Vince: Vendor Independent (and Architecture Flexible) Network Control

Allison Mankin (mankin@cmf.nrl.navy.mil),
Eric Hoffman (hoffman@cmf.nrl.navy.mil) and
Maryann Perez (perez@cmf.nrl.navy.mil)

Abstract

This paper discusses experience gained while implementing Vince, a publicly available software package for the development of protocols in ATM networks. Vince provides a simple set of general abstractions which are extremely useful for protocol development and experimentation.

1 Introduction

Vince, the Vendor Independent Network Control Entity, is a publicly available program that runs in the place of vendor-provided host and switch networking code for ATM networking. Its design and software structure are made first to isolate all code specific to the hardware into a single module for that hardware, and to allow the same implementations of network protocols to be used on varied platforms. The goal of this portable design is to allow us and others to get early experience with the standards for ATM signalling, routing, traffic control, network management and so on. Vince has been developed by the authors for ARPA and the United States Navy, and it is available by anonymous FTP.

The original intent of Vince was to provide our group with networking code we could use in lieu of vendor-provided code in running a very early ATM switch network at and around Naval Research Laboratory. The evolution of Vince, while it still focuses on portable ATM control, has resulted in abstractions that embody a methodology for managing state and process in networks. These abstractions support the functioning of Vince in varied operating systems and hardware platforms, and, they support ease of prototyping varied types of network architecture and control. At this time in both high performance communications and ATM research, there are numerous networking architecture issues to be explored, such as addressing, management of traffic flows, and routing. Vince has developed as a software environment for performing this exploration.

The Vince abstractions discussed in this paper are:

- Extensible system architecture
- Atom scheduling
- Generic addressing
- Protocol stack assembly
- Copyless data buffering
- Layer bridging across disjoint operating system spaces

While the specific mechanisms and design criteria bear resemblance to that of some other systems, as will be described in each section, we believe they are synthesized in a useful and powerful way in Vince. Some drawbacks in performance and understandability result from the Vince abstractions, and we will comment on these as well.

The current implementation includes ATM cell and SAR processing with AAL 3/4 and 5, the Q.SAAL protocol SSCOP, ATM Forum 3.0 UNI signaling and ILMI[1], and RFC 1577 IP over ATM[8][11]. In the final section of this paper we describe some aspects of using Vince in a recently turned-up testbed, the Washington Area Bit Way, or Wabitway.

2 Extensible System Architecture

In developing an environment for network architecture experimentation and protocol development, the focus has been placed on moving as much of the system architecture decision as possible into a position where it can be determined at run-time.

Vince uses a module system based around a central core to facilitate this dynamic behavior. Each individual module implements the syntax and state manipulation for a particular protocol, and the core serves to provide state synchronization and operational context for the modules as a group. Both the core and the modules depend on basic services provided by libraries. The first is a portability layer, called elib, which allows module writers to depend on certain basic operations, such as memory management, regardless of the environment in which

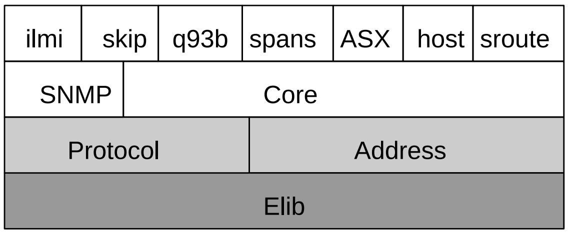

Figure 1: Software Architecture

they are running. The second is a general purpose address manipulation facility, and the third a library which allows the flexible construction of protocol stacks.

The only core currently implemented is that for ATM protocol work, but it is expected that a selection of cores could be made at compile-time. One such core we intend to develop would provide packet forwarding and scheduling as well as primitives for the maintence of IP routing information. A schematic of the modules in Vince is presented in Figure 1, the shaded areas are libraries. The core used for ATM work present abstractions such as calls, virtual connections, switching fabrics and ports, and is only about 1500 lines of C code.

