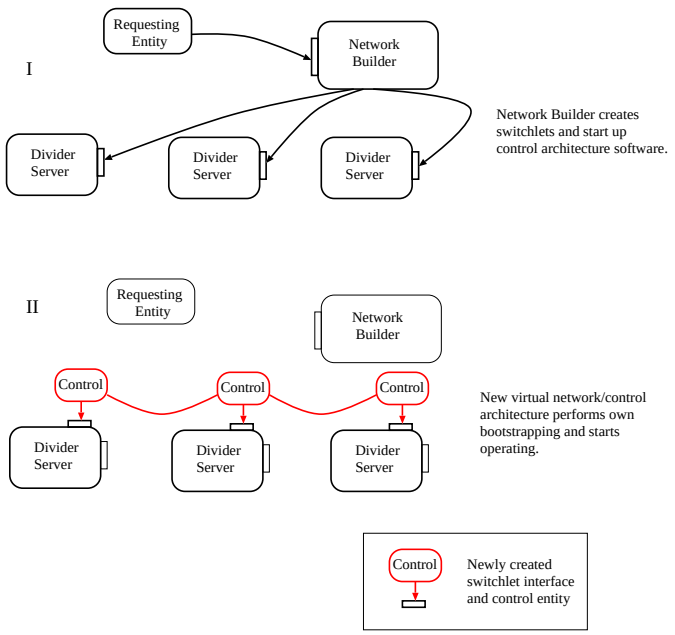

Fig. 4. Creating a virtual network of predefined type

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. Figure 4 depicts this process.
- Alternatively, is 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. This is the case depicted in Figure 5. Since interaction with the switchlet interface involves the services of the DPE, these control architectures will normally be required to be DPE aware.

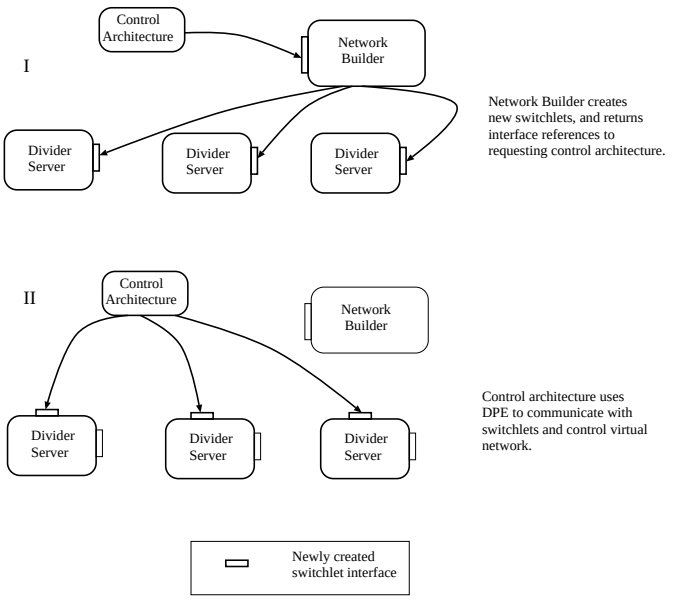

Fig. 5. Creating an anytype virtual network

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 which liberates its virtual network. The resources which comprise that virtual network, then become available for use by other control architectures.

G. The Hollowman Control Architecture

The Hollowman control architecture serves three distinct purposes:

- It is an exemplar 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 re-

quire 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 signalling.

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 signalling 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 synchronises 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 synchronises the actions of the host-managers and a set of soft switches in order to establish the end-to-end connection. Figure 6 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.

H. Experimental Results

All components of the Tempest framework described in this paper have been realised 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 [22], [23].

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 in Section 2.2 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 compare favourably with comparable figures published elsewhere [3].

