

A Hardware-Accelerated Implementation of the RSVP-TE Signaling Protocol

Haobo Wang, Ramesh Karri

Department of Electrical and Computer Engineering
Polytechnic University
Brooklyn, NY 11201
haobo_w@photon.poly.edu, ramesh@india.poly.edu

Malathi Veeraraghavan, Tao Li

Department of Electrical and Computer Engineering
University of Virginia
Charlottesville, VA 22904
mv@cs.virginia.edu, tli@ece.virginia.edu

Abstract—Signaling protocols are primarily implemented in software for two reasons: protocol complexity and the requirement for flexibility. While these are two good reasons for implementing signaling protocols in software, the price paid is in performance. Software implementations of signaling protocols are rarely capable of handling over 1000 calls/sec. Corresponding call setup delays per switch are in the order of milliseconds. To improve performance for high-speed networks, we implemented RSVP-TE signaling protocol in reconfigurable FPGA hardware. Our implementation demonstrates the feasibility of 100x and potentially 1000x speed-up vis-a-vis software implementation. The impact of this work can be quite far-reaching by allowing connection-oriented networks to support a variety of new applications, even those with short call holding times.

Keywords—GMPLS, Hardware-acceleration, RSVP-TE, Signaling

I. INTRODUCTION

Signaling protocols are used in connection-oriented networks to set up and tear down connections. Examples of signaling protocols include the Signaling System 7 (SS7) in telephone networks [1], the Private Network to Network Interface (PNNI) signaling protocol in Asynchronous Transfer Mode (ATM) networks [2], the Resource reReservation Protocol - Traffic Engineering (RSVP-TE) [3] in Multi-Protocol Label Switched (MPLS) networks, and its extensions for Generalized MPLS (GMPLS) [4], which supports Synchronous Optical Network (SONET), Synchronous Digital Hierarchy (SDH) and Dense Wavelength Division Multiplexed (DWDM) networks.

Signaling protocols are primarily implemented in software for two reasons. First, signaling protocols are quite complex with many messages, parameters and procedures, especially the signaling protocols for GMPLS, which target almost all connection-oriented networks both packet-switched and circuit-switched. Second, signaling protocols are updated frequently requiring a certain amount of flexibility for upgrading field implementations. For example, RSVP-TE with extensions for GMPLS evolved from RSVP-TE for MPLS, which in turn evolved from RSVP [5] for IP networks. While these are two good reasons for implementing signaling protocols in software, the price paid is in performance. Signaling protocol implementations in software are rarely capable of handling over 1000 calls/sec. Correspondingly, call setup delays per switch are in the order of milliseconds [6].

Towards improving performance, we undertook a hardware implementation of a signaling protocol. We used reconfigurable hardware, i.e., Field Programmable Gate Arrays (FPGAs) to meet the requirement for flexibility. These devices

are a compromise between 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. As for the challenge posed by the complexity of signaling protocols, our approach is to only implement the time-critical operations of the signaling protocol in hardware, and relegate the non-time-critical operations to software. The signaling protocol we implemented is RSVP-TE with extensions for GMPLS, specifically for SONET switches.

We are designing a prototype board to demonstrate the feasibility of hardware-accelerated signaling. We modeled RSVP-TE in VHDL and mapped it onto a Xilinx XC2V3000 FPGA (with 14% resource utilization). This signaling hardware accelerator is the core of the prototype board. From the timing simulations, we determined that a call can be processed in 7.2 microseconds assuming a 50MHz clock (this includes the processing time for three signaling messages involved in the setup and teardown of a connection, *Path*, *Resv*, and *Path-Tear/ResvTear*). Using a pipelined architecture, a call handling capacity of 400,000 calls/sec can be achieved.

The impact of this work is quite far-reaching. By decreasing call processing delays, it becomes conceivable to set up and tear down connections more often. This allows for a finer granularity of resource sharing and hence better utilization. For example, if a SONET circuit is set up and held for a long duration, given that data traffic using the SONET circuit is bursty, the circuit utilization is low. However, if fast call setup/teardown is possible, circuits can be dynamically allocated and held for short durations, leading to improved utilization.