The resulting system exhibits a great deal of flexibility. When Vince begins execution, each of the core components is initialized, followed by those modules which are linked into the running image. As the host module is initialized it dynamically loads the driver, where possible. The driver itself is another Vince instantiation. In the case where an embedded controller is present, yet another instantiation of Vince is created there. Between any pair of these instances that needs to have a control path, Vince inserts an interpreter with an appropriate data path. At the end of these initializations, the scheduler is invoked, and the system is configured from a series of dynamic interpreter commands.

These commands cause BSD interfaces to be created, stacks to be built for signalling, management, and routing protocols, endpoint addresses for services and terminals to assigned, and network services to be initialized.

Vince uses these interpreters to enable modules to push architecture and control decisions up to the highest level possible, and to vary the location of comparable protocol actions dynamically. The decision to use a software AAL 5 and hardware cell processing and signaling in user process space is

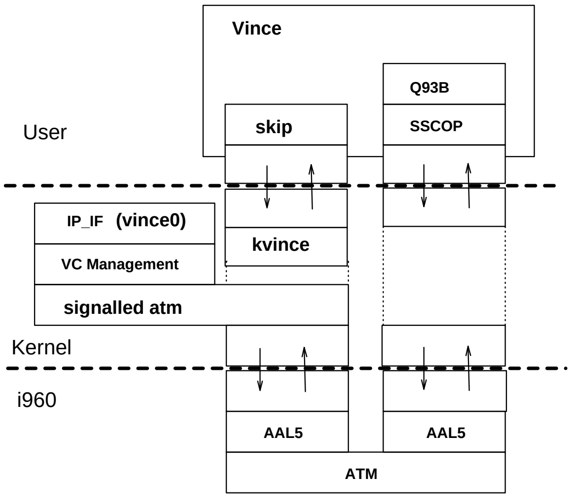

Figure 2: Protocol Bridges

one example, contrasted with a decision with the same hardware and software, and only different interpreter commands, to run the next ATM call with AAL5 and ATM from the hardware facilities and signaling run on the embedded processor. One standards based reason for this capability in Vince is to support the per-call negotiation of higher layer multiplexing and encapsulation, for instance, and other architecturally significant characteristics, that are called for in the ATM standards.

3 Atom Scheduling

Vince separates the scheduler from the rest of the system and attempts to have modules that can be operated with disparate schedulers. We thought in terms of three major types of scheduling:

1. interrupt scheduling, the coupling of external events with a specific process
2. coroutine scheduling, the negotiating of process resources among several current processes, while also waiting for events from remote entities.
3. time-based scheduling, the explicit scheduling of processes by a real-time clock, for instance in protocol timeouts.

There are a number of elegant ways of managing all these categories in one system. Examples which are usable in a C framework include semaphores, threads, and Linda tuples. All lack the platform-independence we were seeking for Vince. Of them, thread/semaphore libraries enjoy the greatest prominence, but are not ubiquitously available or portable.

In Vince each compound action is broken in pieces, each break occurring where some scheduling decision, or asynchronicity takes place. Most often these pieces, or scheduling atoms, represent event handlers, or a form of explicit continuations. These atoms can be assembled together under a given scheduling system, but does not rely on any particular system being in place.

A prototyping advantage in coding modules this way is that decisions concerning the scheduling of these atoms can take place at whatever point in the implementation they are most appropriate, often closer to the application or end-user level than is usually easy in protocol software architectures.

Another advantage is that the atoms of code produced to exist in the scheduling discipline usually consist of only the most simple control structures, and usually correspond to relatively simple, easy to understand actions. Unfortunately, this structure decreases the understandability of the global program control for someone examining the individual code segments.

This methodology has an efficiency penalty in C. Processes are reentrant and are expressed in terms of many small functions. Even simple behaviors may require many function calls to be performed. A single external event can trigger a chain of reactions resulting in an extremely deep stack. Additionally, significant heap memory is needed to preserve the state of the all these invocations of small procedures. This could be addressed by the application of appropriate continuation based compiler technology [2].

4 Generic Addressing

A fundamental need in networking is an identification used for describing and communicating with network entities, interfaces, services and applications. It is not unusual for software designed as common code among a disparate set of protocols to have a generic address abstraction and support facilities, for example the sockaddrs in BSD Unix [9]. While this simple idea allows different protocol families to exist within the same framework, it does not facilitate their interoperation. Specifically, there is no general ability to map system endpoint addresses, protocol multiplexing codepoints, and service addresses from the context of one protocol family into another.

