Introducing PRouST:
The PNNI Routing and Simulation Toolkit

Bilal Khan, David Talmage, Sean Mountcastle, Abdella Battou, Spencer Marsh [*][†]

September 14, 2003

Abstract

Despite the increased availability of affordable ATM hardware in recent years, the scientific research community has often found it difficult to engage in much needed basic research in the areas of ATM protocol design and network performance optimization. We believe that one major cause for this has been the absence of affordable, publicly available source-level implementations of the ATM switch protocol stack. The PNNI Routing and Simulation Toolkit (PRouST) attempts to remedy this. PRouST is a freely distributed, extensible environment for research and development in ATM switch signaling and routing. PRouST includes a complete source-level release of the ATM switch PNNI protocol stack, conformant to version 1.0 of the ATM Forum specification [1]. It is our hope that PRouST will serve as the starting point for bold new initiatives in research and development for ATM technologies. In this paper, we describe the design of PRouST and the features it supports.

1 Introduction

PRouST is used both to simulate networks of ATM switches, and to emulate ATM switches on hosts that reside on an ATM network. In fact, PRouST permits mixing simulated networks with emulated switches and real hardware switches. Each PRouST switch can specify its own custom policies for call admission, path selection, and topology aggregation using a plug-in mechanism. PRouST provides researchers with a language called FATE, which can be used to to specify network scenarios for simulatation. A researcher can conduct PRouST simulation experiments by registering their his/her own custom Investigator objects with PRouST. PRouST reports all network events of interest to registered Investigators for analysis. At present, FATE cannot be used to describe mixed simulated-emulated configurations. Instead, PRouST provides C++ libraries enabling programmers to construct such systems. The libraries facilitate the instantiation of switches, their interconnection, and their attachment to special adapter objects which translate between packets on a network interface card and PRouST's internal data objects.

2 PRouST Software Architecture

PRouST switches are the software analogues of hardware ATM switches. As such, a PRouST switch has a switch fabric, physical ports for connecting to the outside world, a controlling authority that makes decisions in response to the state of the network as perceived by the switch, and programmatic means for external monitoring of switch state. PRouST was designed and implemented using the Component Architecture for Simulating Network Objects (CASiNO). CASiNO is a framework for rapid prototyping of com-

[*]Bilal Khan, David Talmage, and Sean Mountcastle are with ITT Industries, Advanced Engineering & Sciences, Advanced Technology Group, at the Center for Computational Sciences of the Naval Research Laboratory, Washington D.C.

[†]Abdella Battou and Spencer Marsh are with Princeton Optical Systems, Greenbelt MD.

munication protocol stacks and network simulators [3, 6]. In the next section we give a brief introduction to CASiNO components; then we shall be able to describe PRouST succinctly in terms of CASiNO component terminology.

2.1 The CASiNO Framework

The CASiNO framework implements a rich, modular, coarse-grained dataflow architecture, with an interface to a reactor kernel that manages the application's handlers for asynchronous I/O, real timers, and custom interrupts.

The **Reactor Kernel** is the name of the collection of classes that provide applications with access to asynchronous events, such as the setting and expiration of timers, notification of data arrival, and delivery and receipt of intra-application interrupts. CASiNO can be used to implement both live network protocols as well as their simulations; both require precise coordination of program execution relative to some notion of time. In the case of network simulation, the notion of time need not remain in lockstep with the real wall clock time. The Reactor Kernel can switch between these paradigms transparently, and user program semantics are not affected by whether they are operating in simulated or "live" mode.

The **Data-Flow Architecture** is the name of the collection of classes that define the behavior of the modules within a protocol stack. CASiNO has three tiers in its design:

(i) The Conduit level of abstraction represents building blocks of the protocol stack; the function of the Conduit class is to provide a uniform interface for joining these blocks together and injecting data (Visitors) into the resulting network of Conduits. Every Conduit has an A-side and a B-side.

(ii) The Behavior level of abstraction represents broad classes, each capturing different semantics regarding the Conduit's purpose and activity.

