Project Description

1. Research objectives and impact

Toward building large-scale switches, the current industry focus is on increasing packet handling capacities of switch fabrics from Gb/s to Tb/s. Although an increase in packet handling capacities and line card data rates requires a corresponding increase in call handling capacities of switches, this problem has received little attention. This is because most of the work on scalability of packet switch fabrics has targeted connectionless internet protocol (IP) routers, while call handling arises only in connection-oriented networks. However, in the last few years, resource reservation to support Quality-of-Service (QoS) guaranteed flows (or connections\(^1\)) has gained attention. The Internet Engineering Task Force (IETF) is currently addressing this problem of how to upgrade IP routers with connection-oriented capabilities.

An important aspect of a connection-oriented network that impacts call handling capacities is the signaling protocol that is used to set up and release connections. Signaling messages can be complex with many parameters and timers. Furthermore, state information associated with calls can become unwieldy. Consequently, signaling protocols have traditionally been implemented in software yielding call handling capacities in the order of 1000-10,000 calls/sec [1]-[10]. QoS control solutions are being developed and evaluated based on this premise that call handling capacities do not scale with the packet handling capacities of switch fabrics. Integrated Services (Intserv) [11] and Differentiated Services (Diffserv) [12] are two such QoS control solutions that are compared under this assumption. This assumption regarding call handling capacities has also relegated circuit-switched networks, including high-speed Wavelength Division Multiplexed (WDM) networks, to just serve as wires.

The objective of this proposal is to challenge this basic assumption by demonstrating call handling capacities in the order of millions of calls/sec. Such a significant change to this basic assumption regarding call handling capacities would indeed have a far-reaching impact on both QoS control mechanisms for packet-switched networks, and on the use of emerging high-speed circuit-switched networks, such as WDM, for new and challenging applications.

Our solution approach is to implement signaling protocols in hardware. This will dramatically improve call handling capacities. Perceived drawbacks of hardware implementations are inflexibility and inability to handle complex tasks. We propose to use reconfigurable Field Programmable Gate Arrays (FPGAs) [13]-[17] to solve the inflexibility problem. These devices are a compromise between

---

1. In the proposal, we use “call,” “connection,” and “flow” interchangeably.
general-purpose processors used in software implementations at one end of the flexibility-performance spectrum, and Application Specific Integrated Circuits (ASICs) at the opposite end of this spectrum. FPGAs can be reprogrammed with updated versions as signaling protocols evolve while significantly improving the call handling capacities relative to software implementation. As for the challenge posed by the complexity of signaling protocols, we propose to implement the basic and frequently-used operations of the protocol in hardware, and relegate the complex and infrequently-used operations (for example, processing of optional parameters) to software.

Besides flexibility and message complexity, a third research challenge is the state information maintained by signaling protocols. Contrast this with stateless protocols that have been implemented in hardware. For example, ethernet frames, Asynchronous Transfer Mode (ATM) cells and IP datagrams are simply forwarded without any state being maintained for each packet or “flow.” Recently, a transport layer protocol that requires state maintenance at endpoints, i.e., TCP, has been implemented in hardware [18]. Lessons learned from this effort will be considered in our project. However, the functionality of a signaling protocol is considerably different from a transport layer protocol.

The proposed work items for this project include (i) implementation of a typical signaling protocol in FPGAs, (ii) design of a switch controller board using these signaling protocol FPGAs, and (iii) quantifying measures of the implementation to determine achievable call handling capacities. Papers will be written to disseminate our results, allowing the community to explore hardware-based and similar solutions to scaling the QoS-controlled flow handling capacity of switches. We will also make available the protocol processing engine architecture, verified synthesizable VHDL models and the prototype board design to researchers involved in QoS control development.

Section 2 provides background material on signaling protocols and reviews prior work. In Section 3, the motivation for this project is provided. Section 4 describes a feasibility study based on our proposed approach and Section 5 outlines a three-year research plan for this project. The broader impact of this project, and a summary of our previous results are described in Sections 6 and 7, respectively.

2. Background and prior work

Connection-oriented networks are either circuit-switched or packet-switched. In circuit-switched networks, the switching action is based on the “position” of the arriving bits, where “position” is determined by the incoming interface number, time slot and/or wavelength. In packet-switched networks, packets that arrive on the connection after it has been set up carry the labels corresponding to the connection in their headers. When packets arrive at a switch in a packet-switched connection-oriented net-
work, mapping tables are consulted, labels are typically modified to corresponding values for the outgoing link, and packets are forwarded to the next node.