The Vince address mapping facility allows the registration of general address translation functions in a global table, making them accessible through a common interface. Specifically, this facility can be used by a protocol encoder which wishes to express a generic address in its native format.

In the architectural experiments for which Vince is designed, the ability to do this mapping is used for operating trials of different types of address hierarchy. We also envision times when the translation will help us run a piecemeal transition of signaling protocols in our testbed. This is another useful prototyping capability. Translation is used to implement group addressing, address resolution and routing protocol interworking as well, in ways that we describe in following sections.

4.1 Group Addressing

One fortunate side effect of a generic address facility is that semantics which are independent of the particular address family being used can be developed. The most interesting example of this to date is the specification of group addresses. Vince allows multicast call setup, packet forwarding and routing protocols to work with multicast addresses of any family. The translation mechanism allows each family to provide arbitrary mappings from parts of its native space into the general group space.

4.2 Address Resolution

Protocols such as ARP [12] in which many families can be encoded also benefit from Vince's generic specification of addresses, for obvious reasons. More subtly, the generic addressing in Vince has enabled us to develop address resolution facilities and the specific ARP protocols in a mutually independent fashion, allowing experimentation. The mechanism is as follows: an address resolution protocol, such as ATMARP [8], registers itself as a general functional translation. Any request for a translation causes a query to be sent. If the requester wishes to admit the possibility of nonlocal translation information, it registers an event handler which will be called when a specific address binding is put in place. The global translation table takes the place of an arp cache, and the address bindings made are usable by any module in the system.

4.3 Protocol Interworking

A more experimental use of the generic addressing and translation facilities in Vince is to support interworking of protocols that use differing or multiple sets of addresses. In routing, the use of multiple addresses depending on routing needs is well supported (and the multiple addresses might be those produced by a series of masks on one prefix).

A valuable example of using flexible mappings between addresses in the current implementation is the ability of the Vince IP call management code to place ATM calls to IP system endpoints without needing to know the addressing scheme used in the ATM network. Similarly, the ATM signalling layer need not know the details of application level addressing. The signalling call setup code simply asks for a translation from the specified endpoint address, such as IP, into the native family of the signalling protocol, such as ATM Forum NSAP addresses. This decoupling allows services such as IP and signalling protocols to be developed in isolation and interwork easily.

We realize that address translation as a mechanism does not translate easily to large-scale or operational use for interworking, since it does not inherently ensure that translations exist when they are needed, or provide an ability to map complex address semantics from one family into another.

5 Protocol Stack Assembly

A basic component of Vince that was crucial to develop at an early stage was the interface for packet transfer across the protocol processing layers. Vince realizes a general purpose stack structure where small general purpose layers can be written in isolation and assembled dynamically. Logging and digital signing layers are two examples of small layers that can be placed anywhere in the protocol stack. Unlike STREAMS [13], which attempts to generalize all interlayer communication with the use of priority queues and out of band control messages, resulting in an extremely complicated library interface, the protocol stacks used in Vince support general semantics for data movement, but use ad hoc call semantics as the control mechanism.

Layers are expected to conform to a minimal set of control semantics, most of which are mediated through the use of protocol flags. For any instance of a layer, for example, the AAL5 instance associated with a particular ATM virtual connection, there can be no communication downward until the connected flag has been set on an upcall from the instances of the layers below (e.g. ATM and SSCOP). No write downs for an instantiation of some stack can occur after a disconnected flag is set from below. No write ups can occur after disconnected is set from above (this is also analogous to mechanisms in BSD sockets). Upon the receipt of a shutdown from either direction, the layer immediately frees all resources associated with stack instance and forward the flag in the correct direction. The connect/disconnect mechanism allows a general notification of whether or not a stack is fully assembled and ready for data transmission. Connection oriented protocols (again, e.g. ATM call setup) begin to negotiate a connection with the remote peer upon receiving connected status from the lower layer, and forward the connected flag to the upper layer only when the connection has been established with the peer. This allows each layer to be written outside the context of whether or not layers in the same stack are connection oriented or not, or what their particular state might be.

