The Tempest
a Framework for Safe, Resource Assured, Programmable Networks

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

Abstract

Most research in network programmability has stressed the flexibility engendered by increasing the ability of users to configure network elements for their own purposes, without addressing the larger issues of how such advanced control systems can coexist both with each other and with more conventional ones. The Tempest framework presented here extends beyond the provision of simple network programmability to address these larger issues. In particular we show how network programmability can be achieved without jeopardizing the integrity of the network as a whole, how network programmability fits in with existing networks and how programmability can be offered at different levels of granularity. Our approach is based on the Tempest's ability to dynamically create Virtual Private Networks (VPNs) over a switched transport architecture, e.g. an ATM network. Each VPN is assigned a set of network resources which can be controlled using either a well known control system or a control system tailored to the specific needs of a distributed application. The first level of programmability in the Tempest is fairly course grained: an entire virtual network can be programmed by a third party. At a finer level of granularity the Tempest allows user supplied code to be injected into parts of an operational virtual network, thus allowing application specific customisation of network control. The paper shows how the Tempest framework allows these new approaches to coexist with more conventional solutions.

I. Introduction

Much networking research over the last decade has explored ways in which new services can be more quickly introduced into existing networks [1], [2], [3]. All of these approaches have attempted to make the network more *programmable* in some way; they differ as to:
- The classes of individuals they permit to program the network,
- The nature of the programming allowed, and
- The extent that reprogramming of network elements can be carried out without affecting existing services.

For example, for Intelligent Network (IN) services [1] within the Public Switched Telecommunication Network (PSTN), the programmers are essentially network operators, the IN programming interface supplied by the equipment vendor is limited to high-level telephony specific primitives and new services are developed off-line and only introduced into the actual network when thoroughly tested. More recently proponents of *Active Networks* [3] have argued that the class of programmer should be extended to potentially include all network users; that programmers should be able to dynamically add code written in a general purpose programming language to the network elements and have that code interact with the element across a low-level interface. This low-level interface might, for example, give access to routing tables and offer control of the underlying crossconnect. The key aim of the approach is to enable the addition of user code to the network element to be part of the normal operation of the network, allowing new functionality to be rapidly introduced into the network, perhaps on the time-scale of a single session or even packet.

The more general the programmability offered by the network, the greater the flexibility that results, but the more difficult the task of ensuring the integrity of the network becomes. Indeed much of the controversy surrounding the active networks debate has resulted from pre-conceived and outdated architectural assumptions about the capabilities of switches, routers and their operating systems support environments. In our opinion, different levels of programmability will be appropriate in different contexts, the choice being dictated by the bandwidths of the communications paths, the power of the processing elements available in the network, and the degree of hardware integration offered by switching elements. Certainly in the core network the scope for customised programming of switching elements seems severely constrained by the limited processing power available compared to typical data transmission rates. This view is supported by active networking proponents.

Whatever the scope adopted, any attempt to make a network programmable raises a number of major concerns:
- How do we allow third party activity within the network without jeopardizing the integrity of the network as a whole?
- How do we accommodate existing networking solutions, which although they have their shortcomings, are at least known to work?
- How do we manage the transition to the new world and interwork with legacy parts of the network?

Essentially, the key need which must be addressed in order to facilitate programmability in networks is that of *safe sharing* of network resources. The Tempest addresses this need by:
- Providing a framework in which virtual private networks (VPNs) can be dynamically created and assigned a dedicated set of network resources (including processing, routing, table space, buffer space and bandwidth) at each of the network nodes.
- Enabling each VPN to be controlled by its *own* instance of a control architecture, whether standards-based, such as the ATM Forum's PNNI [4] or alternatively a user programmable VPN control system such as Xbind [2]. All VPNs

share a common physical switching infrastructure and it is the control software operational in a VPN that determines the type of VPN. We use the term *control architecture* to refer to all such control software.

• Firewalling VPNs from each other by ensuring that only the owning control architecture instance of the VPN can perform out-of-band control operations on it. VPNs therefore enjoy resource guarantees throughout a switched core network.

Thus, the Tempest facilitates the provision of resource-assured VPNs, solving a pressing need for per-VPN Quality of Service (QoS) guarantees in the Internet. Moreover the guarantees offered are of a nature which suits the needs of Service Providers wishing to offer Service Level Agreements (SLAs) to their corporate customers.

While the Tempest is currently implemented on commercial ATM hardware this does not imply ATM as specified by the ATM Forum; although a Tempest VPN can be a fully compliant ATM Forum network. Moreover the reader will observe that the Tempest approach is more widely applicable than to cell-switched architectures, applying equally to modern IP switch-routers.

In our opinion, the successful introduction of more programmability into the core network will require a Tempest like infrastructure. Otherwise, it is difficult to envisage how the different degrees of programmability appropriate for different services could coexist nor how very advanced control architecture, such as Active Networks, could be safely introduced over a physical infrastructure offering more basic, existing, services.

The remainder of the paper is structured as follows. Section II motivates our approach and presents an overview of the Tempest framework showing the different building blocks and how they interwork to allow a managed VPN environment. Different control architectures which have been, or are currently being integrated with the Tempest framework are presented in Section III. This includes the Hollowman which provides for simple network programmability in two ways:

• It provides a set of tools for the construction of new control architectures, and

• At a finer level of granularity it allows the injection of user supplied code into an already running control architecture.

Section IV considers how the Tempest environment interworks with non-Tempest networks. Related work is explored in Section V and we conclude with remarks on the scope for extension of our framework.

II. The Tempest

A. Motivation

As with other approaches to network programmability [2] and active networking [3], our work has been motivated by a desire to increase flexibility and innovation. As such we agree with the line of thinking which questions the "one-size-fits-all" approach of current networking architectures. Too often we have seen monolithic standards devised by committee, imposed on otherwise flexible architectures. In the case of ATM, for example, it is ironic that a transfer mechanism which was designed to be flexible enough to support a wide range of services has become burdened with a control system based on the standard telephony model. At the same time we realize the truth of the "one-size-fits-many" approach; indeed it might be argued that IP routing is the control system of primary concern nowadays, however it is still the case that in IP the control system is evolving; resource constrained routing protocols and the use of RSVP (Resource reSerVation Protocol) [5] resource reservation are examples of this. Therefore one of the fundamental architectural principles of the Tempest was to avoid prohibiting the use of any current or future network control architecture. On the contrary, our goal is to provide an environment which is conducive to the continued evolution of network control, on multiple time scales.

The cell switching capabilities of ATM hardware provide a very attractive platform on which to achieve this goal. The forwarding of cells is determined by a short identifier in the cell header, which has local significance only. Ensuring that these local identifiers are used in a meaningful way is the responsibility of control software (collectively control architecture) running on the switch controller. Exploiting the separation of data and control is at the heart of the Tempest approach.