Fig. 1 shows a generic NxN switch architecture with N input and N output interfaces. Input line cards process protocol headers and demultiplex the input signals received on the interfaces. Based on header information in packet switches or the position of arriving bits in circuit switches, the appropriate output interface for each demultiplexed unit is determined. After passing through the switch fabric, the demultiplexed units are routed to appropriate output line cards and sent out.

The switch controller processes signaling protocol messages and programs the switch fabric.

Connection setup and release procedures, along with the associated message exchanges, constitute the signaling protocol. Setting up a connection in a circuit- or packet-switched connection-oriented network consists of four steps:

1. Determining a route for the connection: this is done by consulting routing tables created from data collected via routing protocols, such as link-state or distance-vector protocols.
2. Performing connection admission control to determine if there are sufficient resources, reserving these resources, and selecting channel identifiers (interface numbers and time slots in Time Division Multiplexed (TDM) circuit switches; interface numbers and wavelength numbers in WDM circuit switches; interface numbers, Virtual Path Identifiers/Virtual Channel Identifiers in ATM switches; interface numbers and labels in MultiProtocol Label Switches [19]; a combination of source and destination IP addresses, source and destination port numbers, protocol type, etc., in IP switches [20]).
3. Programming switch fabrics by creating table entries that map incoming channel identifiers to outgoing channel identifiers, and programming schedulers.
4. Updating the connection state information and managing connection references.

Following data transfer, a release procedure is executed to release the reserved resources and to delete entries from the appropriate tables.

Signaling protocols have their roots in TDM circuit-switched telephony networks. These protocols include Q.931 [21] and ISDN user part [22]. 64 Kbps (DS0) circuits are set up and released on demand.
as phone calls are placed and completed. Signaling protocols are also used in packet-switched connection-oriented, such as X.25 and ATM networks. The User Network Interface (UNI) signaling protocol [23] and Private Network-to-Network Interface (PNNI) signaling protocol [24] are used to set up and release virtual circuits in ATM networks. Signaling protocols for IP networks are a more recent development, given the emerging interest in augmenting the connectionless IP networks with some connection-oriented networking capabilities. Examples include the Resource reSerVation Protocol (RSVP) [25], YESSIR [4] and Label Distribution Protocol (LDP) [26].

Other signaling protocols have also been proposed in the literature [7]-[10], [27]-[32]. Some of these protocols such as Fast Reservation Protocol (FRP) [33], fast reservation schemes [8] [10], YESSIR [4], UNITE [7] and PCC [28] [29] have been designed to achieve low call setup delays by improving the signaling protocols themselves. FRP is the only signaling protocol that has been implemented in ASIC hardware. Such an ASIC implementation is inflexible because upgrading the signal protocol implementation entails complete redesign of the ASIC. The proposed research targets improving call handling capacities and does it in a flexible manner by implementing the signaling protocol in reconfigurable hardware.

3. Motivation

The motivation for this work is to increase the call handling capacities of switches by at least two orders of magnitude thereby changing a fundamental assumption made in the design of QoS control mechanisms for packet-switched networks, and in the development of applications for high-speed circuit-switched networks, such as WDM and Synchronous Optical Networks/Synchronous Digital Hierarchy (SONET/SDH). Sections 3.1 and 3.2 describe examples illustrating the need for high call handling capacities in packet-switched connection-oriented networks and circuit-switched networks, respectively. Section 3.3 discusses the drawbacks of making the assumption that call handling capacities of switches do not scale well with switch fabric size increases.