6 Copyless Buffering

While it was considered extremely important that individual protocol implementations in a stack be thoroughly isolated from the data layouts enforced by the layers above and below it, explicit copying of data into properly laid out buffers at each layer was considered prohibitively expensive. A simple and flexible buffering strategy was developed in order to make this possible. Other existing systems [9] [13] [5] [6] use buffer chains to allow protocol elements to attach and detach headers and trailers. One important drawback to a simple chain is that copying is often necessary in order to align and pad parts of the buffer for use by the protocol process.

The basic notion of a buffer which spans the entire stack is that each layer performing processing on the buffer will have a local context in which the buffer appears exactly as it should for its processing to be performed. Operations in this layer-local context use standard functions to operate on the buffers which take care of translating local offsets into actual memory locations.

The resulting buffer is a tree of fragments, and each layer has a pointer into the first byte allocated to its data.This tree is completely determined by the assembled stack in which it was allocated and its size. An entrypoint is registered upon the creation of any particular layer to provide buffer allocation for the layer above it. Each layer in turn sees a request for a buffer of a particular size, makes decisions concerning padding and the allocation of head-

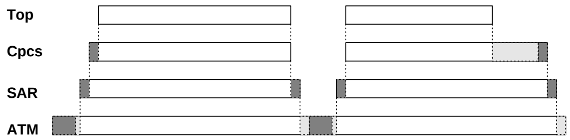

Figure 3: Example Buffer Layout

ers, trailers, or individual segments, and requests an allocation from the layer beneath it. As each layer writes in the relevant protocol fields, they are placed in real memory as they would need to be for the message to be sent out the physical interface. While in general this approach should provide a good balance between abstraction and efficiency, this has not yet been borne out by experience. Due to the high number of very small layers and very small fragment sizes in the existing implementation, benefits gained from not copying at the layer boundaries are outweighed by the heavy impact on the memory management system, and the number of function calls required. Another major efficiency problem is the use of functions to write into and read out of buffers. As in the BSD IP implementation, for notational convenience, structures are used to describe header contents, and 'pullups' must be used to read out of the general, unaligned buffer into a format which is convenient to process. While the alignment issues remain problematic, integration of the buffer mechanism into the compiler would cut down the run time overhead considerably and allow for more readable protocol code. A great deal of efficiency could also be gained by managing a resizable pool of buffers of a fixed set of sizes, such as powers of two.

7 Bridge Layers

One important function of Vince protocol stacks is to abstract away information as to where each layer will execute from the layer itself. Based on administrative decision, parts of the protocol processing will run in different parts of one tightly coupled system. Two examples of this are the use of an offboard processor associated with a particular network interface, or protocol processing which takes place in both user and kernel space in a Unix system.

In order to effect this, layers referred to as 'bridges' are constructed. A bridge passes data bidirectionally across the interface, preserving the generalized protocol stack semantics in both directions.

Doing this in an efficient manner requires careful consideration of the buffer management on both sides of the interface. The details of any given instance of a bridge layer are perhaps beyond the scope of this brief paper. But as an example, consider the small layer that we use on SunOS Unix between SSCOP in a user space Vince process and AAL5 in the Unix kernel. All specifics of passing a packet between these two spaces, and potentially, not even copying it across the boundary, are contained in the bridge layer module and are not known to modules in either space [10].

8 Current Implementation Status

Vince is currently implemented as a single core which maintains a set of structures and state which are relevant to ATM networks, such as calls, switches, ports, signalling and connection routing protocols, and ATM hardware components. The portability layer allows components to be run in process space on several commercially available Unix systems, specifically SunOS, IRIX, HPux, Ultrix, AIX, and Linux, as well as BSD kernel space, and in a standalone mode on an embedded i960 processor.

Protocol modules exist for the Fore Systems Spans signalling protocol, the ATM Forum's UNI 3.0 signalling protocol, Q93b, an SNMP version 1 module, an RFC 1577 ATMARP module, and simple distance vector VC routing protocol.