Section II presents background on connection setup and teardown procedures and surveys prior work on this topic. Section III describes the subset of RSVP-TE that we defined for hardware acceleration. Section IV describes our hardware implementation while Section V summarizes our conclusions.

II. BACKGROUND AND PRIOR WORK

In this section, we will provide a brief review of connection setup and teardown. We will also describe related prior work.

A. Background

Connection setup and teardown procedures, along with the associated message exchanges, constitute a signaling protocol. Setting up a connection at a switch consists of five steps:

- Determining the next-hop switch toward which the connection should be routed.
- Checking for the availability of and reserving required resources (link capacity and optionally buffer space).
- Assigning “labels” for the connection. The exact form of the “label” depends on the type of connection-oriented network. For example, in SONET/SDH switches, a label identifies time slot(s) on the incoming and outgoing switch interfaces.
- Programming the switch fabric to map incoming labels to outgoing labels.
- Updating the state information associated with the connection.

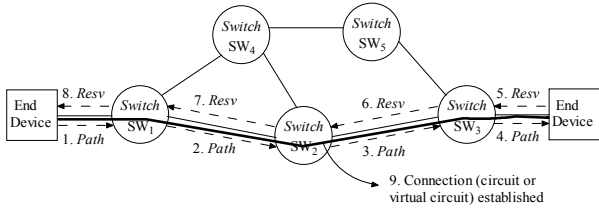


Figure 1. Illustration of connection setup

In a typical connection setup procedure as illustrated in Fig. 1, a signaling message requesting the setup of a connection (e.g. *Path* message in RSVP-TE) progresses from the calling end device toward the called end device hop-by-hop, and a response signaling message (e.g. *Resv* message in RSVP-TE) travels in the reverse direction, again hop-by-hop. The first two steps should be performed in the forward direction so that connection is routed along a path on which resources are available. The last step, updating the state information, should be performed in both directions. Steps 3 and 4 could be performed as signaling messages proceed in either direction. For example, in [3][4], labels are assigned in the reverse direction and carried in *Resv* messages. However, in [4], a switch can also assign labels in the forward direction and include these as “suggested” labels in *Path* messages. The connection release procedure follows a similar hop-by-hop approach. In RSVP-TE, this procedure may be initiated by either end with *PathTear/ResvTear* message. Switches processing the *PathTear/ResvTear* messages free up resources and label assignments.

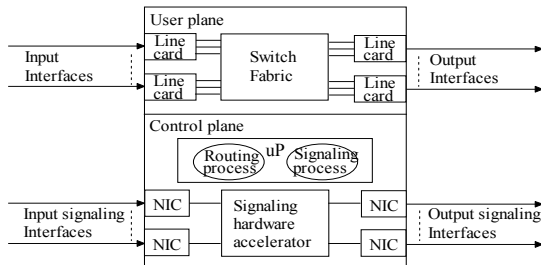


Figure 2. Unfolded view of a switch in connection-oriented networks

Fig. 2 illustrates a typical architecture of a switch in connection-oriented networks. The user plane hardware consists of a switch fabric and line cards that terminate input/output interfaces carrying user traffic. The control plane unit consists of a signaling protocol engine, which could have a

hardware accelerator as we are proposing, or be completely implemented in software. Input and output signaling interfaces carry signaling messages. These are “logical” and could be realized as multiplexed channels on the user plane interfaces.

B. Prior work

There are many signaling protocol standards as listed in Section I. In addition, many other signaling protocols have also been proposed in the literature [7]-[12]. Of these protocols, FRP [7] and JIT [12] have been implemented in hardware.

In [13], we proposed a performance-oriented signaling protocol called Optical Circuit Signaling Protocol (OCSP), which we defined for SONET/SDH networks. Our primary goal in designing OCSP was to achieve high performance. As a consequence, OCSP was designed to be simple enough for hardware implementation. Our challenge now is to take a generic flexible signaling protocol such as RSVP-TE, which was not defined for high performance, and yet demonstrate a hardware-accelerated implementation.