3.1 Need for high call handling capacities in packet-switched connection-oriented networks

Given the industry momentum to carry voice telephony traffic over packet-switched networks, there is an increasing awareness of the need for delay guarantees for interactive sessions. Interactive voice traffic has a one-way maximum delay requirement of 150ms for excellent quality voice (with echo cancellers) and 400ms for acceptable quality voice [34]. Such a guarantee is difficult to provide in connectionless networks, such as currently-deployed IP networks. For example, we once measured a 45 sec delay difference between a live TV broadcast and a “live” real-audio broadcast on the Internet of a
A number of telecommunications service providers are currently deploying ATM networks with TDM-ATM gateways to relieve the capacity exhaust problem in their tandem switches (switches that interconnect central offices). To support the large number of voice calls, on-demand setup/release of ATM virtual circuits is needed. Consider admitting voice calls into a Fore Systems’ ASX-4000 ATM switch, which supports a maximum of 224 ports and has a switch fabric capacity of 40Gbps [1]. Assuming that all virtual circuits are for voice calls (a worst case scenario), and the use of 32Kbps coding [35], \((40\text{Gbps})/(32\text{Kbps}) \approx 1.25M\) calls can be held simultaneously. The mean holding time of voice calls has been measured to be 3 minutes. The signaling protocol processor should ideally keep the switch interfaces fully loaded, i.e., it should handle \((1.25M)/(180\text{sec}) = 7000\text{calls/sec}\). If 8Kbps voice coding is used, then this number increases to 28000 calls/sec. Fore Systems reported that the ASX-4000 can support 1200 calls/sec and about 100,000 connections [1]. With Tb/s switch fabrics, these numbers increase by another two orders of magnitude. For voice over IP calls, YESSIR simulations show that a maximum of 1550 calls/sec can be handled [4]. This paper notes that even one OC3 can support 2400 voice calls at 64Kbps, which means that a Tb/s IP router in the worst case (i.e., all IP flows are voice calls) needs to support over 15M simultaneous calls.

Clearly not all flows will be voice calls, and hence arguably, the call handling capacities of switches do not need to be as high as projected above. However, while telephony voice represents a human-to-human form of interactive communication, opening up the possibility of supporting human-to-computer and computer-to-computer interactive communications on connection-oriented networks will indeed require major increases in call handling capacities. For example, telnet and web accesses are interactive. Reference [36] describes the delay needs of interactive applications (a 200ms delay number has been mentioned for telnet by Van Jacobson). In the limit, using empirical data from a paper by Thompson et al. [37], which shows that on average only 10KB are exchanged per flow, a Tb/s switch would handle 100M flows/sec. Attempting to perform flow-based QoS control of such a large number of flows is often dismissed as “impossible.” For example, inability to handle a large numbers of flows/sec is the most cited disadvantage of RSVP. Demonstrating the possibility of handling such large numbers of flows/sec with hardware implementation could indeed impact how a number of applications are currently implemented, not just voice.
3.2 Examples illustrating need for high call handling capacities in WDM circuit switches

A second motivating example for increasing call handling capacities is found in WDM circuit-switched networks. Gilder, in his recent Gilder Technology Report [38], prophesies a return to circuit switching on the assumption that bandwidth will become a commodity given the very large WDM switch fabrics being unveiled by new optical networking companies (Xros demonstrated a 1152 x 1152 WDM circuit switch) [39], and Lucent’s Allwave technology [40], which allows over 3300 wavelengths per fiber. In other words, if bandwidth is no longer expensive, it is less important to send data in packets to exploit silences in bursty traffic. Instead a lightpath can be set up end-to-end in an all-optical bit-rate-transparent WDM network. A key however would be the ability to set up and release lightpaths on demand to allow for a “call level” sharing as opposed to a “packet level” sharing of resources. While we do not entirely subscribe to this completely-packet-switchless network concept (it seems a complete waste to carry a 64Kbps voice call on an end-to-end lightpath capable of carrying 10Gbps), we would nevertheless like to enable such a possibility by scaling the call handling capacity.

The notion of “switched lightpaths,” i.e., lightpaths set up and released on-demand has already been proposed. For example, Enron bandwidth trading service [41] has already changed how bandwidth is leased (i.e., for wire service), allowing for customers to trade bandwidth on a daily or even hourly basis. A proposal for dynamic management of lightpaths has been defined by Optical networks [42] in the Optical Link Management Protocol (OLMP) Suite, which is part of a Dynamic Transport System (DTS). Such thinking opens up the need for signaling protocols for WDM networks to support switched lightpaths. Besides leased-line services, such lightpaths could also be used for web mirroring (which requires large downloads of web site information [43]). However, if the holding times of these lightpaths are long (as in leased lines) or the numbers of lightpaths needed are small (as in initial web mirroring deployments), the call handling capacities of WDM switches are still manageable with software implementations of signaling protocols. However, in the future, bandwidth trading in seconds and widespread use of personal web site mirrors remains a possibility.