There is a virtual hardware module which uses the layer facility to simulate an ATM switching fabric over UDP. Hardware modules in Vince's platform portability layering scheme currently include modules to run on Fore Systems' ASX100 switching hardware and Fore Systems' host interface hardware. Hardware modules for other vendors' products are under way, both at NRL and also at several other organizations.

9 Vince Wabitway Support

As of April 1994, the Washington Area Bitway comprises seven backbone sites at six government agencies and a telephone provider, located throughout the greater DC area and linked by OC-48 SONET. The SONET is provisioned to offer multiple OC-3 links among the sites and ATM switches initially from Fore Systems. Other ATM switches are to be added as we or vendors do Vince ports or as the vendors begin to offer ATM Forum and IETF standard protocols on them. The testbed particularly supports early trials of routing, addressing and service multiplexing because it is straightforward to vary its logical topology at the SONET layer.

We plan to have an opportunity to further validate the Vince architecture by using it to develop more of the protocols of the IP suite in support of the Wabitway. One implementation we believe will be fruitful and interesting is the exploration of mapping soft state IP flows as supported by RSVP [14] [3] to hard state QOS ATM connections and the experimentation with routing approaches in the context of this soft state. The Wabitway application set is potentially as varied as the founding agencies (sensor integration, simulation, visualization, and supercomputer interconnections are all proposed), and this increases the incentive for our group to offer extended network control with Vince protocols.

10 Conclusion

While the operating system and software issues involved in protocol implementation are complex, use of a few simple abstractions enable protocol development while enhancing portability.

As a software architecture, Vince allows the development of disparate protocols within the same framework without violating natural abstraction boundaries, thus permitting an examination of their behavioral syntheses.

References

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

[2] Andrew W. Appel, *Compiling with Continuations*, Cambridge University Press, Cambridge, 1992

[3] C. Brazdziunas, "IPng Support for ATM Services", work in progress

[4] R. H. Campbell, S. M. Dorward, A. Iyengar, C. R. Kalmanek, G. J. Murakami, R. Sethi, C.-K. Shieh, and S.-M. Tan *Control Software for Virtual-Circuit Switches*, Springer-Verlag

[5] D.D. Clark and D.L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", Proceedings of SIGCOMM '90 Symposium, September 1990

[6] Peter Druschel, Larry L. Peterson, "Fbufs: A High-Bandwidth Cross-Domain Transfer Facility", SOSP 14, December 1993

[7] N.C. Hutchinson, L.L. Peterson, "Design of the X-Kernel, Proceedings of SIGCOMM '88 Symposium", August 1988

[8] M. Laubach, M., "Classical IP and ARP over ATM", RFC1577, Hewlett-Packard Laboratories, December 1993

[9] S. J. Leffler, M. J. McKusick, M. J. Karels, J. S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operation System*, Addison-Wesley 1989.

[10] C. Maeda, B. Bershad, "Protocol Service Decomposition for High-Performance Netoworking", SOSP 14, December 1993

[11] M. Perez, D. Grossman, F. Liaw, A. Mankin, E. Hoffman, A. Malis, "ATM Signaling Support for IP over ATM", work in progress

[12] D. Plummer, "An Ethernet Address Resolution Protocol", RFC826, November 1982

[13] D.M. Ritchie, "A Stream Input-Output System", AT&T Technical Journal, Vol 63, No. 8, Part 2, October 1984

[14] L. Zhang, B. Braden, D. Estrin, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) Version 1 Functional Specification", work in progress

Author Information

Eric Hoffman began working at Naval Research Lab (NRL) in Washington D.C. in 1987 with parallel algorithms, and has been the principle developer of Vince since its inception in 1992.

Allison Mankin came to NRL in 1993. She is involved in design and planning for Vince. She has published research on congestion avoidance and multimedia integrated networking, based on work using the DARTnet and BLANCA testbeds, as well as NRL's testbed. She is on the Internet Engineering Steering Group, serving as Area Director for Transport and Co-Director for IP Next Generation.

Maryann Perez started work at NRL in 1993. She has been involved in the development of ATM signaling software for Vince. She received a B.S. in 1989 and an M.S. degree in Computer Science in 1991 from Purdue University. For two years she worked at the MITRE Corporation where she did research in the area of ATM network performance.