(iii) The Actor level of abstraction represents the user-defined functionality of a Behavior. Behaviors are made concrete when the user specifies a custom Actor instance with which the Behavior is composed.

There are five abstract subclasses of the Actor class: Terminal, Accessor, State, Creator, and Expander. These subclasses of the Actor correspond respectively to the Behavior subclasses: Adapter, Mux, Protocol, Factory, and Cluster. We now describe each of these Behaviors and their associated Actors.

1. **Protocol Behavior - State Actor**

   A communication protocol is often specified by a finite state machine. A Protocol behavior implements a finite state machine using a specialized State Actor. When a Visitor arrives at a Protocol, it is handed to the State actor. The State then queries the Visitor to dynamically determine its type. Once the State determines the type of the Visitor, the State transforms the Visitor into the appropriate specialization by typecasting and operates on the Visitor accordingly. The State Actor is implemented according to the state pattern defined in [5]. Protocols are typically represented as rectangles or cylinders labelled with the name of the State.

2. **Adapter Behavior - Terminal Actor**

   The Adapter Behavior converts or adapts outside world information into instances of Visitors that can be interpreted by the CASiNO application. Adapters are used for testing (e.g. when a protocol stack is capped at both ends with two Adapters—one to inject Visitors and the other to sink them) and to perform conversion between different types of Visitors (e.g. between two networks of Conduits which are are expecting different types of Visitors). The *Terminal* actor is used to configure the Adapter, and to specify how Visitors are to be injected and absorbed. Adapters are typically represented as squares labelled with the name of the Terminal.

3. **Mux Behavior - Accessor Actor**

   The Conduit with a Mux Behavior allows developers to implement multiplexing and demultiplexing of Visitors. Demultiplexing occurs when

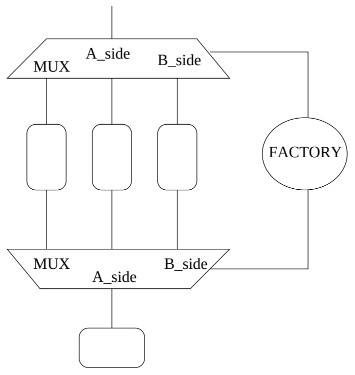

Figure 1: Common Usage of two Muxes and a Factory

a Visitor arrives and needs to be routed to one of the several other Conduits on the basis of some aspect of the Visitor's contents; we refer to this as the Visitor's key or address. A Mux is specified by composition with a suitable Accessor actor. The Accessor Actor is responsible for examining the incoming Visitor's contents and deciding to which Conduit the Visitor should be sent. Muxes are typically represented as trapezoids labelled with the key that Visitors are required to provide in order to be demultiplexed.

4. **Factory Behavior - Creator Actor**

   The Factory Conduit allows dynamic instantiation of new Conduits. A Factory Behavior is typically found together with Muxes in a configuration such as the one depicted in figure 1. A Conduit configured with Factory Behavior is connected to the B sides of two Conduits with Mux Behavior. Upon receipt of a Visitor, the Factory may make a new Conduit. If it decides to do this, the Factory registers the newly instantiated Conduits in the Accessor of two Muxes. In this manner, Visitors can be sent to a Mux in order to install new Conduits dynamically, provided that a Factory is attached to the Mux. The Factory behavior is specified by composition with a Creator Actor. After the installation of the newly created Conduit object, the incoming Visitor which prompted the installation, is handed to the newly instantiated Conduit. Factories are typically represented as circles which are labelled with the type of Conduit that their Creator instantiates.

5. **Cluster Behavior - Expander Actor**

   In addition to the four primitive Behaviors of Protocol, Factory, Mux, and Adapter, CASiNO provides generic extensibility via the Cluster Behavior. Just as the primitive behaviors mentioned above are configured with a particular Actor (i.e. a concrete State, Creator, Accessor, or Terminal object), the programmer can specify a Cluster's behavior by composition with a concrete Expander Actor. An Expander may be thought of as a black-box abstraction of a network of Conduits. Clusters are typically represented as dashed rectangles labelled with the name of the Expander.