Going one step further, we realized that WDM lightpaths are indeed ideally suited for large end-to-end bulk-data transfers. In fact, work done in 1975 by Miyahara et al. [44], which is reproduced in Mischa Schwartz’s book [45], shows analytically that for large file transfers, at comparable data rates, circuit-switched networks outperform packet-switched networks from a delay and throughput point of view. Since there is no burstiness involved in such bulk transfers of stored files from one computer to another, high-bandwidth WDM lightpaths can be set up on demand, used to download a file (application data directly on the WDM layer without TCP/IP) and then released. We are currently developing
a hybrid solution where URLs are sent using TCP/IP, and bulk data is downloaded directly on a WDM lightpath. The implication of such end-to-end applications is that given the large number of end hosts, very large call handling capacities could indeed be needed. For example, the Xros switch, if used in conjunction with Lucent’s Allwave technology, should support \(3300 \times 1152 = 3.8\text{M}\) lightpaths simultaneously. In the worst case, if all these lightpaths are used to transfer 10MB files at data rates of 10Gbps (in all-optical networks, this data rate is determined solely by the transceiver’s rate), assuming that end hosts can keep up with data arriving at this rate, the holding time of lightpaths is as small as 8ms. This implies a need to handle \(3300 \times 1152 / 8 = 475\) Mcalls/sec!

### 3.3 Drawback of assuming that call handling capacities do not scale

Any recognition of the fact that large numbers of flows (or calls) are hard to manage leads to solutions based on provisioning (i.e., reserving high-bandwidth pipes a priori). This “age-old” solution was indeed first used (and is still in use) in telephony networks. A DS0 circuit for a voice call is seldom set up at every transit circuit switch along the end-to-end path. Instead, a DS1 pipe passing through a large number of core digital crossconnects (this number could be as high as 30 going across the USA) is set up a priori. A DS0 circuit then only requires call setup at the edge (local/regional) switches. This concept of using hierarchical or layered “connections” was also applied to ATM networks, which resulted in the concepts of “virtual paths” and virtual channels.” More recently, in IP networks, IETF’s Diffserv [12] approach also proposes this method for offering a coarse level of differentiated services without managing individual flows.

In [28], we demonstrated the disadvantages of such provisioning. Fig. 2 is a graph from [28] that shows the bandwidth required on a link under two configurations. Whereas configuration 1 uses provisioned virtual paths, virtual circuits are set up on demand in configuration 2. Example 1 is under exact traffic characterization and example 2 is under approximate traffic characterization. For an example network, the graph shows that to achieve a 0.1% blocking probability, setting up virtual circuits on demand yields 14.4% and 47.9% resource savings.

---

2. A simpler transport layer protocol is needed to maintain end-to-end flow control given differences in computer speeds.
under exact and approximate traffic characterization assumptions, respectively, when compared to the provisioned virtual path case. The approximate traffic characterization case is the more likely. Our proposed research would preclude the need for provisioned pipes, and hence this inefficiency.

A second disadvantage of “living” with the assumption that call handling capacities do not scale well is that Service Level Agreements (SLAs) need to be established a priori to support these high-bandwidth provisioned pipes. This is counter to the highly-successful Internet model that grew outwards from small campus networks interconnecting with hardly any inter-campus contracts. Supporting the switched mode of connection-oriented networking, where a connection setup request is sent whenever needed, possibly traversing large numbers of switches, maintains the spirit of the Internet model.

4. Feasibility study

Our proposed solution approach to achieve the 2-3 orders of magnitude improvements in call handling capacities is to implement signaling protocols in hardware. In other words, the switch controller shown in Fig. 1 will be implemented in reconfigurable hardware (instead of a typical general-purpose processor based software implementation). Interestingly, since processing of call setup messages can in some ways be compared to the processing of packet headers, much of the work done on hardware based packet switches [46]-[50] can be reused here for switch controller designs. Such designs should consider queueing and scheduling effects in the same manner as these aspects are considered in packet switches [51][52]. We will explore these design issues as part of this project.

To test the feasibility of hardware implementation of signaling protocols, we extracted a subset of messages from a number of signaling protocols and implemented this subset in reconfigurable hardware. The reconfigurability feature of FPGAs allows us to add more features to the base subset. We also undertook the design of a switch board with the FPGA-based switch controller and a switch chip.