In the Tempest the need for per-VPN customisation of control and inter-VPN resource firewalling is met by partitioning the switch resources into subsets for use by multiple control systems. Rather than have a single switch control entity, implementing a 'multiplex' of control functions as would be the case in conventional ATM switches, the Tempest allows several switch control entities to be simultaneously operational on a switch. Each of these control entities operates on a partition of the physical switch resources. We call such a partition a "Switchlet"[1] because to all intents and purposes the switchlet appears to the control software to be a complete switch. Figure 1 shows a switch simultaneously supporting two instances of Multi-Protocol Label Switching (MPLS) [6], one instance of the standard ATMF control architecture [4] (UNI/P-NNI) and an instance of a recently developed lightweight signalling system called UNITE [7].

The switch partitioning is enforced by a software entity called a divider. This arrangement is analogous to a multi-tasking operating system which allows several programs to execute "simultaneously" on a single processor. The divider polices invocations on the Switchlet by each control architecture, confining control actions of each control architecture

[1] The Tempest framework is being commercially exploited to deliver QoS assured VPNs by a Bay Area startup, Control Plane Technologies (`cplane.com`). The switchlets concept is the subject of international patent applications.

to its legitimately owned subset of the resources. Attempts to control or modify resources not 'owned' by a Switchlet are analogous to application faults on an operating system and are rejected by the divider. The interface between the control architecture and divider is "open" allowing control systems to be supplied by parties other than the switch vendor; this is in contrast with partitioning mechanisms that a switch vendor might use to simply isolate two distinct control architectures offered by that vendor on the same switch. Note that switchlet resources reside on the physical switch, because a switchlet is a partition of the physical switch. However, the divider has knowledge of and controls, these resources and partitions.

Of course, control is also required in the data path - in this case the analogy is with the operating system scheduler which must ensure that applications are not allowed to consume more than their allocated share of operating system resources, e.g. CPU time. On an ATM platform this is delivered by the built-in policing functions on each switch.

The division of resources of the physical switch, both in the control and data path, allows us to address the first two concerns raised in the Introduction:

- The control operations on switch resources can be strictly policed by the divider to ensure that only the assigned resources are modified/used. It is thus impossible for one control system to adversely affect another control system or indeed the rest of the network beyond its own confined set of resources.
- In this way it is possible to support existing control architectures, such as ATM Forum standards-based systems, as simply "another" type of control entity executing on the switch (or Switchlet to be more correct).
- By opening up the Switchlet interface it is possible to support enhanced functionality, by allowing each VPN to be customised to a particular purpose. Because of the protection offered by the divider, even a rogue control system will not be able to affect resources other than those specifically assigned to it, thereby ensuring the safety of the network as a whole.

A consequence of our approach is that network programmability is limited to out-of-band network control. (A control architecture can of course use different cell scheduling mechanisms but is fundamentally limited in its offering by the underlying functionality of a particular switch.) In an ATM switch fabric, cell forwarding is normally carried out by specialized hardware and in-band control is very simple and efficient. Allowing programmable in-band control would compromise this simplicity. (It is worthwhile to note that optimized forwarding in specialized hardware is also becoming the norm in high performance IP routers which would place similar restrictions on the flexibility available in the data forwarding path.). Note that our approach does not preclude the existence of special purpose devices, such as format converters, in the data path, however this can be in a "network server" and need not be in the general optimized data path. Here again the resources of such servers can be divided (assuming that a sensible operating system is being used), to ensure that each VPN is assured of a fair proportion of the computational resources available.

Extending our discussion from a specific node to a network of nodes, it is easy to see how individual Switchlet partitions can be connected to form virtual networks, each with dedicated sets of resources at the switches that it traverses. Again, the fact that in the data path identifiers have only local significance means that the address space used in one particular virtual network is completely separate from that used in other virtual networks. The virtual networks are therefore private virtual networks (or virtual private networks VPNs). The consequence of this is that:

- Different virtual networks can use different addressing schemes, e.g. IP and ATM addressing for example.
- Different virtual networks can use the same addressing scheme with logically distinct address spaces. (This has an interesting and useful side-effect, namely the potential to salvage vast tracts of IP address space through re-use of the name space within multiple VPNs).
- A virtual network can use a private and proprietary addressing scheme.

In summary, the Tempest allows the coexistence of multiple VPNs controlled by distinct control architectures over the same physical network. The network operator is free to choose the proper allocation of resources to each of these VPNs and the most appropriate control architecture to use to control it. For example the operator might decide that half of the available resources should be divided between four VPNs each supporting a separate Internet service and being controlled by distinct instances of MPLS, while the remaining resources should be used to support a continuous media service, controlled by the ATMF control architecture. The network operator may allow third parties to use their own control architectures to control their VPNs: for example two network operators might agree to collaborate such that each would allocate a VPN to the other over its own physical network and allow the other to use its own dedicated control architecture to manage it. This enables network programmability at the VPN granularity and allows for VPNs which are dedicated to a specific service running service specific control architectures [8]. Moreover, control architectures can themselves be programmable, allowing third parties to dynamically change the control policies for a given VPN, without affecting the control of other VPNs. This allows fine grained application specific programmability [8]. Service providers are thus able to to 'dial up' VPNs with robust Quality of Service guarantees, on demand. Each VPN is allocated a dedicated subset of the resources at a switch or router, allowing operators to implement and charge for per-VPN SLAs. The potential exists for dynamic extension of VPNs: mobile users can thus join a VPN remotely, and enjoy guaranteed performance access to corporate LANs. In addition, each VPN can implement customised control of its resources to support new network-aware applications. One might envisage enterprises using our approach to exploit advances in

IP telephony, data networking and new multimedia applications across a public Internet, with guaranteed performance dedicated to multiple VPNs, each of which is tailored to a specific function.

B. Tempest Building Blocks

In this section we describe how the architectural framework of the previous sections can be implemented, enumerating the core functional entities that make up the Tempest framework and enable it to deliver safe sharing of network resources between multiple control planes. The Tempest consists of the following functional components:

- the Prospero Switch Divider, which allows the resources of the switch to be partitioned between control architectures;
- the Ariel Switch Independent Control Interface, which a control architecture uses to control its part of the switch;
- the Caliban Switch Management Interface, which a network operator uses to manage its part of the switch, e.g. test performance, check for faults;
- the Bootstrap Virtual Network, which supports the communication needs of the Tempest infrastructure itself;
- the Network Builder, which allows virtual networks to be created;
- the Hollowman Control Architecture[2], which is both a fully functional ATM control and a set of components that third parties can use to build their own control architectures.