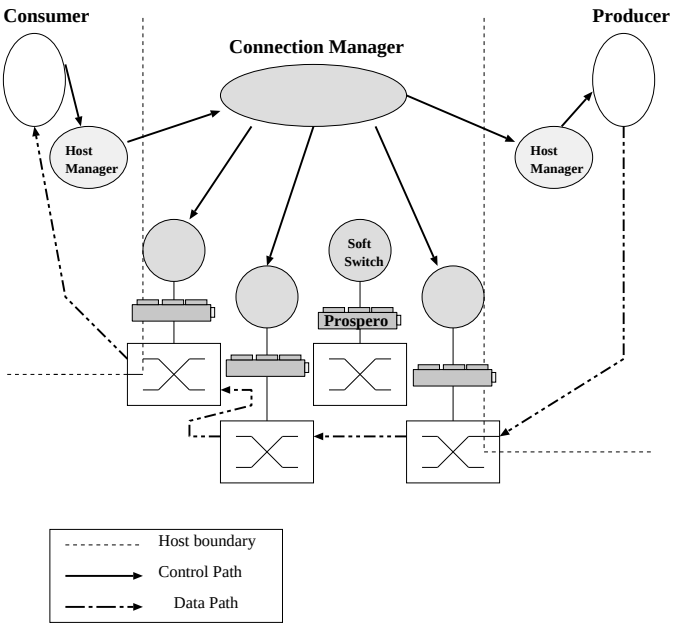

Fig. 6. Hollowman control path

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 7 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 parallelising the connection establishment over a number of switches in order to reduce the signalling latency and "caching" certain connections in the expectation that they will be reused, also with the objective of reducing latency. [24], [25] and more recently [9] have proposed similar techniques. Even without these optimising techniques the Hollowman compares quite favourably in terms of performance with more conventional signalling systems [26], [27], 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 [27].

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

III. 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.

A. 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 [4]. 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 synchronised fashion over the whole network.
- The control architecture has to cater for all possible applications in a generic fashion, which leads to inefficient and complex protocols.

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 in allowing *Service Specific Control Architectures* (SSCA) to be deployed. 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.

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 multi-party video conferencing environment has several characteristics which can be exploited by an SSCA to provide a scalable solution that efficiently utilises 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 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 multi-casting and inverse multicasting capabilities of ATM switches, as well as application specific knowledge to control the network. Figure 7 depicts the tree structures created by the VideoMan control architecture for a single site video conference.

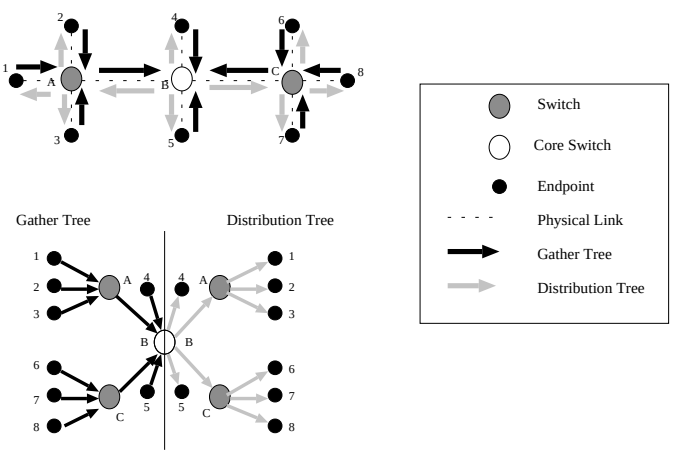

Fig. 7. Gather and distribution trees for single site media distribution

B. 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 connection 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.

An example of the use of closures is now given:

Mobile ATM

Reference [28] describes the SWAN mobile ATM control architecture and identifies one of the challenges of mobile ATM as the necessity to distinguish within the mobile control architecture between different types of applications. Certain applications are keen to learn about changes in network conditions in order to be able to adapt their behaviours whilst others wish to operate transparently of the fact that the control architecture supports mobility. [28] states that current ATM APIs are tailored for static environments and only allow basic control operations such as VCI establishment and tear-down; SWAN gets around this problem by allowing applications to register an interest in events - such as hand offs - at the medium access level

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 hand-overs.