In terms of CASiNO framework components, the high-level structure of a PRouST switch can be expressed by the schematic of fgure 2. In the subsequent sections, we will dissect this picture and view it at higher magnifications.

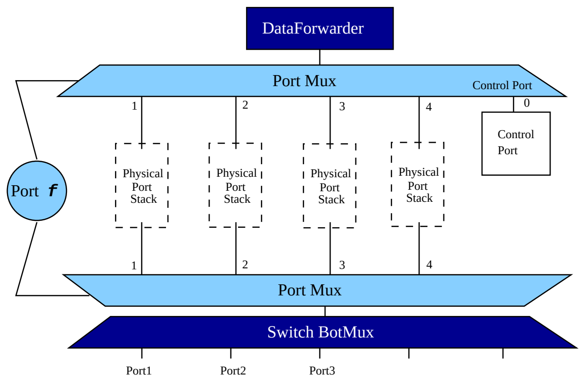

Figure 2: A switch with 4 physical ports and a control port

Like any CASiNO application, the data that flows through a PRouST switch is encapsulated in Visitor [5] objects. PRouST uses specialized Q93bVisitors to carry signaling information, PNNIVisitors to carry routing information, FastUNIVisitors to carry intra-switch messages, and DataVisitors to carry non-control information.

Emulated switches are constructed by placing an adapter below the SwitchBotMux. The adapter converts any incoming network messages arriving on the network interface card into the appropriate type of PRouST Visitors. This adapter also performs the reverse conversion, converting outgoing PRouST Visitors into network PDUs.

2.2 The DataForwarder

The DataForwarder (depicted at the top of the schematic of figure 2) represents the connection fabric of a PRouST switch. The DataForwarder maintains a table of all of the bindings between incoming port number and virtual circuit (VC) and outgoing port number and VC. When a DataVisitor flows into the DataForwarder, it is queried about its *in-port* — the port on which it arrived, and its *in-VCI*— the VC on which it is traveling. The DataForwarder then directs the Visitor to the appropriate output port and VC, based on the binding information in its table. When control-related Visitors (e.g. Q93bVisitors, FastUNIVisitors or PNNIVisitors) enter the DataForwarder from one of the Physical Ports, they are routed to the Control Port for processing. When control-related Visitors enter the DataForwarder from the Control Port, they have already been stamped with their intended egress port; the DataForwarder then simply routes them to this port.

2.3 Physical Ports

The location of the Physical Port modules is sketched in figure 2; the internal structure of a Physical Port is shown in figure 3. A Physical Port has several constituents:

1. A Physical Port operates one instance of the *Physical Hello* Protocol Conduit [1, p62], which is used to monitor the state of the (perhaps simulated) physical link attached to that port of the switch.

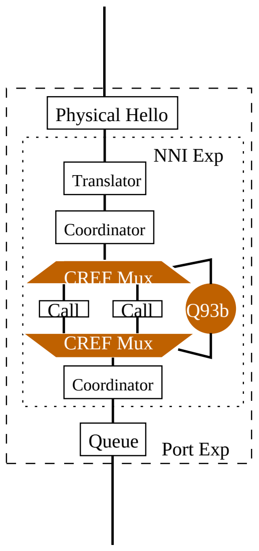

Figure 3: Architecture of a physical port stack

2. The the NNI Expander, which contains one instance of the NNI finite state machine [1, pp39,173] per Switched Virtual Circuit (SVC) that has been (or is being) established through this port. Inside the NNI Expander Conduit
   - The *Q93bCall* object implements one instance of the NNI finite state machine. Note that a call that passes through a PRouST switch will have two Q93bCall instances (one in the NNI Expander of the incoming Physical port, and one in the NNI Expander of the outgoing Physical port).
   - The *CREF muxes* direct Visitors to the appropriate Q93bCall objects based on the "Call Reference (CREF) Value" reported by the Visitor. If the CREF reported by the Visitor is not the key of any Q93bCall object, or if the Visitor is carrying a Setup message, the CREF Mux sends the Visitor to the *Q93bFactory*. The Q93bFactory is responsible for deciding whether to instantiate a new Q93bCall object and attach it

between the CREF muxes or to pass it upward uninterpreted.