B.1 The Prospero Switch Divider

The Prospero Switch Divider is a crucial component of the Tempest framework and is depicted in Figures 1 and 2. Prospero is solely in control of the physical switch and partitions the switch into switchlets. It does not however exercise any control on the switch on its own behalf. Rather it relays control requests by the various control architectures executing on top of it. In addition to this multiplexing function, Prospero performs a policing function on all control requests issued by the various control entities. This policing function ensures that a control request, e.g. the setting up of forwarding information in the switch, will only make use of resources that were allocated to the switchlet in question. Local policy determines the strictness of the switchlet partitioning. In some cases the partitioning will be hard and a control entity will be tightly controlled to only use resources that were allocated to it, even if other switchlets are not using all their resources. Alternatively a control entity can exceed its allocation when resources are available, e.g. on a best effort basis.

Switchlet resources include a subset of the switch ports, VPI/VCI space, bandwidth and buffer space. More than one switchlet can share the same switch port, but all switchlets need not have the same set of ports. Switchlet resources are made available to the control entities by means of the Ariel switchlet control interface. This interface, discussed in the next section, allows a controller to control the switchlet in exactly the same way as it would control a physical switch.

Partitioning of the switch into switchlets is controlled through a separate Prospero Management interface on the Divider Controller. Partitioning of control access 'rate' is also important. Although not supported on our current implementation, we envisage partitioning (by means of operating system scheduling) processor resources to ensure that each control architecture has sufficient time to exercise control for its VPN. When there are a small number of control architectures this guarantee could be made in crude fashion by allocating a separate processor to each control architecture. The division of access rate on the divider itself could be policed simply by counting invocations.

Prospero also exports the Caliban switch management interface, which is discussed in Section II-B.3. The Prospero divider can either be implemented on the control processor or on an off-board processor which communicates with the switch by means of some open control interface. This represents a tradeoff between the functionality required on the switch versus the performance required by control architectures.

B.2 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.

In the case where the Prospero divider is implemented on an off-board processor, the switch Ariel interface (at the physical switch side of Prospero), is mapped to the control interface exported by the physical switch, e.g. Generic Switch Management Protocol (GSMP) [9]. In our on-board Prospero divider implementation the switch Ariel interface is directly mapped onto the switch fabric control interface. Similarly, for the switchlet Ariel interface (the interface on top of Prospero), Ariel can be mapped onto any of a number of open switch control interfaces, such as GSMP or a

[2] Although it is a core component of the Tempest framework, the Hollowman control architecture is described together with other Tempest aware control architectures in Section III.

portable switch-independent layer such as offered by many ATM software vendors. To demonstrate this mapping we have adapted Ariel to a vendor's generic switch fabric interface; this is described in Section III-B [3].

Ariel makes uses of the QoS parameters defined by the ATMF (e.g. Cell Delay Variation Tolerance, Maximum Burst Size, Minimum Cell Rate) in order to allow control architectures to specify the resources that should be associated with connections. These should be adequate for many applications and supported by most commercial switches and avoids requiring knowledge about the precise implementation of the fabric, e.g. queueing algorithms, which is unlikely to be available for commercial switches.

B.3 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. Currently most, if not all, commercial LAN ATM switches support SNMP. However, SNMP's lack of fine grained access control (even in SNMP v2), means that it is not suitable for the management of a Tempest VPN. Many commercial switches allow connections to be created and modified by means of the appropriate SNMP operations on the switch MIB. If this functionality were still supported in the management plane, in addition to a divider, a control architecture could subvert the partitioning imposed by the divider. 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. One solution to this problem is to partition the SNMP MIB, into multiple communities, each access controlled and dynamically updated to reflect the true resource allocation to the control architecture associated with the MIB partition. These partitioned MIBs (or MIBlets) can be made available to standard SNMP management tools if the communities are carefully managed.

A second, and indeed the chosen, solution to the management plane problem 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 the administrator of a particular VPN to verify and ensure the overall health of their 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.

Reference [10] identifies the difficulty in achieving generic, efficient management and refers to this as the *micro-management* problem: the nature of management operations means that they are low-level, e.g. consisting of stylised reads and writes; in consequence, many operations – and therefore communication exchanges – are required to achieve any useful function. Sophisticated functions (such as the sum of the traffic on all ATM VCs used for signalling within a VPN) require processing of potentially large amounts of MIB data, and would be most conveniently located with(in) the SNMP agent itself. [10] proposes a dynamically incrementable management server, termed an *elastic server*, to tackle this problem. In this system, the client is permitted to add code to the management server, for example the SNMP agent, running on the switch. The client code becomes an intrinsic part of the switch management interface for the client. The approach based on dynamic customisation of the management plane is particularly attractive within the Tempest, since it allows different Tempest users to customise their management policies and functions. This approach can be exploited by network managers to:
- correlate and filter data at the MIB;
- trigger the emission and reception of arbitrary patterns of OAM cells to test performance;
- add control architecture specific alarms.

An important restriction is that any dynamically loaded code must preserve access control, and be safe.

Caliban also addresses the problem of micro-management. Caliban clients can change the behaviour of the Caliban server by dynamically loading and executing code as close to the agent as possible. Dynamically loaded code has the same access control restrictions as its emitting client regardless of where it is executed.

B.4 Bootstrap Virtual Network

Remote access to and interaction with the management interfaces on the Prospero dividers of multiple switches is required in order to allow the network operator to create VPNs and manage their control architectures. Since the Tempest framework itself 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?*

[3] An open switch interface is in the process of being standardised within the IEEE P1520 "Standard for Application Programming Interfaces for Networks" working group (http://www.ieee-pin.org).

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 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. 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.

One of the useful things that the bootstrap virtual network can provide is the services of 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 DPE is the framework in which distributed entities cooperate in order to achieve some objective. We have used a CORBA-based DPE in our prototype implementations. This would allow us to easily adopt and incorporate control architectures developed in the OMG and TINA-C frameworks as Tempest control systems.

One of the services that most DPEs 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 in order to establish communication with and 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.

B.5 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 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 Figure 3 the creation of a virtual network can be summarized with the following steps:

1. At start of day each of the Divider Servers (each is associated with a physical switch) registers its configuration with the Trader. This is repeated whenever any of these capabilities change.
2. The Network Builder, on receipt of a virtual network specification from the network operator, consults the Trader to obtain interface references to all Divider Servers which satisfy the criteria of the request.
3. The Network Builder performs appropriate routing functions across the physical topology of the network, to establish a topology for the VPN to be built.
4. The Network Builder invokes appropriate methods on the selected Divider Server(s) involved in the VPN to create the switchlets and instantiate the control system for the VPN.

A control architecture can either build its VPN 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 the network builder can choose an optimal path through the network, or it can be allocated a predefined virtual network with a pre-configured topology. 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 the resources should be manipulated. In the case of conventional control architectures, such as ATMF signalling, the network builder will instantiate the appropriate control entities at appropriate nodes in the network. Of key importance is the fact that VPNs built in this way are dynamically extensible: a mobile user wishing to join a corporate VPN could be dynamically added to the network with the appropriate amount of resources to support their traffic; additional switchlets can be added into a VPN or deleted from it, when required.