4.1 Signaling protocol subset for implementation in hardware

As a basic first step, we implemented four signaling messages, Setup, Setup-Success, Release, and Release-confirm. Processing a signaling message entails manipulating data tables at each switch along the end-to-end path. These data tables are listed in Table 1.

Connection setup consists of route determination, connection admission control and channel selection, switch fabric configuration, and state information updating. We assume that routes are precomputed and stored in a routing table. Route determination is then reduced to a simple routing table lookup. This lookup yields the node address for the next hop. Connectivity table lookup yields the interface numbers connected to this next hop node.
**Connection admission control** (CAC) consists of determining resource availability for the connection on any of the interfaces to the next-hop node identified during the routing table lookup step. In this first implementation, we only considered bandwidth. This will be extended to include more complex traffic descriptors, QoS parameters, such as loss and delay, and buffer space. This operation requires the signaling protocol processing engine to read from/write to the *available resources table*.

The third step in connection setup, *switch fabric configuration*, consists of writing the switch configuration data table with incoming and outgoing channel identifiers. The exact channel identifier is specific to the type of switch. Although *switch fabric configuration can be carried out either in the forward setup or in the reverse directions*, we implemented it in the forward direction.

The final step in connection setup entails updating the *connections state table*. Each connection is identified by a connection reference number. Once again, different mechanisms can be used to assign connection reference numbers. A connection reference number can be globally unique, unique to a switch, or unique to an interface on a switch. In our implementation, the connection reference numbers are unique to a switch. Thus, each connection passing through a switch has a different connection reference number. These connection setup actions occur sequentially as the setup message moves from one switch to the next until the destination is reached.

*Setup-success* messages are returned in the reverse direction. Since the switch fabric configuration in our implementation is carried out in the forward direction, processing of the *Setup-success* message is very simple. It only requires an update of the state table entry corresponding to the connection.

Connection release follows a similar sequential procedure. A *Release* message is generated at one end of the connection and each switch receiving the message deletes one or more rows in the switch configuration table. Next, all released resources are added to the available resources data table. Finally,
the connections state table is updated. Release-confirm message deletes the entry in the connections state table and releases the associated connection reference number. The state diagram for this subset signaling protocol implemented for this feasibility study is shown in Fig. 3. Failure and error scenarios are more varied and are hence handled in software.

![Fig. 3 Signaling protocol state transition diagram](image)

4.2 FPGA implementation of the reconfigurable signaling protocol processing engine

The architecture of the reconfigurable signaling protocol engine is shown in Fig. 4. For the feasibility study, we implemented all the blocks shown in Fig. 4 in a single FPGA. The external interfaces of this FPGA are to a bus that could serve both a switch fabric and a general-purpose microprocessor, a link termination device, and memory, both RAM and Content Addressable Memory (CAM).

The reconfigurable signaling protocol engine is composed of a message parser, a message processor, control registers and working registers. Whereas the message parser is responsible for receiving and dispatching signaling messages, the message processor performs all the actions needed for call setup and release. The signaling protocol engine interfaces with the line termination devices via two handshaking signals. The handshake

![Fig. 4 Architecture of the signaling protocol engine](image)
signal from the protocol engine indicates that the device is busy, while the handshake signal from link termination device indicates that a signaling message has arrived.

The message parser and message processor state diagrams are shown in Fig. 5a and b, respectively. The message processor is the core of the signaling protocol engine interacting with the message parser, external memory that implements the routing table, connectivity table, available resource table and connections state table (see Table 1), and the switch fabric, which maintains the switch configuration table.

Initially, the message parser and the message processor are idle. On receiving a signaling message, both move to the receive message state and remain there until the entire message is received into the Rx_Msg registers. Then the message parser asserts the start_processing signal to inform the message processor of a new message and goes into the idle state. The message processor transitions into the message decode state where it decodes the message type field of the signaling message and takes one of four branches in the state machine. States 41-47 correspond to the Setup message, states 31-33 correspond to the Setup-success message, states 21-24 correspond to the Release-confirm message and states 11-13 correspond to the Release message. Once the signaling message processing is complete, the message processor asserts the send_message signal. This triggers the message parser to transition into the send_message state and sends the outgoing Setup message.