III. A SUBSET OF RSVP-TE FOR HARDWARE IMPLEMENTATION

As a generalized signaling protocol targeting different type of connection-oriented networks, RSVP-TE with extensions for GMPLS is complex and flexible. But most of the signaling functions are non-time-critical. It is not only impractical but unnecessary to implement the complete RSVP-TE signaling protocol in hardware. Our approach is to extract a subset of RSVP-TE functions for hardware acceleration and relegate the remaining functions to software. The former should be large enough to handle time-critical signaling functions and yet small enough to make hardware implementation feasible.

The following six aspects of GMPLS signaling protocols make hardware implementation difficult [13]:

- A large number of object (parameter) types and values for fields within objects have been defined to support a variety of switches
- Need to generate new messages significantly different from received messages and/or automatically initiate messages
- Need to maintain state information associated with each connection
- Support for timers
- Need to parse out data from the flexibly encoded parameters structured in Type-Length-Value (TLV) format.
- Additional difficulties arise because the GMPLS protocol was designed without high-performance implementation as an objective. For example, consider the connection reference parameter used to identify a connection within a switch. Since the connection reference parameter in the GMPLS protocol has a global significance, it is large. If this parameter is designed to have only a local significance, the size of this parameter can be reduced significantly making data table lookups simpler.

Some of these difficulties can be overcome by defining a subset of the GMPLS RSVP-TE signaling protocol for

hardware implementation, while others require innovative implementation techniques. In this section, we address those difficulties (specifically, the first three) that can be overcome by limiting the features supported in the subset. In section IV we will discuss implementation techniques.

We start with the specification of GMPLS RSVP-TE with extensions for SONET/SDH networks, and define a subset to include the following functionality.

- The common-case scenario handling of messages related to the setup and release of point-to-point unidirectional SONET circuits at a transit SONET switch that operates at a cross-connect rate of STS-1.
- IPv4 addresses are used to identify sources and destinations of the SONET circuits. The switch itself and its neighboring switches are assumed to have only one IP address per node. Interface numbers are used to identify interfaces of a switch. Separation of control plane from the user plane is supported by this subset.
- We allow for the presence of logical links on end-to-end paths. This means switches that are not physically connected by direct links can still be RSVP-TE neighbors.

A. RSVP-TE signaling messages

RSVP defined seven signaling messages, *Path*, *Resv*, *PathErr*, *ResvErr*, *PathTear*, *ResvTear*, and *ResvConf*. RSVP-TE added the *Hello* message for node failure detection. When RSVP-TE was extended for GMPLS, the *Notify* message was introduced to support fast failure notification. Among these messages, *Path* and *Resv* messages are used to set up a connection while *PathTear/ResvTear* messages are used to tear down a connection. We selected these four messages for hardware acceleration because *Path* and *Resv* messages are needed for connection setup, the procedure targeted for speedup, and *PathTear/ResvTear* because resources need to be released at a pace corresponding to the fast setups. The remaining messages are relegated to software.

B. Objects and fields within objects

Each RSVP-TE message begins with a common header followed by a body consisting of a variable number of variable-length “objects.” As noted earlier, there are a large number of objects and fields within objects. Consider the SESSION object as an example. In RSVP, a C-Type of 1 was defined for the SESSION object. The fields include destination address, protocol ID and destination port number. However, with the extension of RSVP to RSVP-TE, a new C-Type 7 was defined with the fields redefined to include destination address, tunnel ID, and extended tunnel ID. This means the signaling processing engine should first read the C-Type field and then based on its value parse the remainder of the object. This example illustrates why a hardware implementation for handling these objects can become quite complex. Our approach to this problem is to define C-Type of 7 in the subset and delegate the handling of all messages carrying a SESSION object with a different C-Type to the software signaling process. Details of these choices are provided in [14].