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.

C. Resource Management

Resource management in the Tempest needs to be considered at different levels of granularity and on different time scales. First, within each VPN resource management may need to be performed at the individual call or flow level if the control architecture associated with the VPN wishes to implement such fine grained admission control. This process is normally called *Connection Admission Control* (CAC) in the ATM context. CAC deals with the issue of whether, given the current state of the (virtual) network, a new connection can be accepted without violating its resource and QoS requirements and without adversely affecting existing connections. In the Tempest context CAC is performed independently by each VPN if it requires this function, however (see below) the switch divider implements functions which effectively provide all of the information needed by a control architecture in order to make admission control decisions.

Equally important in the Tempest is resource management at the level of VPNs. The first issue to be addressed is similar to the CAC problem, namely whether a new VPN can be created given the current state of the network, a problem which we refer to as *Virtual Network Admission Control* (VNAC). Bandwidth allocated to a VPN can be guaranteed for the duration of its existence (hard guarantee). Alternatively, a virtual network can be given a statistical or soft guarantee and the available multiplexing gain resulting from the statistical multiplexing of switchlets can be exploited to allow more VPNs to be created than would be possible if each were allocated resources for its peak demand. This is very similar to peak rate admission versus the concept of admission based on the *effective bandwidth* of the traffic, as proposed by several authors. We address this in more detail below.

A more general concept in the domain of resource management which is specific to Tempest virtual networks, is our support for a mechanism which allows the network builder to arbitrarily allocate and reallocate bandwidth resources in a virtual network environment. This allows the resource allocation to a particular VPN to be dynamically changed based on a provisioning policy imposed by the network operator, for example the time of day. Resource management at the virtual network level typically occurs at a much longer time scale than that required of CAC, and is most appropriately placed in the management domain.

At the heart of our resource management strategy is a firm belief that the demand process of modern network traffic is inherently un-characterisable in the conventional stochastic modelling sense. It is hard enough in the standards-based ATM context to perform admission control, given a host of ATM Forum traffic parameters. But in the Tempest framework, no such luxury exists: the demands are inherently un-knowable because the traffic types are unknown. Only the VPN itself has any understanding of the concept of flows and the types of traffic which it carries. We have therefore capitalised on recent advances in measurement based admission control (MBAC) developed by some of the present authors [11].

Our approach is based on recent work based the on estimation of the so-called effective bandwidth of the arrivals process at a queueing system, which is computed from applying large deviations theory to online traffic measurements; reference [11] gives full details. The approach has shown very promising results, and has the further advantage that it can equally well be applied to both individual traffic flows as well as aggregate traffic flows. This feature is particularly attractive in the Tempest environment where the partitioning of switch resources into switchlets requires resource management at both the flow/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. Perhaps more importantly, it leaves as an open policy option whether a virtual network has rigid guarantees or whether it should be statistically multiplexed with other VPNs (and indeed with which other VPNs it might be willing to statistically multiplex).

Another argument in favour of this approach is that the success or failure of any resource management function, to a large extent, depends on its ability to deliver QoS guarantees at *reasonable cost*. If it requires users to know or derive large numbers of parameters which are difficult to obtain then they will be forced to allocate resources based on the peak rate. An alternative might be to use the traffic parameters such as the peak and mean rates, and the delay variation tolerance - these, however give insufficient information to achieve a useful degree of statistical multiplexing gain.

We built a toolkit which implements a set of algorithms which estimate, on-line, statistical parameters of the traffic demand viewed as a stochastic process. These parameters, akin to the thermodynamic entropy of a gas, provide precisely the information, that is required by our control system to determine the effective resource requirements of the traffic. The estimation procedure imposes minimal requirements on the switches while permitting a high degree of statistical multiplexing gain in the network. When the toolkit is fed a supply of traffic measurements it returns estimates of the effective bandwidth of the traffic given a QoS constraint for the buffer through which the traffic is multiplexed. The toolkit implements an estimation service both for the switch divider as a whole, and for the individual switchlets on a switch. It can be used as a CAC server by a control architecture such as Q.2931 or perhaps RSVP/Int-Serv, and also as a VPN admission controller by the divider. Figure 4 shows typical experimental results from the use of the estimation toolkit with multiple switchlets carrying ATM multimedia streams.

In this particular experiment a switch was partitioned to form two switchlets. The results are from a port divided between 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 4 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 (without taking the Switchlet partitions into account). The top line in the figure is the sum of the effective bandwidth estimates for the individual 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 or virtual network level. At a connection level, within a given VPN, multiple sources can be multiplexed to achieve statistical multiplexing gain while preserving an overall Service Level Agreement (SLA) for the VPN as a whole. Multiple switchlets can also be statistically multiplexed (if desired) in order to better utilise the network while still meeting the individual SLA requirements of the different VPNs.

D. On-going work

Our work on the Tempest is on-going; currently the Tempest is being readied for deployment in an experimental testbed with a large international operator. In the Laboratory, our research is exploring the following open issues:
- How best to distribute the building of networks, between domains managed by different operators.
- Economic models for VPN pricing.
- The application of the Tempest to WANs.
- Sharing basic infrastructure, for example OAM, auto discovery, between distinct control architectures.
- The addition of both experimental and standard control architectures to the Tempest environment.

The University of Cambridge has made a version of the Tempest available under terms of a royalty free licence for research purposes only.

III. Diverse VPNs in the Tempest

In this section we substantiate our claim that the Tempest supports diversity in the control plane by considering how different control architectures operate in the Tempest environment. With the exception of the UNITE protocol, discussed below, all other control architectures have been made to work in our experimental environment.

A. A lightweight control architecture

The UNITE signalling protocol was recently developed as a lightweight signalling mechanism in an ATM environment [7]. While lightweight signalling has been attempted before, we see the UNITE approach as an important development because the possibility of hardware based signalling holds the promise of at least an order of magnitude improvement in signalling rates and latencies.

UNITE based signalling need not be done in hardware, and in a software based implementation UNITE would simply be "another control architecture" as depicted in Figure 1. A hardware based UNITE implementation does however require special consideration when integrated with the Tempest environment.

UNITE has recently been extended to include a virtual private network identifier (VPNID). The VPNID is carried in the UNITE micro-setup message and allows demultiplexing during connection setup time so that the UNITE protocol can use different routing tables based on the different VPNs. This is depicted in Figure 5.

A small extension of the Switchlet control interface will allow different control architectures to populate the routing tables used by UNITE. This is shown in Figure 6. Since in this case the UNITE processing is assumed to be performed in hardware, the processing takes place below the divider. The divider will however control and police the way the UNITE routing tables are updated, in addition to its already described functions.