As an example, consider the Setup message processing. In state 41 the message processor initiates the routing table lookup (the first step of connection setup, see Section 4.1) and transitions to state 42. An error condition, such as an infinitely-looping Setup message (indicated by a time-to-live field) or a
routing table lookup failure will cause a transition to the error state. Otherwise, the message processor decrements the time-to-live field by one and goes to state 43 where it waits for the routing table lookup to complete. Besides the integrated routing/connectivity table, the available resources table is also consulted in this phase. On a successful data retrieval from memory, the matched entries including next-hop address, interface numbers and available resources for each interface are loaded into the working registers. In state 44, the second step of connection setup, i.e., connection admission control, as described in Section 4.1 is carried out. The message processor checks if the resources requested in the Setup message can be allocated. If the requested resources can be allocated, the message processor moves into state 45 and the available resources table is updated to reflect the resources allocated to this connection. Otherwise, the call setup is terminated and the message processor goes into the error state. In state 46, the third step of connection setup, switch fabric configuration, as described in Section 4.1, is executed. The message processor updates the switch configuration table through the bus interface to the switch fabric/general-purpose microprocessor. The final step of connection setup is to update the state information in the state table located in the memory external to the FPGA. This requires the assignment of a free connection reference number to this new incoming connection. Management of free and assigned connection reference numbers was one of the challenges we identified with hardware implementation of signaling protocols. As part of this project, allocation and release of connection references will be implemented using two first-in first-out queues.

The message processor now prepares the information needed to send an outgoing Setup message to the next hop node identified by the routing table lookup. It gathers all the parameters for this message and places them into the working registers in state 47. It initializes \( tx_{cnt} \) with the number of 16-bit words to be transmitted. Finally, the message processor asserts the send_message signal to trigger transmission of the outgoing Setup message and returns to the idle state. The self-loops in the state machine indicates that these operations may take several clock cycles.

We developed a prototype VHDL model for the signaling protocol engine and then compiled, simulated and synthesized it using Altera Maxplus II 8.3 FPGA design tools on a Pentium III 450 IBM PC with 64MB memory. The design fits into an Altera FLEX 10K100GC503-3 FPGA device with 60% resource utilization (about 30,000 gates). In most signaling protocols, Setup message processing consumes the most amount of time, which includes the time to parse the received Setup message, process the message, and construct an outgoing Setup message.

From the timing simulation of the Setup message, which is included in a detailed technical report [53], we determined that receiving a Setup message and transmitting a Setup message consume 9 clock
cycles each, and Setup message processing consumes 25 clock cycles (this includes a match time of 5 clock cycles for the routing/connectivity table, a match time of 3 clock cycles for the available resources table based on Motorola MCM69C432 CAM device with a worst case match time of 8 clock cycles). Overall, this adds to 45 clock cycles (with two dead cycles) to receive, process and transmit a Setup message. Processing Setup-success, Release and Release-confirm messages consumes about 15 clock cycles each since these messages are much smaller (four 16-bit words vs. nine 16-bit words for Setup) and require simple processing. Assuming a 25 MHz clock, this translates into 1.8 microseconds for Setup message processing and about 0.6 microseconds for Setup-success, Release, and Release-confirm message processing. Compare this with the 1-2 milliseconds it takes to process signaling messages in software [4]. Using a slightly faster clock (50 MHz) halves these processing times. FPGAs with 100 MHz clocks are already on the market and using them will decrease the processing time of the messages even further. Pipelined processing of signaling messages by selective duplication of the data path will further improve the throughput. Overall, this feasibility study has demonstrated the potential for 1000 x^3 speedup by using a reconfigurable signaling protocol processing engine.

Besides processing times, the memory requirement to store state information for large numbers of calls is an issue. Since memory is external to the FPGA, we address this issue in the next section.

### 4.3 Integrating the signaling protocol engine with a switch fabric

A second step in the feasibility study was to determine if the FPGA implementation of the signaling protocol could be used in an actual switch as the switch controller (see Fig. 1). Towards this end, we started with the design of a board containing our signaling protocol FPGA and a switch fabric.

As an example study, we selected the SONET/SDH TADM device from Lucent Technologies [54] as the switch. This ASIC is

---

3. perhaps even 10,000x speedup is possible with 100Mhz FPGAs and pipelining.
versatile in that it combines line cards and the switch fabric in one chip. It supports 155, 622, and 2488 Mbits/s SONET/SDH interfaces with a cross-connect rate of STS-1 granularity between receive ports. It supports STS-48/STM-16, quad STS-12/STM-4 and quad STS-3/STM-1 interface rates. The switch fabric can be configured through the local data bus either by a microprocessor or by the signaling protocol engine. In our design, we are using the signaling protocol engine to create and delete entries in the switch configuration table located on the switch ASIC.