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

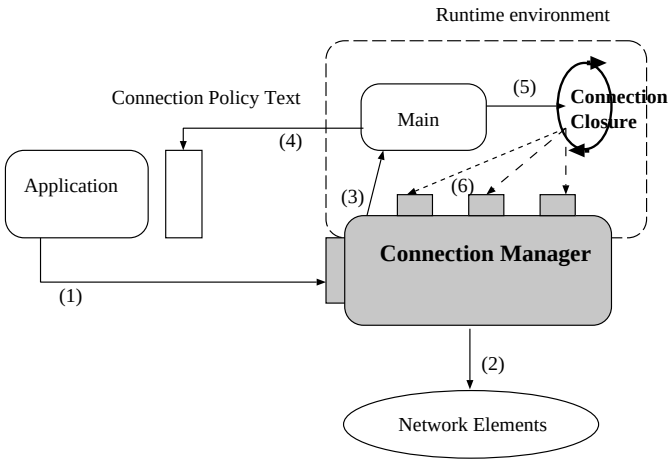

Fig. 8. Closure creation

The Hollowman control architecture was extended in order to support the loading of code into the Hollowman connection-manager. Figure 8 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. In our current implementation this policy is defined in Java byte code. The Hollowman reads the policy over the network, loads it in to the runtime environment and allocates it the required resources on the network devices. It then combines handles on the resources with the control policy in order to form the connection closure, which it starts to execute.

We have experimented with using closures to define the application-specific strategy for managing mobile ATM connections. Many different mobile closures can run simultaneously allowing each to have their own policy.

The mobile connection closure is about 200 lines of Java and compiles to 4 kilo-bytes 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.

IV. On-going work

The Tempest is a new approach and on-going research is exploring some of the remaining open issues. These mainly relate to how the Tempest and the control architecture are

themselves managed, especially in relation to recovery from failure and the management of resources. Some of this work is now discussed.

A. Adaptive Management

Mobile Agents in the Tempest

Mobile code is data that can be executed as a program; an autonomous mobile code system is one in which an executing program can stop executing on one location, and ask to be moved to another location on which its execution will be resumed. The combination of mobile code and the environment it executes in is called a *mobile agent*. The attractive feature of mobile code is that if the mobile code can reach the host at which a service is running, then all communication with that service can take place locally. We are using the Tempest framework in order to investigate the *possibility* of using mobile software agents for carrying out Operation and Maintenance (OAM) functions for control architectures within the Tempest.

Our experiments have taken the form of allocating to a certain Tempest control architecture its own set of mobile agents. The mobile agents of a control architecture know the topology of the virtual network of the control architecture. The mobile agents move using a dedicated virtual network, allowing the amount of resources the agents use to be bounded. The control architecture which manages this virtual network is extremely simple; all the connections across which the agents move are created at start of day, removing the need for signalling to take place in order to perform OAM functions. This is analogous to the situation in the ATM standards where OAM messages are passed in-band.

The control architecture specific mobile agent can communicate with the control architecture entities at a particular host using the normal interfaces of those entities. The sequence of operations that a mobile agent uses to perform its function is determined by the mobile agent. Since new mobile agents can be introduced without affecting a control architecture, message types can be added dynamically.

The mobile agent uses its knowledge about the topology of the virtual network of its control architectures to determine where it should be sent next. The mobile agent may also take additional factors into account, for example, where it has already been, when taking this decision. This allows the OAM messages to discriminate about where they are sent without adding complexity to the Tempest infrastructure or the control architecture. Moreover, the normal operation of the control architectures is kept completely distinct from their exceptional behaviour when managing failure.

Mobile agents allow processing to be brought to state, rather than state to processing. The mobile agent can examine the state at one location and then correlate that state with that which it finds at the next location it executes at. Since the mobile agent can process the state locally, it can reduce the amount of information to be transported. If this saving is larger than the size of the agent, then the total amount of data that need be transported is reduced. As the agent is control architecture specific, it need only examine parts of the state that concern its control architecture.