The question arises as to why other control architectures will be needed if UNITE performs connection setup extremely efficiently in hardware. The simplest answer is of course that as long as there are competing approaches to network control then the Tempest environment is essential. At a slightly more technical level consider the following:
- UNITE provides destination based routing as is the default in the current internet.
- For various reasons, e.g. QoS sensitive routing, source based routing as is done in PNNI might be preferred.
- Similarly, while the need for on-demand label distribution is recognised in MPLS, by default label distribution is achieved by distributing labels together with topology information.
- The flexibility provided by programmable control architectures might make use of novel connection structures.
- Finally, the Tempest provides QoS and resource management functions between VPNs, which is essential even if all VPNs are of the same type.

It is our view that these different approaches represent points on a spectrum and that the Tempest provides a framework in which they can be sensibly combined.

B. A standards based control architecture

Q.Port [12] is a commercial implementation of the ATMF signalling standard that we have slightly modified so that it executes in the Tempest environment.

Each host within the Q.Port domain runs a *host-manager* entity which is the initiator and receptor of signalling messages at a host. For each switch within the Q.Port domain there is a *switch-manager* entity which controls the progress of signalling messages at a given switch. Figure 7 shows Q.Port running within the Tempest framework.

Q.Port achieves some switch-independence by defining a *Fabric* interface that Q.Port switch controllers use to communicate with the actual switch. The Fabric interface is tightly coupled to the Q.Port control architecture and is not as general as Ariel. The execution of Q.Port within the Tempest environment is achieved by implementing the fabric interface using the more fundamental Ariel interface. Q.Port is a commercial implementation; the fact that it can be treated in the same manner as in-house control architectures supports the assertion that *any* control architecture can be run within the Tempest. More importantly it shows that a standards based control architecture can work within the Tempest environment, allowing the reuse of existing solutions.

C. A Simple proprietary control architecture

The Hollowman is an experimental ATM control architecture; it was designed in the first instance to test the Tempest environment. The Hollowman is a realistic ATM control architecture both in terms of the functions that it offers and the efficiency with which it performs those functions. The core Hollowman functions enable end-user applications to create and delete point-to-point, point-to-multipoint, and third-party connections between ATM capable workstations and devices using an Application Programmer Interface (API) which is small and simple to use.

The basic Hollowman entities are: the *soft switch*, through which the rest of the Hollowman interacts with the physical switch; the *host-manager*, which manages the resources of a host and the the *connection-manager*, which synchronizes the creation, modification and deletion of connections. The Hollowman also supports a trading service, by which applications can register and obtain service offers. Figure 8 shows the Hollowman running within the Tempest framework.

While Q.Port and the Hollowman offer similar control functions to applications, they privilege different non functional requirements. The Hollowman is a much simpler control architecture than Q.Port because it solves a less general problem, e.g. the Hollowman assumes the existence of a DPE - in the current implementation Adaptive Communication Environment (ACE) -[4] while Q.Port implements the full Q.2931 signalling protocol stack. The Tempest environment allows network operators to choose the control architecture that they require as a function of their needs and environment; this may involve selecting control architectures with different control functions, e.g. mobility, groupcast, third party signalling, or with different non functional requirements, e.g. trading efficiency against reliability.

Table I summarises some of the published results [13], [2], [14] for the latency in point-to-point connection creation across a single switch. Note that direct comparison of the absolute performance numbers is not sensible, because the results were obtained from a number of unrelated experiments. However, it is enough to demonstrate that there is no radical difference between open signalling systems and those more tightly bound to the switch. Although the Hollowman compares favourably with a variety of other signalling system, it is important not to overstate the importance of signalling latency. If a difference of some small number of milliseconds in signalling latency has adverse effects on an application, then it is probably inappropriate for that application to be carrying out signalling in its critical path.

Since many control architectures may run simultaneously within the Tempest, a small, specialised control architecture can be dedicated to a single service [8]. Although each of these control architectures is designed for a single specific service, e.g. video-conferencing, they share many common control functions. It should not be necessary within the Tempest environment to rebuild fundamental components, such as routing services, from scratch each time a new control architecture is developed.

With that in mind, the Hollowman can be viewed not simply as a control architecture, but as a set of components with which network operators can build their own control architectures within the Tempest framework. For example, the Sandman is a service-specific control architecture being developed at the University of Cambridge Computer Laboratory, which makes use of the Hollowman as a component library. The Sandman controls the network resources of a set of video servers containing parts of commercial films. It is hoped that by being able to create a schedule of connections, the network resource usage can be optimised.

D. RSVP as an ATM Control Architecture

The current version of IP, version 4, is well-suited to ‘traditional’ Internet usage, such as E-mail, news, file transfer and the world-wide web. More recently, multi-media applications, such as Internet telephony and video streaming, have become popular; these are much more sensitive to bandwidth and latency fluctuations, and consequently the IETF has developed the Integrated Services (Int-Serv) framework for flow characterisation and resource allocation. Int-Serv is often associated with RSVP (the Resource reSerVation Protocol) which is essentially an IP-centric signalling mechanism for the Internet based on leaf initiated joins and soft state at routers. RSVP reserves resources on intermediate routers, setting aside bandwidth for particular identified traffic flows. State is stored ‘softly’ in routers: it times out and must be periodically refreshed if the reservation is to remain. We decided that an implementation of RSVP *directly* as a control architecture for an ATM switch would be an interesting project, and would demonstrate the flexibility of the Tempest framework.

Our implementation supports soft state and most of the fundamental functionality of the current RFC2205 draft [5]. Our initial experiments on the performance of this system show that flow setup can be achieved as efficiently as connection setup in an ATM context. The drawback of using RSVP on ATM is that RSVP filters cannot be expressed in their full generality (it is impossible at layer 2 to perform layer 3 filtering functions, for example) however we believe that our approach to the use of RSVP on ATM is preferable to the more traditional multi-layered approach which uses RSVP on top of classical IP on ATM, thereby imposing two layers of signalling and one redundant layer 3 protocol on the process of reservation for flows with QoS requirements.

[4] Note that the Hollowman assumes existence of another control architecture capable of supporting IP. This is a legitimate assumption within the Tempest environment since it is capable of supporting multiple control architectures.

E. A Programmable Control Architecture

Within existing ATM control architectures, after signalling its requirements for a connection, an application delegates its control over the connection to the control architecture. The connection belongs to the application in the sense that the resources comprising that connection can be unambiguously accounted to it. There is no fundamental reason why the application should not use its own resources in whatever way it wishes, even if those resources are scattered across the various network devices over which the connection exists. This can be accomplished by offering applications a lower level API than that which is commonly offered by existing control architectures, with which to manipulate network resources. Lowering the level of the API increases the number of invocations that the application is required to execute to achieve some complete control function. If all these invocations need to be transported across the network from application to control architecture, then the latency of the operation will increase. This is a major disadvantage in itself, and also means that certain types of resource management are excluded purely due to the latency in communication between network and application.