C. State transition diagram for a connection at a switch

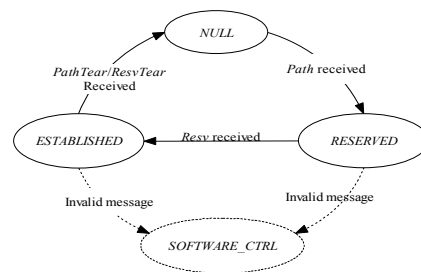


Figure 3. RSVP-TE messages and objects

Unlike in stateless connectionless networks, in connection-oriented networks, a switch needs to maintain state information for each connection. We defined four states for a connection as shown in Fig. 3. Before a *Path* message arrives, the connection does not exist, and is hence in the NULL state. After a *Path* message is received, a next hop IP address is determined and resources are reserved, the connection enters the RESERVED state. When a *Resv* message arrives, the switch fabric is programmed and the connection enters the ESTABLISHED state. When a *PathTear/ResvTear* message is received, the allocated resources on the switch are released; the state of the connection returns to NULL. The connection enters the SOFTWARE_CTRL state when the hardware accelerator cannot handle a received message, in which case it passes control of this connection’s state table entry to the software module.

D. Optional objects

Besides all the mandatory objects in *Path*, *Resv*, *PathTear*, and *ResvTear* messages, we support three optional objects, MESSAGE_ID, MESSAGE_ID_ACK, and SUGGESTED_LABEL.

RSVP supports the notion of soft-state and periodic refresh messages. If a refresh is not received before the timeout interval expires, connections are released. Since packet forwarding is based on the IP routing data table, as routing data changes, the resource reservations need to follow. Hence RSVP included the use of refresh message. A second reason for refresh messages is that since RSVP uses unreliable IP service, the occasional loss of an RSVP message is handled through refreshes. However, if the refresh interval is small, the overhead spent in processing refresh messages can become excessive; while if the refresh interval is large, it takes longer to detect the loss of an RSVP message. RFC 2961 [15] makes the case for not using refresh messages in GMPLS networks, where once a circuit is established, the routing data table is not consulted for data forwarding. To handle the reliability issue, it introduces new objects MESSAGE_ID and MESSAGE_ID_ACK, along with the concept of re-transmission timers and exponential back-offs to ensure reliable message transport. Since refresh timeout values are mandatory in RSVP, the hardware signaling accelerator accepts these values, but processing of refresh messages will be relegated to the software signaling process. We expect that refresh messages are not likely to be used in GMPLS networks, and even though the extensions proposed in [15] are currently optional, we expect these to become widely adopted in implementations of RSVP-

TE for GMPLS networks. Hence we support these objects in our hardware-accelerated subset.

In RSVP, the downstream switch selects the label. RSVP-TE with extensions for GMPLS allows for the *Path* message to carry a SUGGESTED_LABEL object, though it is optional. We note that there is potential conflict in SONET networks if the SUGGESTED_LABEL object in the *Path* message is left as optional. Consider the following scenario. An outgoing interface (STS-12) has four available time slots, 000 011 111 111. A first call requests an STS-3c; the upstream switch tentatively reserves the first three time slots. A second call requesting an STS-1 arrives next, and needs to be routed on this same outgoing interface. The switch will then make a tentative reservation of the remaining time slot. If the *Resv* message for the second call returns first, and the downstream switch assigns a time slot different from the one tentatively reserved in forward direction, the first call for which a tentative reservation was made can no longer be accommodated because the it requires a concatenated assignment. Hence, we recommend that the SUGGESTED_LABEL object be mandatory in *Path* messages to force the downstream switch to use the label selected by the upstream switch. This is the result of RSVP-TE growing out of RSVP, which was developed as a protocol for receiver-initiated additions to a multicast tree. In GMPLS networks, where hard resource reservations of time slots and wavelengths are necessary, a reservation and corresponding timeslot/wavelength selection needs to be made in the forward direction of call setup.