- The topmost Conduit in the NNI Expander is the *Translator*, which is an adapter to convert Q93bVisitors (which are processed by the Q93bCall finite state machine) into FastUNIVisitors (which are processed by the internal components of the switch). The FastUNIVisitor is an augmented class that is used to communicate generic events internally within a PRouST switch. Similarly, when a FastUNIVisitor arrives at the Translator from above, it is converted into a Q93bVisitor before being sent downwards.
- The *Coordinator* Conduits flank the Q93b (NNI) Finite state machines and implement the "Coordination Protocol" as per the PNNI specification.

3. The *Queue* Conduit is a buffer for incoming and outgoing Visitors. It is serviced at a user-specified rate.

2.4 Control Port

The Control Port (depicted in the schematic of figure 2) is shown in greater detail in figure 4. The Control Port is often referred to as Port 0 of the PRouST switch; it contains Conduits which manage call admission, path selection, peer group leader elections, aggregation, and the network database for the switch. Call admission, path selection, and aggregation components are aided by policies which can be specified using a plug-in mechanism. PRouST provides reasonable default policies and a library of objects for users to craft their own.

As per the PNNI specification, a PRouST switch may find i necessary to operate concurrently at many levels of the PNNI hierarchy—the precise number of "logical instances" within the switch depends on the outcome of Peer Group Leader elections within its Peer Group at each level. Control messages that are sent to a *specific level* of a PRouST switch must specify this level in the selector byte of the message destination NSAP address.

Internally, the Control Port is split into two parts: The *instance-dependent* part where components are isolated according to the level of the hierarchy at which they operate, and the *instance-independent* part, where components operate "at all levels" concurrently.

I. The *instance-dependent* part consists of:

- A *Physical Node Expander*, containing a set of NodePeer Protocol Conduits, one for each adjacency established by the switch *at the physical level.* A NodePeer Protocol Conduit cooperates with a peer Protocol Conduit in the adjacent switch; together they function to synchronize relevant information in the routing databases of their respective switches. When such synchronization is completed (or fails), the NodePeer Conduits indicate this to other Conduits in the Control Port by sending them NPUpVisitors (or NPDownVisitors). The Physical Node Expander also contains an election finite state machine, which operates at the physical level to determine if the switch should have a logical instance.
- A *Logical Group Node (LGN) factory* which instantiates and installs Logical Group Node objects whenever this switch wins an election. These Logical node instances are installed between
- a *pair of Muxes*, which route Visitors to the appropriate Physical/Logical Node instance
- Zero or more *Logical Node Expander* instances. There is one Logical Node instance for each level of the PNNI hierarchy at which the switch has won an election and is the peer group leader. Each logical instance contains:
  - A set of LogicalHello Protocol Conduits, one for each higher-level neighbor.
  - A set of Logical NodePeer Conduits, one for each higher-level adjacency.
  - An election finite state machine, which operates at that particular logical level to determine if the switch should have a higher logical instance.

II. The instance-independent part of the Control Port consists of the ResourceManager Expander. Within this Expander lies a stack of six closely cooperating Conduits.

- The *Actual Call Admission Control* (ACAC) Protocol Conduit handles all of the call admission control and the (re-)advertising of physical links. When a NodePeer Protocol reports that synchronization with an adjacent peer has completed, ACAC originates a Horizontal Link PTSE and begins managing bandwidth for the link. ACAC also advertises the disappearance of these links, when reported by the NodePeer Protocol. When a call setup message arrives at ACAC, it delegates to the ACACPolicy to determine if the call should be accepted; the ACACPolicy can be customized by the user.

- The *RouteControl* is a Protocol Conduit which serves as a cache for pending connection setups that either recently originated at the switch or recently entered the peer group at the switch. If crankback should occur for these setups, RouteControl will work with the Logos module to generate a new candidate route for the setup. Once a connection has been successfully established, RouteControl flushes the information about it from its internal tables.