At connection creation, an application should be able to pass into the control architecture an application-defined control policy as well as a connection description. The control policy is a dynamically loadable program in some suitable programming language which the control architecture can read, load and execute.

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 through the handles on the resources it is allocated. Connection closures are free to manipulate these resources in whatever way they see fit; the role of the control architecture is simply to:

- assign resources to applications;
- offer an interface to the resources;
- provide the context in which connection closures execute.

An application-defined entity can have knowledge about the use that end-systems are making of connections that a generic signalling system cannot. Executing a connection closure within the control architecture allows the closure to take advantage of this high-level knowledge while interacting with the network resources at a very low level. This technique will not replace the need for distinct control architectures but allows applications greater flexibility in the use of their connection resources.

A proof-of-concept implementation of closures has been carried out using the Hollowman. Closures allow applications to extend the function of the Hollowman. For example, loading a closure for mobile communication enables the Hollowman to behave like a mobile control architecture for a period; when the mobile closure stops running the Hollowman can 'forget' about mobility.

Figure 9 shows how closures are loaded into an extended version of the Hollowman.

IV. Interworking with rest of the world

Since the Tempest can accommodate standard control architectures interworking with non-Tempest environments is very straightforward. A VPN running a standard control architecture in the Tempest can directly interact with a similar control architecture outside the Tempest domain. However, more functionality is required to accommodate advanced control architectures such as the Hollowman. These advanced control architectures allow network operators and end-users a great deal of flexibility in the control over their network resources. This makes them convenient experimental platforms; to be more generally useful they need to be able to interoperate with other, more widely deployed, control architectures. This interoperation requires that the control operations of the advanced control architecture can be translated to and from the format of other control architectures, that routes to addresses outside the advanced control architecture's domain can be determined, and that routing information can be made available to other control architectures. In short an interoperation protocol is required.

A single standard ATM signalling mechanism — assuming it was accepted and widely implemented — would guarantee universal interoperation. Preceding sections have argued that this *one size fits all* approach is too restrictive and is unlikely to be successful. Alternatively, if every switch could support the switch divider then potentially any control architecture could be deployed universally. The following reasons argue against this as a solution:

- In all probability users will be restricted in their use of network elements outside their management domain.
- The number of virtual networks that can be supported by a network element at a given moment is bounded.
- Not all switches will have a switch divider.

So, while the Tempest environment allows a service provider to operate a global virtual network, in practice the extent of the virtual network is likely to be closely related to the equipment owned by the body that authorized that virtual network. There is still a need for interoperation between control islands belonging to different management domains.

The ATM Forum terms the group of signalling interfaces between two switches the Network-to-Network Interface (NNI). The Private-NNI or P-NNI [4], is intended for use within private networks. P-NNI contains two distinct interfaces;

one for the exchange of routing information between switches and a second for the management of connections. P-NNI is designed to scale to planetary scope.

We have extended the Hollowman so that it supports PNNI, demonstrating that advanced control architecture can interoperate with standard ones. Applications using this version of the Hollowman have the same scope in their signalling as those directly using the UNI. An advanced control architecture which implemented P-NNI is in fact a partial implementation of the standard ATMF one; it behaves as a normal P-NNI instance to other control architecture instances, however within its own domain it can avail of non-standard signalling techniques, e.g. connection closures, lightweight signalling.

V. Related work

A. *Virtual Private Networks*

In the telecommunications world virtual private networks (VPNs) have been a service offering since the mid eighties. A loose definition of a VPN in this context would be a network with a logically closed user group implemented over a public (switched) network. By and large these virtual networks are realised by allowing clever address mappings over which users can exercise some control. Although VPNs created in this fashion give every indication of being a dedicated virtual network as far as the customer is concerned, network resources are not necessarily dedicated to the VPN.

VPNs are also found in the Internet environment. The work of the IP security group provides the framework in which VPNs are offered. Current Internet VPN approaches concentrate on providing secure links between private IP networks through the public Internet and include address translation capabilities if required. Secure links are implemented by encapsulating protected data in the payload of a regular IP packet thus creating a secure tunnel through the regular IP network. Of course, owing to the best-effort model of the Internet, it is impossible to provide resource guarantees to a VPN or manage Service Level Agreements using this approach.

In the ATM environment, virtual networks are normally created by means of Virtual Paths (VPs). Subdivision of traffic into more homogeneous groups is also a motivation for using VP based virtual networks.

The VPN environment provided by the Tempest framework described in this paper is unique in allowing different control architectures to be operational in different VPNs. Furthermore, it is unique in the amount of user control that it allows, which makes the Tempest a safe framework for introducing network programmability.

B. *Network Programmability*

Intelligent Networks (IN) allow non-basic call functions to be associated with signalling end-points. An IN checks in a database to see if a special script, e.g. call forwarding, is associated with a dialled telephone number, and if so triggers that script. The scripts are network operator defined; since they add knowledge to the basic call model about special call types, they require extensive modification to the signalling infrastructure and can only be introduced on a long time scale. AIN (Advanced Intelligent Networks) has lessened the introduction time by making a clearer separation between the switching and controlling planes, but the nature of the interface between the two means that while adaptations of existing services, e.g. call forwarding, can easily be introduced, completely new services still require modifications to the switching plane. Moreover the time scales on which new functionality can be introduced are large: specialised knowledge is required and enormous amounts of testing are required to avoid feature interaction.

Many attempts have been made to make network nodes more programmable by adding interpreters or dynamic linkers to the node software. Programs are sent along the data path and tagged for interpretation. The network node, e.g. the switch, detects these programs and executes them. It is possible to distinguish two different motives for doing this:
- to deploy and upgrade control software at network nodes;
- to allow data streams to change the policies controlling them.

The first happens infrequently, stopping or inhibiting the normal function of the node until the transfer and loading are complete, and is normally initiated by the network operator. An example is the modification of the access control policy of an Internet firewall. Networks which have the second aim are often called *Active Networks*; [3], [15] are examples. [15] states that:

"Active Networks" are packet-switched networks in which network infrastructure is programmable and extensible and where network behaviour can be controlled on a per packet, per user or other basis. For example, a packet might carry executable code.

The execution of these active packets — or *capsules* as [3] calls them — happens on a time-scale which is at, or close to, the speed with which they are transferred across the network node. The IP model of *store and forward* has a major influence on this work. In fact, the intention of the Active Network could be summarised as modifying this model to *store, execute and forward.* Our approach, using the Tempest framework, is a pragmatic approach to the introduction of active networking: it allows soft functionality to be introduced into the control plane of the network in a fundamentally safe way, and avoids some of the nasty issues which must be faced by active networking practitioners who are working in a conventional router environment. We are actively continuing our research in this direction.