IV. A HARDWARE IMPLEMENTATION OF THE RSVP-TE SUBSET

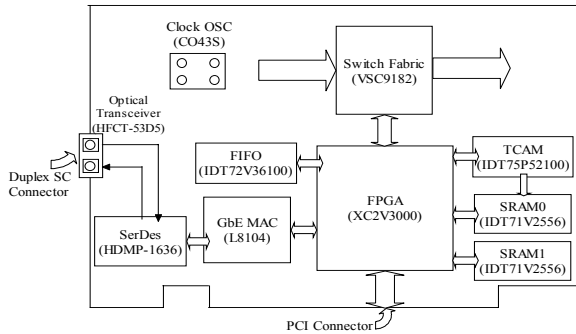


Figure 4. Architecture of the prototype board

To demonstrate the feasibility of hardware-accelerated RSVP-TE signaling, we are implementing a prototype board as shown in Fig. 4. The core component on the board is a hardware signaling accelerator, which is a Xilinx XC2V3000 FPGA. It implements the RSVP-TE subset described in Section III. We use a single Gigabit Ethernet (GbE) interface as the signaling link (see the input and output signaling interfaces shown in Fig. 2). All RSVP-TE messages are carried on IP on this interface. Messages sent to different neighboring switches can all be carried on the same GbE link and switched through an IP router to their appropriate destinations.

Processing of RSVP-TE messages requires many data table lookups. We use a TCAM and an SRAM (SRAM0) to hold these data tables. Another SRAM (SRAM1) is used to buffer

the transmitted but as-yet unacknowledged messages. The FIFO is the interface unit between the hardware signaling accelerator and the software signaling process. Signaling messages that cannot be handled by hardware are buffered in the FIFO. The software signaling process then reads the messages from the FIFO and processes them as needed.

Finally, we have a SONET switch fabric ASIC that operates at a cross-connect rate of STS-1 (note that in Section III, we stated that this RSVP-TE subset is being designed for such a SONET switch). Specifically, we selected the VSC9182 device, which is a 64x64 cross-connect with STS-12 interfaces. We include this user plane device mainly to test whether or not our RSVP-TE engine can perform the functions of configuring and deleting cross-connections through the user plane cross-connect chip.

Fig. 5 illustrates the architecture of the hardware signaling accelerator. It has three stages, object dispatching, object processing, and object assembling. In the object dispatching stage, signaling messages are buffered in an internal SRAM for checksum verification. Meanwhile, objects and fields are checked and dispatched to different registers. In the object processing stage, message objects are processed in parallel. Finally, appropriate objects are reassembled into a new message, which is then sent to the next switch. These three stages are fully pipelined to achieve a high throughput.

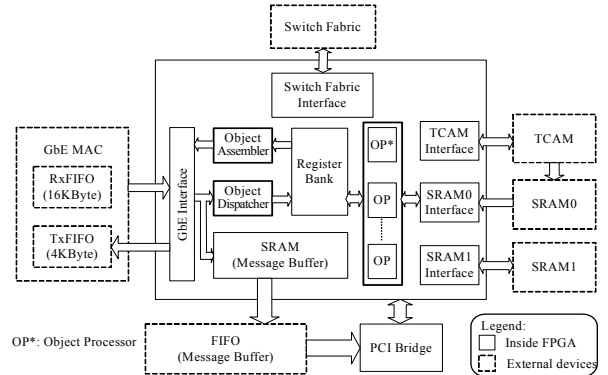


Figure 5. Architecture of the hardware signaling accelerator (FPGA)

A. Implementation of the object dispatcher

Instead of defining objects at fixed location within messages, RSVP-TE uses a flexible TLV structure. Each object is a self-contained element and composed of a type field (Class-Num and C-Type), a length field (Length), and a variable-length value field (Object contents). The objects can appear in any order (with some constraints). The TLV structure was designed for flexibility, allowing protocol designers to add parameters in arbitrary order. But this construct makes parameter extraction in hardware a complex task. This difficulty (aspect 5 in the above list) however cannot be overcome by limiting the subset definition. It needs innovative implementation, which is described in Section IV.

The TLV structure makes parameter extraction in hardware a complex task. For example, when processing an IP packet header, the hardware can always extract the 5th word from the IP header to obtain the destination IP address. But in RSVP-TE messages, since the SESSION object carrying the destination

IP address can occur anywhere in the message, it is hard to find the object and extract its fields.