The use of mobile agents reduces the need for doing the correlation centrally, thus allowing for a more scalable solution. It increases the probability of correct fault detection as the distance between the location where an alarm is emitted and the location where it is interpreted, is lessened, thereby reducing the amount of "noise" that has to be filtered out. It allows more adaption, since the mobile agents can change their behaviour as a function of the state of the network. For example, if they detect the network is overloading due to alarm emission they start suppressing the emission of alarms. Such a system is inherently decentralised, allowing information to be gathered and management decisions to be made closer to the concerned network elements.

Experiments with Automatic Resource Freeing

The Tempest framework has resulted in the implementation of many experimental control architectures. A control architecture which, due to programming error, only partially liberates or creates a connection pollutes the VCI space. Sometimes failure can occur in subtle ways. For example, programming error in handling concurrent connection creation requests can lead to erroneous connections being created. The control architecture's view of the network becomes out of phase with the real state.

The existence of erroneous connections may not become manifest until some time after their occurrence. Deciding whether the complete set of connections that a given control architecture possesses is valid or not, requires correlating information from the virtual channel tables of all the switches that the control architecture uses. This task is laborious and better automated. It would have been possible to write a central entity which constantly polled the switches across the network in order to determine whether the VCI space for all control architectures was healthy or not. However this is not a scalable solution as it involves the transfer of the complete virtual channel tables from all switches. It makes more sense to move the program to the switch state rather than the other way around. Mobile agents offer a natural way of achieving this.

The experiments that this section describes address a practical problem. While this problem is particular to the experimental environment it is an exemplar of a class of problem in which distributed data must be correlated.

Automatic Resource Freeing (ARF) agents were implemented to operate in the Tempest framework [15]. Each ARF agent is associated with a particular control architecture. The agent moves through the virtual network of its control architecture and invokes operations on the Ariel interfaces of the switchlets in this virtual network to determine the state of connections. The ARF agent compares the reported state of connections in two adjacent switchlets to find "suspicious looking" connections, e.g., connec-

tions leaving one switch port that are not terminated on an adjacent switch. The ARF agent queries the control architecture about the existence of such connections, and connections unknown to the control architecture are removed.

The ability of ARF agents to move themselves means that the processing goes to the state, rather than the state to the processing (i.e. to the control architecture). Since ARF agents are small, this reduces the amount of network communication required to clean up resources.

The efficiency with which Operation and Maintenance functions are executed is of much less importance than their need for robustness and making minimal demand on network resources. The overhead in executing interpreted code is therefore not critical. Of more concern is that the use of software mobile agents in the way suggested requires that every location in the network that they visit be capable of executing them. To some extent whether the approach defined here is useful will depend on the ubiquity of environments to run those agents and the security guarantees which those environments can enforce.

The Tempest environment's ability to upper-bound the amount of network resources that mobile agents can use, makes it the ideal environment for carrying out experiments of the type described in this section.

B. Resource Management

Resource management in the Tempest needs to be considered at different levels of granularity and on different time scales. Firstly, resource management needs to be performed at the call or connection level. This process is normally called *connection acceptance control* or *connection admission control* (CAC). CAC deals with the issue of whether, given the current state of the network, a new connection can be accepted into the network in such a way that its requested resource guarantees can be satisfied without adversely affecting existing connections. Equally important in the Tempest is resource management at the virtual network level. The first issue to be addressed is similar to the CAC problem, namely whether a new virtual network can be created given the current state of the network, or *virtual network acceptance control* (VNAC). Bandwidth allocated to a virtual network can be guaranteed for the duration of its existence (hard guarantee). Alternatively, a virtual network can be given a statistical or soft guarantee and ATM's potential multiplexing gain can be exploited to allow more virtual networks to be created. This is similar to peak rate allocation versus effective bandwidth allocation in CAC [29]. A more general concept specific to Tempest virtual networks, is the ability to arbitrarily allocate and reallocate bandwidth resources in a virtual network environment. Resource management at the virtual network level normally happens at a slower time scale than that required by CAC.