As stated in Section 4.2, the FPGA does not have enough memory to store state information for all connections, and hence external memory is used. As shown in Fig. 6, external memory consists of both RAM and CAM. The external RAM is used to maintain the connections state table. In our feasibility study, the signaling protocol message fields were defined to support up to 4096 connections simultaneously. With a per-connection state information of 116 bits (i.e. about 15 bytes), this translates into a 4K x 116-bit RAM. Compare this with the 500 bytes per connection reported for RSVP [4]. Choice of the signaling protocol parameters to support 4096 connections also need to be relaxed. It was shown in Section 3, that the Xros WDM switch can support about 3.8M simultaneous lightpaths. With a per-connection state information of 15 bytes this translates into about 60MB of RAM. As part of this project we will explore if the signaling protocol can be further simplified to reduce per-call state information. Combining this with the availability of cheap 16MB and 32MB RAMs, the memory requirements to handle such large number of simultaneous calls will fall within the feasible range.

Integrated routing/connectivity table entries are at least 64-bits wide and are indexed using 32-bit destination IP address. Several efficient architectures have been developed to implement routing tables with RAMs [55]-[56], while others have used the more expensive and versatile CAMs [57]-[58]. In our feasibility study we assumed a CAM-based routing/connectivity and resource availability tables.

Other blocks shown in Fig. 6 include appropriate link termination devices. These are used to handle the lower-layer protocols of the signaling messages. Finally, there will be a generic microprocessor on the board to manage an on-board local bus that connects to all devices. By reading from and writing to the registers in these devices, it provides a channel for the software to control the hardware.

5. Research schedule and management plan

In the first year, we will identify the subset signaling protocol, define clean and simple interfaces to the other devices on the board, and develop and simulate the VHDL model of the message processor and parser. In the second year we will synthesize the signaling protocol processing engine, perform timing simulation, program an FPGA with the signaling protocol processor, verify its functionality
using test cases and perform a preliminary board design. Close attention will be paid to any aspects of signaling protocol functionality that are difficult to implement in hardware. This could lead to changes in signaling protocol specifications. Also, we plan to take measurements and use the service times in queueing models to analyze whether scalability to the large orders of magnitude promised with hardware implementations are indeed possible. In the third year we will finalize the PCB design, fabricate the board, integrate the various devices and demonstrate improvement in the call handling capacity.

6. Broader impact on infrastructure and education

CATT (Center for Advanced Technology in Telecommunications) is a center at Polytechnic University that offers courses to disseminate the results of our research to industry. We plan on organizing such courses as well as developing an undergraduate course on hardware implementation of network protocols. The course will teach design, analysis, software simulation, reconfigurable hardware implementation and verification of network protocols. The infrastructure used in this project will form the basis for the protocol design work benches used in teaching this laboratory based course.

While packet switching, especially of the connectionless variety, is receiving an unprecedented amount of attention in the commercial sector owing to the success of the Internet, we feel that it is important to the goals of NSF ANIR to foster heterogeneity in networks to keep alive innovation and progress by supporting this proposal, which opens up tremendous possibilities for connection-oriented networking, both of the packet-switched nature and circuit-switched (be it WDM or TDM).

7. Summary of previous results and achievements

The proposed project is joint work of Prof. Malathi Veeraraghavan and Prof. Ramesh Karri. Malathi Veeraraghavan has worked on signaling protocols for a number of years. As member of the Center for Advanced Technology in Telecommunications (CATT) at Polytechnic University, she recently received a research grant from Bell Atlantic to work on the SS7 signaling protocol. She has also designed parallel connection setup algorithms for ATM networks.

Ramesh Karri’s research focuses on VLSI CAD algorithm development targeting high-performance, low-power, and high reliability. His research in CAD algorithms for reliable and on-line testable designs is funded by an NSF CAREER grant CCR-9996139. His group also conducts research in developing high performance architectures for various applications such as next generation high speed encryption. This combination of signaling protocol design expertise and VLSI design expertise is the basis for this collaboration.