VI. Conclusions

This paper has presented a practical framework – the Tempest – in which network programmability can be realised safely and with robust resource guarantees for different, customised network functions. The programmability enabled by the Tempest exists at different levels of granularity. The Tempest allows multiple control architectures to simultaneously coexist over the same physical network, with individual resource guarantees of a nature which would allow the network operator to provide, administer and charge for Service Level Agreements. Programmability at this level allows the network operator to choose the most appropriate control architecture for the management of a given virtual private network. Indeed it reduces the activation energy for change in network services: without needing to replace the physical infrastructure it is possible to implement new services, assign them resources appropriate to the demand for their functions, and then remove them at a later date. This gives operators vastly more freedom to customise their services than is currently possible.

The Tempest's Switchlet concept provides a 'safe' environment in which third party and even dynamically loaded code can be executed without threat to other network users. Advanced and more conventional control architectures can exist together solving the problem of retaining existing network solutions while at the same creating innovative control systems for new services. Service-specific control architectures can be dedicated to particular services or distributed applications, and at a finer level of granularity, a Tempest aware control architecture may be capable of dynamically accepting and executing end-user code, allowing application-specific control.

Finally, it should be noted that although the Tempest provides the opportunity to *potentially* open up control of network resource to anyone - and thereby allows anyone to be a network operator - network owners may implement their own policies to determine who in fact is allowed to control their network. In all probability, these policies will range from liberal, in the case of research networks, to very conservative in the case of commercial networks.

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, Herbert Bos, Richard Mortier, Dave Halls, Brian Cowe and David Richerby.

VIII. Biography

Sean Rooney received the 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 PhD degree at Cambridge University. He completed his Ph.D. in 1998 and has since been working as a post-doctoral researcher at the Cambridge Computer Laboratory. His current research interests are in ATM network control and self regulating control systems.

Jacobus E van der Merwe received the B.Eng. and M.Eng. degrees from the University of Pretoria, South Africa, in 1989 and 1991 respectively. He has 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, NJ. His research interests are in network control and management, network resource management and the Internet.

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

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 PhD 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.

References

[1] ITU-T, "Recommendation I.312/Q.1201. Principals of Intelligent Network Architectures," *ITU publication*, 1992.
[2] A. Lazar, "Programming Telcommunication Networks," *IEEE Networks*, vol. 11, pp. 8–18, September/October 1997.
[3] D. Tennenhouse and D. Wetheral, "Towards an Active Network Architecture," *ACM SIGCOMM CCR*, vol. 26, pp. 5—18, Apr. 1996.
[4] 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.
[5] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Rfc2205: Resource reservation protocol (rsvp) — version 1 functional specification," September 1997.
[6] E. C. Rosen, A. Viswanathan, and R. Callon, "A proposed architecture for mpls," *Internet Draft: draft-ietf-mpls-arch-00.txt*, August 1997.
[7] K. Ramakrishnan, G. Hjalmtysson, K. van der Merwe, F. Bonomi, and S. Kumar, "Unite: An architecture for lightweight signaling in atm networks." ATM Forum Contribution 98-0520, July 1998.
[8] J. E. van der Merwe, S. Rooney, I. Leslie, and S. Crosby, "The tempest - a practical framework for network programmability," *IEEE Network*, vol. 12, May/June 1998.

Name	Latency (ms)	Environment
UNI, Q.Port/GSMP	20	Off-switch on Linux PC
UNI, Fore ASX-200WG	10	On Switch
UNI, Fore ASX-200BX	20	On Switch
UNI, ATM Forum testbed	53	Environment not explained
Xbind/GSMP	16	Off-Switch on Sparc workstation
Hollowman/GSMP	11	Off-Switch on Sparc workstation

TABLE I

SUMMARY OF RESULTS FOR THE CREATION OF PT-TO-PT CONNECTION

[9] P. Newman, “Ipsilon’s General Switch Management Protocol Specification Version 1.1,” *Internet RFC 1987*, August 1996.
[10] G. Goldszmidt and Y. Yemini, “Distributed Management by Delegation,” *Proceeding’s of the 15th International Conference on Distributed Computing Systems*, June 1995.
[11] J. T. Lewis, R. Russell, F. Toomey, B. McGurk, S. Crosby, and I. Leslie, “Practical connection admission control for ATM networks based on on-line measurements,” *Computer Communications*, 1998. Special Issue on Stochastic Analysis and Optimisation of Communication Systems (To Appear).
[12] Bellcore, *Q.Port Portable ATM Signalling Software*, 1997. Product Information.
[13] A. Battou, “Connections Establishment Latency: Measured Results,” *ATM Forum T1A1.3/96-071*, October 1996.
[14] 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.
[15] 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.