Recent developments [30] in an approach based on estimation of effective bandwidth computed from applying large deviation theory to online traffic measurements, have shown some very promising results. This approach has the further advantage that it can equally well be applied to both single connections as well as groups of connections. This feature is particularly attractive in the Tempest environment where the partitioning of switch resources into switchlets requires resource management at both the call/connection level and the virtual network level. A further attractive feature of the measured effective bandwidth approach is that it enables the network to exploit the multiplexing gain of several multiplexed data streams. This advantage can be used at all levels of bandwidth management.

For these reasons effective bandwidth estimators using online measurements have been implemented and experiments were performed to investigate its use in the Tempest framework. Figure 9 shows some of the results from these experiments.

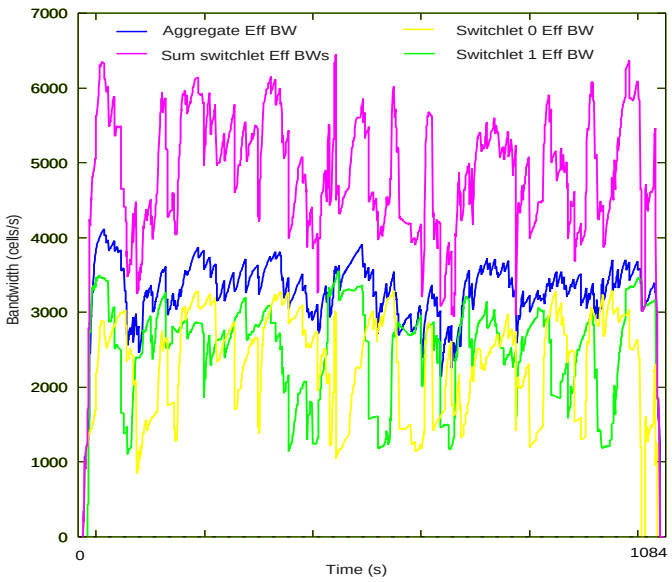

Fig. 9. Effective bandwidth estimates for the switchlets, their sum and the switch port as a whole

In the particular experiment a switch was partitioned to form two switchlets. The results are from a port which formed part of both switchlets. Each switchlet carried a number of bursty traffic streams. Each traffic source waited for a random period before requesting that a connection be established, and the connection then lasted for a random period of time. The bottom two lines in Figure 9 show the effective bandwidth estimates for each switchlet over the time of the experiment. The third line from the bottom shows the aggregate effective bandwidth estimates for the switch port as a whole, in other words without taking the switchlet partitions into account. The top line in the figure is the sum of the effective bandwidth estimates for the switchlets. The difference between the aggregate and sum lines shows the multiplexing gain achieved by the multiplexing of the bursty sources. Within the Tempest framework, this multiplexing gain can be exploited at either a connection of virtual network level.

On-going work is exploring the use of these techniques on switches with more sophisticated buffer management and scheduling mechanisms than those used in the work presented.