In order to solve this challenge, we design a scheme with two-level dispatching. The message dispatcher first delimits a message and objects based on message length and object length. The delimited object is then sent to all object dispatchers. But only the object dispatcher matching the object type is triggered and the fields in the object are dispatched into corresponding registers. The unknown object processor captures all unsupported objects. If a message contains such an object (i.e., one outside the set of objects defined in the RSVP-TE subset), the message should be passed to the software signaling module.

As objects are dispatched, different fields of the objects are verified with the preset values supported by the subset we defined. These values could either be "hardwired" or set in initialization registers. For example, the hardware only supports the SESSION object with C-Type 7 as described in Section III. This C-Type value (7) is "hardwired." An example of an object whose field values are handled by storing data in initialization registers is the LABEL_REQUEST object. This object specifies the type of switch supported by this signaling engine. Since we are defining this subset for a SONET switch, the field LSP Encoding Type within this object can have only one value, i.e., 5 (for SONET/SDH). The field Switching Type within this object can have one value 100 (for TDM). We place these two numbers 5 and 100 in corresponding registers during initialization time and design the hardware circuitry to compare the values in an incoming message carrying this LABEL_REQUEST object with the values stored in these registers. This makes our hardware implementation more flexible allowing the same implementation to be used even if the switch is, for example, a Lambda switch, in which case these two values would be reset to 8 and 150, respectively. The tradeoff between performance and flexibility was considered when we made our choices of which values to hardwire and which values to store in initialization registers.

B. Implementation of data tables

We defined six data tables to support our implementation of the RSVP-TE subset defined in Section III, including Routing table, Incoming Connectivity table, Outgoing Connectivity

table, Outgoing Connection Admission Control (CAC) table, User/Control Mapping table, and State table.

The Routing table is used to determine the next-hop switch. Traditionally a routing table is implemented in software using Trie data structures [16]. Trie-based routing tables are often large, and lookups are slow. Many schemes have been proposed to compress the size of the routing table and improve its performance [17][18]. However, these schemes need complex user logic. We use a Ternary Content-Addressable Memory (TCAM) and a SRAM to store the routing table. The index part, a 32-bit destination IP address, is stored in TCAM, and the return value, a 32-bit next hop address, is stored in an associated SRAM. TCAMs have fast lookup speeds, are flexible, and allow for simple user logic. The drawback of TCAMs is scalability, which we currently ignore because of the prototype nature of this implementation.

The Incoming Connectivity table determines how the interface ID used by a neighbor maps to a local incoming interface. The index into the table, stored in the TCAM, is the combination of a 32-bit previous IP address and a 6-bit output interface number. Similarly, the Outgoing Connectivity table maintains data on the local output interfaces leading to neighboring switches. The Outgoing CAC table maintains the available bandwidth on the output interfaces. Because the control plane and user plane on a switch can have different IP addresses and the next-hop address obtained from routing table lookup is for the user plane, we defined a User/Control Mapping table to map a 32-bit user plane IP address to the corresponding 32-bit control plane IP address.

The State table maintains the state information for each connection. In RSVP-TE, a 128-bit 5-tuple $\langle \text{Src_IP_Addr}, \text{LSP_ID}, \text{Dest_IP_Addr}, \text{Tunnel_ID}, \text{Ext_Tunnel_ID} \rangle$, uniquely identifies a connection. The number of simultaneous connections current-day switches can support is in the order of thousands. For example, the VSC9182 device on our prototype board can support at most 768 simultaneous connections. Even the VSC9187 device, with its lower cross-connect rate of VT1.5, supports a maximum number of 3024 simultaneous connections. We implement a state table with 1K entries in the TCAM and SRAM. The 5-tuple used to identify a connection is stored in TCAM and the associated state information is stored in SRAM as shown in Fig. 6.