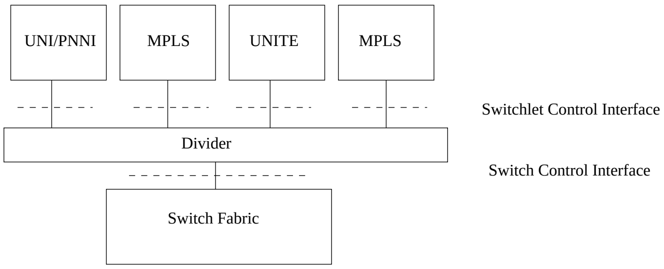


Fig. 1. Multiple control entities on a Tempest-aware switch

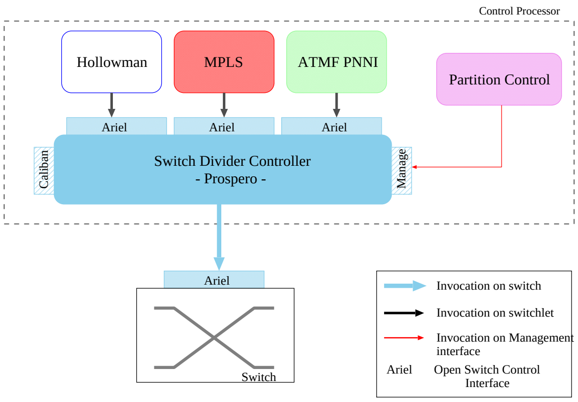


Fig. 2. Creating switchlets

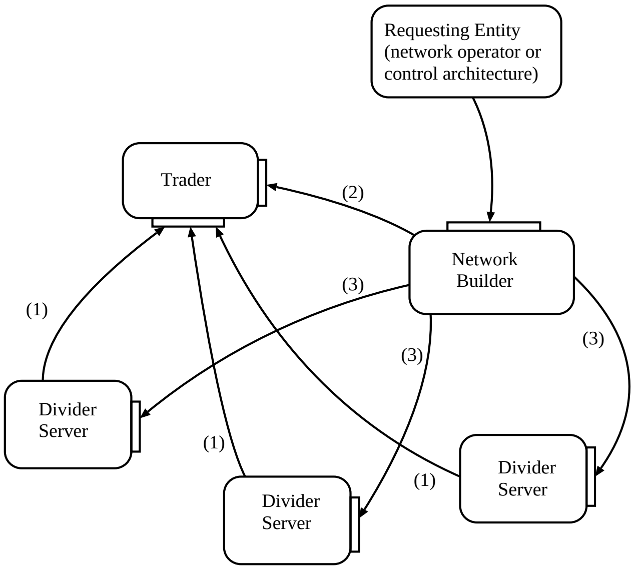


Fig. 3. Dynamic virtual network services

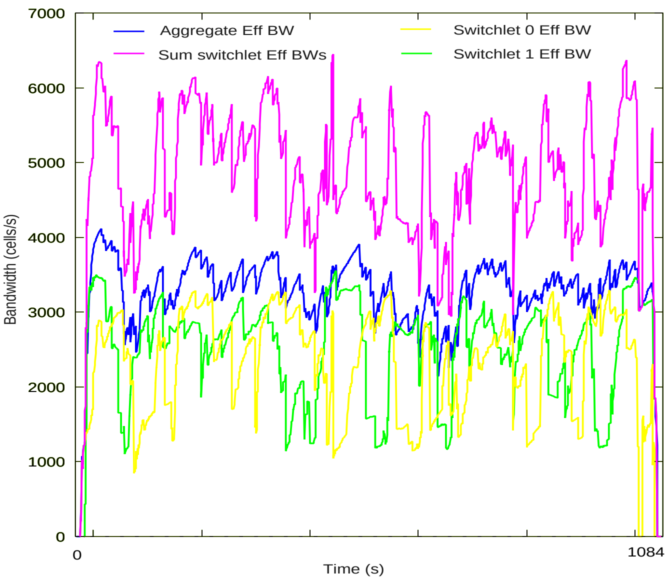


Fig. 4. Effective bandwidth estimates for individual switchlets, their sum and for the switch port as a whole

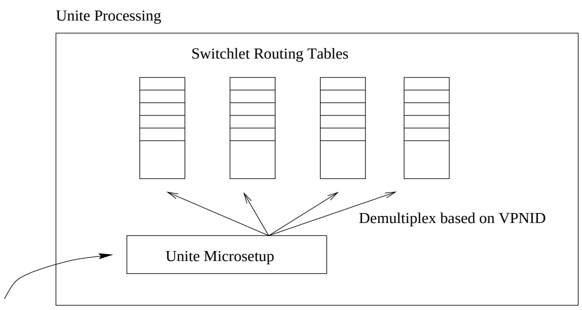


Fig. 5. Use VPNID to choose routing tables

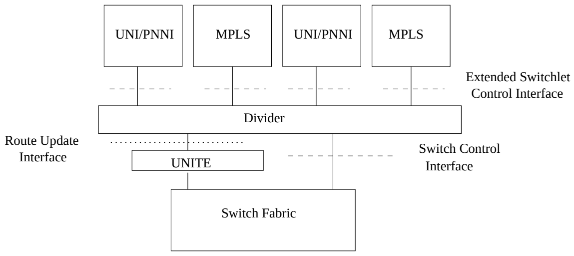


Fig. 6. Integrating UNITE in the Tempest environment

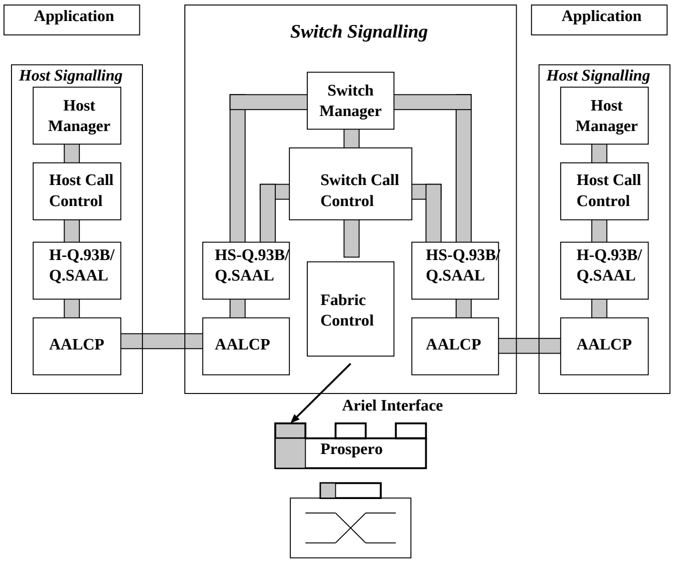


Fig. 7. Commercial ATMF control architecture in the Tempest

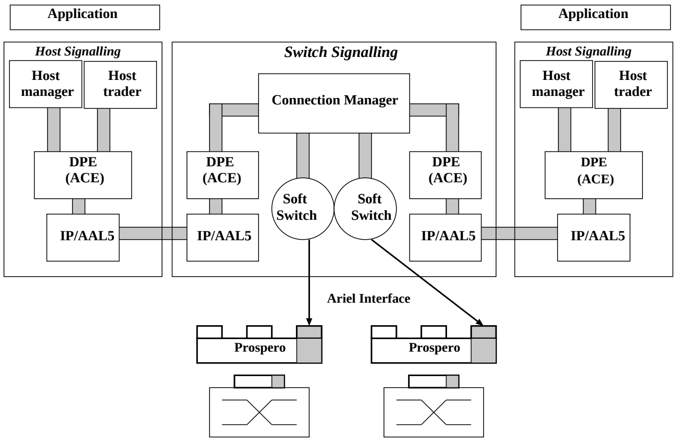
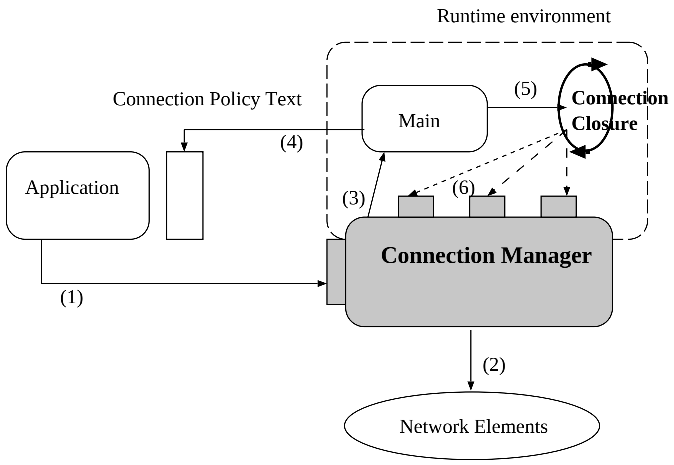


Fig. 8. Proprietary control architecture in the Tempest



(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

Fig. 9. Closure creation within the Hollowman