C. Networks Without Boundaries

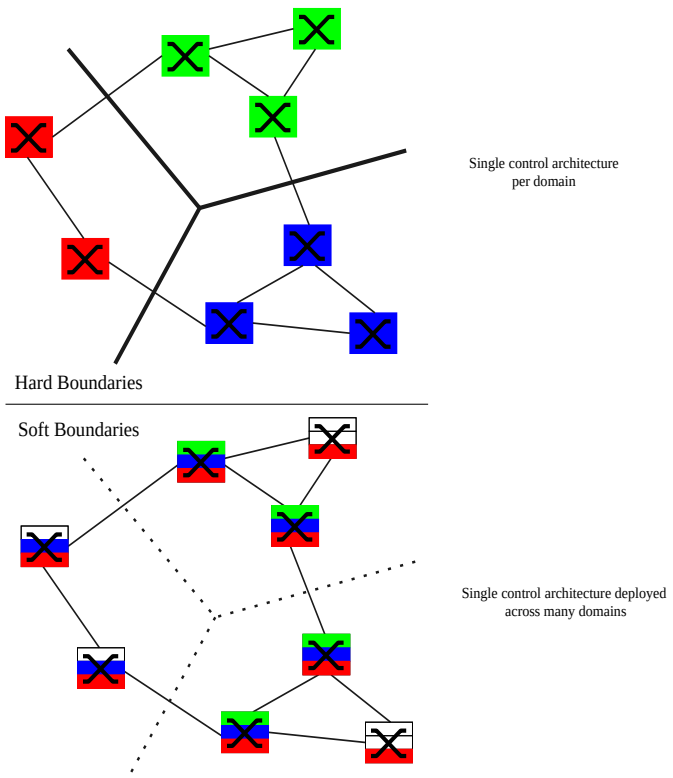

Fig. 10. Conventional hard administrative boundaries versus Tempest soft administrative boundaries

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 standardised 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 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 10 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 [31], 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.

V. Related Work

Signalling System No. 7 (SS7) [7] is the signalling network and suite of protocols used in the Public Switched Telecommunication Network (PSTN). Intelligent Networks (IN) [32] 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 [1] and the ATM-Forum [4], [5] have used SS7 as their model for ATM signalling. Using telephony signalling as the basis for general multi-service 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 [33], the Xbind binding architecture from Columbia University [9], the Distributed Call Processing Architecture from Bell Labs [25] and the UNITE lightweight signalling protocol from AT&T Labs[34]. Many of these control architectures use the services of a distributed processing environment such as CORBA, but none has formalised 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 favour within the IP community. Initial work on the IP Switching approach by Ipsilon [3] has lead to the creation of an IETF working group on *Multiprotocol Label Switching* [35] 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 fulfil 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 [36], [37]. Similarly, ATM based virtual networks normally assume a single control architecture and provide limited flexibility in terms of topology changes and resource management [38], [39].

Several proposals which can be loosely grouped under the

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

umbrella of "active networks" have recently been made [40]. Most originate from within the Internet community, or at least assume that the unit of transfer within the network is datagram based [41], [42]. 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.

VI. 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 signalling 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 paper, 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 realised without a Tempest-like infrastructure.

VII. Acknowledgements

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] 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.

[2] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: a new resource ReSerVation protocol," *IEEE Network*, vol. 7, pp. 8—18, September 1993.

[3] P. Newman, G. Minshall, T. Lyon, and L. Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, vol. 35, pp. 64—69, January 1997.

[4] 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.

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

[6] T. Chen and S. Liu, "Management and Control Functions in ATM Switching Systems," *IEEE Network*, vol. 8, pp. 27–39, July/August 1994.

[7] ITU-T, "Specification of Signalling System No. 7," *ITU Recommendation Q.709, ITU publication*, 1993.

[8] P. Newman, W. L. Edwards, R. Hinden, E. Hoffman, F. C. Liaw, T. Lyon, and G. Minshall, "Ipsilon Flow Management Protocol Specification for IPv4 - Version 1.0," *Internet RFC 1953*, May 1996.

[9] A. Lazar, "Programming Telcommunication Networks," *IEEE Networks*, vol. 11, pp. 8–18, September/October 1997.

[10] P. Newman, "Ipsilon's General Switch Management Protocol Specification Version 1.1," *Internet RFC 1987*, August 1996.

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

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

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

[14] S. Rooney, "Connection Closures: Adding application-defined behaviour to network connections," *Computer Communication Review, 27(2):74—78*, vol. 27, pp. 74—88, April 1997.

[15] D. Halls and S. Rooney, "Controlling the Tempest: Adaptive management in advanced ATM control architectures," *IEEE JSAC*, vol. 16, pp. 414–423, April 1998.

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

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

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

[19] M. Laubach, "Classic IP and ARP over ATM," *Internet RFC 1577*, January 1994.

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

[21] G. Li, "Dimma Nucleus Design," Tech. Rep. 1553.00.05, APM, Poseidon House, Castle Park, Cambridge, CB3 0RD, October 1995.