- The *Logos* is a Protocol Conduit which maintains information about the topology of the network in the form of a directed graph, with symbolic information associated with each node and edge. Logos computes routes for connection setup requests by delegating to a BaseLogos plug-in, which can be customized by the user.

- The *Leadership* is a Protocol Conduit which collects information about election results from all election finite state machines (located in Physical and Logical Node instances). If a switch loses elections at some level (at which it was previously the leader) then the Leadership Conduit is responsible for destroying the Logical Instances of the switch corresponding to those levels at which the elections were lost. The Leadership Conduit is also responsible for responding to election victory including such matters as: originating Nodal Information Groups, extending the Nodal Hierarchy Lists, augmenting the Next Higher Level Bindings, informing the Aggregator that complex node representations at the higher level are required, etc.

- The *Aggregator* is a Protocol Conduit that is responsible for originating higher level horizontal links and uplinks corresponding to aggregated lower level horizontal links and uplinks. Additionally, the Aggregator maintains the Complex node representation for the logical group node, based on the peer group topology at the level of the corresponding peer group leader. These computations are implemented in part by delegation to the AggregationPolicy plug-in, which can be customized by the user.

- The *Database* is a Protocol Conduit that holds all the PNNI Topology State Elements (PTSEs) received by this switch. The Database responds to requests from the NodePeer finite state machines at each level, providing them with the appropriate information for synchronization with their peers. Information in the Database is "sorted" by the level of the hierarchy at which the information was received. PNNI's downward flooding policies are implicit in the design of the Database Conduit: higher level NodePeer FSMs are restricted from seeing PTSEs that were received by lower-level instances of the switch. The Database is also responsible for PTSE expiry, and for originating inter-Peergroup Leader connection setup requests in response to receipt of uplink PTSEs.

3 Experiments with PRouST

3.1 FATE Language

FATE will be described in detail in the final version of this paper.

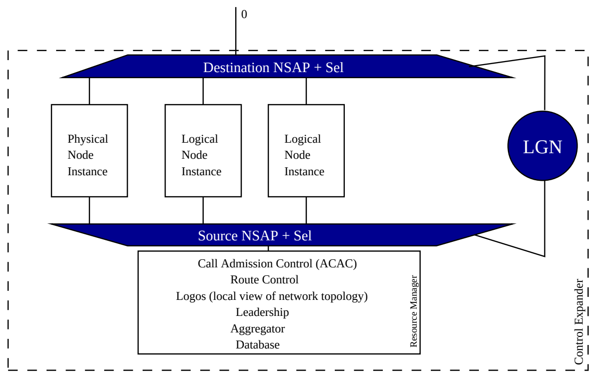

Figure 4: Architecture of the control port

3.2 Plug-in Policies

PRouST supports custom plug-in policies for call admission, path selection, and aggregation. A plug-in policy is derived from one of three policy base classes included with PRouST: ACACPolicy for call admission, AggregationPolicy for aggregation, and BaseLogos for path selection.

- All call admission plug-ins must make their decisions based on local bandwidth available on links incident to the switch ports. If a policy admits a call, it must also decide if it is necessary to readvertise the link due to "significant change". The policy may use all information elements in the setup message.

- Aggregation policies aggregate lower-level horizontal links and uplinks into higher-level ones. For nodes that are peer group leaders, they also summarize lower-level topology into "complex node representations" for use by higher-level peer groups. Aggregation policies have access to resource availability of the links to be aggregagated and the graph representation of the state of the peergroup. This graph is assembled from the PNNI Topology State Elements (PTSEs) known to the switch at the time of the decision.

- Path selection policies determine the route by which a setup message traverses the peer group. The policy has access to the entire Q93b setup message and the graph representation of the state of the network; The latter is assembled from PNNI Topology State Elements (PTSEs) known to the switch at the policy is consulted.