[22] J. van der Merwe, *Open Service Support For ATM*. PhD thesis, Cambridge University, Computer Laboratory, UK, September 1997.

[23] S. Rooney, *The Structure of Switch Independent ATM Control*. PhD thesis, Cambridge University, Computer Laboratory, UK, 1997. To be available as technical report.

[24] M. Veeraraghavan, T. L. Porta, and W. S. Lai, "An Alternative Approach to Call/Connection Control in Broadband Switching Systems," *IEEE Communications Magazine*, vol. 33, pp. 90—96, November 1995.

[25] M. Veeraraghaven, "Connection Control in ATM Networks," *Bell Technical Journal*, vol. 2, Winter 1997.

[26] A. Battou, "Connections Establishment Latency: Measured Results," *ATM Fogrum T1A1.3/96-071*, October 1996.

[27] D. Niehaus, A. Battou, A. McFarland, B. Decina, H. Dardy, V. Sirkay, and B. Edwards, "Performance Benchmarking of ATM Networks," *IEEE Communications Magazine 35(8):134—143*, vol. 35, pp. 134—143, August 1997.

[28] P. Agrawal, E. Hyden, P. Krzyzanowski, M. Srivastava, and J. Trotter, "SWAN: A Mobile Multimedia Wireless Network," *IEEE Personal Communications Magazine*, pp. 18—33, April 1996.

[29] J. Y. Hui, "Resource allocation for broadband networks," *IEEE Journal on Selected Areas in Communication*, vol. 6, pp. 1598–1608, December 1988.

[30] J. T. Lewis, R. Russell, F. Toomey, B. McGurk, S. Crosby, and I. Leslie, "Practical Connection Admission Control for ATM Networks Using On-line Measurements." Invited paper for Computer Communications. To appear, 1997.

[31] S. Crosby, I. Leslie, J. Crowcroft, L. Sacks, and C. Todd, "Pearl: Proposal for Experimental Academic Research using LEANET." Available form http://www.cs.ucl.ac.uk/staff/jon/pearl/pearl.htm, 1997.

[32] ITU-T, "Recommendation I.312/Q.1201. Principals of Intelligent Network Architectures," *ITU publication*, 1992.

[33] J. Bloem, J. Pavon, H. Oshigiri, and M. Schenk, "TINA-C Con-

nection Management Components," *Proceedings TINA Conference, Melbourne, Australia*, February 1995.

[34] G. Hjalmtysson and K. Ramakrishnan, "UNITE - An Architecture for Lightweight Signalling in ATM Networks," in *INFOCOM, San Franciso*, April 1998.

[35] R. Callon, P. Doolan, N. Feldman, A. Fredette, G.Swallow, and A. Viswanathan, "A Framework for Multiprotocol Label Switching," *Internet Draft: draft-ietf-mpls-framework-01.txt*, July 1997.

[36] M. Macedonia and D. Brutzman, "Mbone provides Audio and Video across the Internet," *IEEE Computer*, vol. 27, pp. 30–36, April 1994.

[37] N. Doraswamy, "Implementation of Virtual Private Networks (VPNs) with IP Security," *Internet Draft: draft-ietf-ipsec-vpn-00.txt*, March 1997.

[38] V. Friesen, J. Harms, and J. Wong, "Resource Management with Virtual Paths in ATM Networks," *IEEE Network*, vol. 10, pp. 10–20, September/October 1996.

[39] M. Chan, A. Lazar, and R. Stadler, "Customer Management and Control of Broadband VPN Services," in *Integrated Network Management V*, pp. 301–314, May 1997.

[40] D. Tennenhouse, S. Garland, L. Shrira, and M. Kaashoek, "From internet to activenet." Request for Comments, January 1996. Available from: http://www.sds.lcs.mit.edu/activeware/.

[41] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A Survey of Active Network Research," *IEEE Communications Magazine*, vol. 35, pp. 80–86, January 1997.

[42] D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith, "Active Bridging," *Computer Communications Review (SIGCOMM 97)*, vol. 27, pp. 101–111, October 1997.