3.3 NetStatsCollector and the Investigators

During the course of a simulation, the global NetStatsCollector module of PRouST gathers reports of all high-level PNNI events. These events are posted by all PRouST finite state machines and Conduits. A partial list of these events is given in table 1. The NetStatsCollector provides a mechanism for registration of custom Investigator objects. Each Investigator can specify the types of events that it is interested in, and the criteria under which it should be activated. The NetStatsCollector reports all relevant PNNI events to registered Investigators. Investigators can then consult and examine the NetStatsCollector's event history and write reports of their findings.

3.4 Experiments with PRouST

PRouST has been used to conduct several experiments. For example, in [2], we defined new performance indicators based on the time and traffic required for the PNNI protocol to first enter and subsequently return to the meta-stable state of *global synchrony*, in which switch views are in concordance with physical reality. We found that high call admission rate and low setup latency were related to these indicators. We used PRouST to conduct simulations of PNNI networks, with the aim of discovering how topological characteristics, such as the diameter, representation size, and geodesic structure of a network, affect its performance with respect to these new indicators.

Others have also used PRouST to conduct experiments. Recently, J. Turner and D. Wu at the Applied Research Laboratory at Washington University successfully combined PRouST with their Gigabit switch controller to form a network node [7]. They have developed several applications with one network node (running modified PRouST with GBNSC, their switch controller) and two end-systems running modified SEAN.

More information about PRouST can be found in [4].

3.5 Acknowledgements

We thank Sandeep Bhatt for his help in early stages of PRouST development, particularly in the development of the NodePeer FSM and the Database. We are grateful to Rich Verjinski, who helped us test interoperability of PRouST with FORE ATM switches, and to Dardo Kleiner, who proofread many drafts of this paper.

References

[1] ATM-Forum. *Private Network-Network Interface Specification, Version 1.0.* ATM-Forum, 1996.

[2] A. Battou, B. Khan, and S. Mountcastle. PNNI and the Optimal Design of High-speed ATM Networks. Journal of Informatica, January 2000.

[3] N. R. L. Center for Computational Science. CCS Series on ATM Internals Volume 0: The Devlopment Framework, CASiNO. `http://www.nrl.navy.mil/ccs/project/public/casino/`.

[4] N. R. L. Center for Computational Science. PROUST: PNNI Routing and Simulation Toolkit. `http://www.nrl.navy.mil/ccs/project/public/proust/PRouST-dev.html`.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley Professional Computing Series, April, 1997.

[6] S. Mouncastle, D. Talmage, S. Marsh, B. Khan, A. Battou, and D. C. Lee. CASiNO: A component architecture for simulating network objects. Proceedings of 1999 Symposium on Performance Evaluation of Computer and Telecommunication Systems, (Chicago, IL), pp. 261–272, Society for Computer Simulation International, July 1999.

[7] J. Turner and D. Wu. Washington University's Gigabit Network Technology Distribution Program. `http://www.arl.wustl.edu/gigabitkits/kits.html`.

Table 1: The NetStatsCollector reports these PNNI events

Event Type	Meaning
Call_Submission	A call setup request has been submitted at a switch
Call_Arrival	A call setup has reached and was accepted at its destination
Call_Admission	A call has been admitted by ACAC at a switch
Call_Rejection	A call has been rejected by ACAC at a switch
DTL_Expansion	The route and designated transit list for an ATM setup message has been computed at the border node entry point of a peergroup
Crankback	Crankback has occurred
Start_Elections	A switch has started participating in elections
I_am_PGL	A switch believes it is peergroup leader
Voted_Out	A switch that was peergroup leader is no longer peergroup leader
Lost_Election	A switch believes it has lost the election in a peergroup
Hlink_Aggr	Some lower level links are being aggregated at a higher level
NSP_Aggr	Peergroup topology is being aggregated as the complex representation of the logical group node
Hello_Up	A Hello FSM has entered 2-way inside or common hierarchy
Hello_Down	A Hello FSM has left 2-way inside or common hierarchy
NP_Full	A Nodepeer FSM has entered Full State
NP_Down	A Nodepeer FSM has left Full State
NP_Exchanging	A Nodepeer FSM has entered Exchanging State