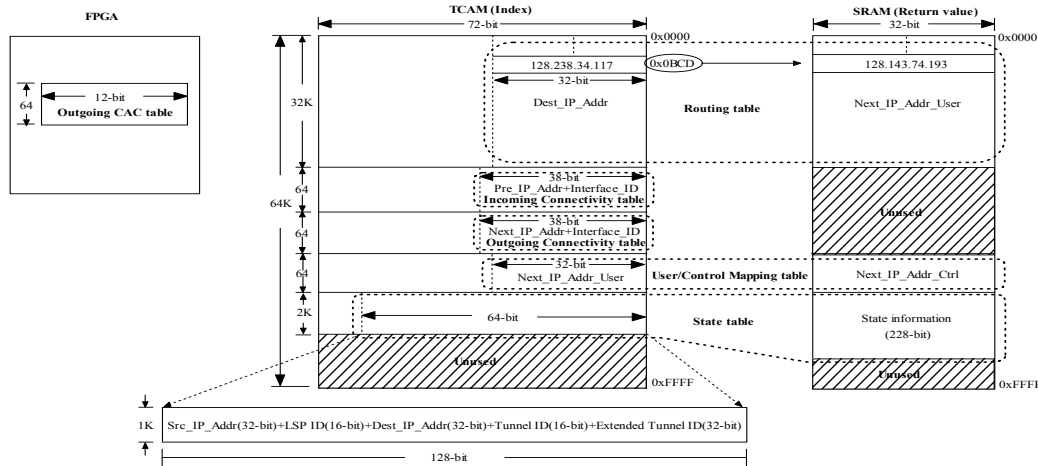


Figure 6. Organization of the data tables

C. Implementation of timers

Timers are required to support the solutions proposed in [15] for reliable message transmission. All signaling messages carry MESSAGE_ID object, which is acknowledged by MESSAGE_ID_ACK object carried in an Ack message or piggy-backed in a message in the reverse direction. If the MESSAGE_ID_ACK is not received before a retransmission timer times out at the sender, the message is retransmitted. A second timer, which we call piggyback timer, is used to hold MESSAGE_ID_ACK objects awaiting a message to be sent in the reverse to avoid unnecessary Ack messages. Ack message is generated only if this timer expires.

For each neighbor i , there is a piggyback timer p_i and a buffer organized as FIFO. Each entry in the buffer contains a MESSAGE_ID_ACK object destined for i and the associated time when the corresponding MESSAGE_ID was received. Entries in each buffer are time ordered (i.e. the head of the buffer always contains the oldest MESSAGE_ID_ACK) given the FIFO nature. Piggyback timer p_i will expire if the head entry expires. The retransmission timers are implemented in a similar manner.

D. Simulation

We developed a prototype VHDL model for the signaling hardware accelerator, used Synplify for synthesis and Xilinx ISE for place and route. The implementation uses only 14% of the FPGA resources (Xilinx XC2V3000).

We performed timing simulations of the signaling hardware accelerator using ModelSim simulator. Processing the Path message, which involves the access and updating of the data tables, takes 37 clock cycles. The time to receive a Path message is 40 clock cycles, as is the time to transmit the outgoing Path message. Receiving, Processing, and transmitting all other messages each takes no more than 40 clock cycles. Because the receiving, processing, and transmitting stages are fully pipelined, idle cycles are inserted if the stage is not ready. As a worst-case estimate, the total time for receiving, processing, and transmitting a single message consumes 120 clock cycles, which is 2.4 microseconds with a 50MHz clock. Since connection setup/release requires the handling of three signaling messages, we require a total of 7.2 microseconds per call. The call handling capacity is as high as 400,000 calls/sec because of pipelining, which allows the system to accept a new message every 40 clock cycles.

V. CONCLUSIONS

Implementation of signaling protocols in hardware poses a considerably larger number of problems than implementing user plane protocols such as IP, ATM, etc. Our implementation has demonstrated the hardware handling of functions such as parsing out various fields of TLV-encoded messages, maintaining state information, writing resource availability tables, etc., all of which are operations not encountered when processing IP headers or ATM headers. We also demonstrated the significant performance gains of hardware-accelerated

implementation of RSVP-TE, i.e., call handling within a few microseconds. Overall, this prototype implementation of RSVP-TE for GMPLS in FPGA hardware has demonstrated the potential for 100x-1000x speedup vis-a-vis software implementations on state-of-the-art processors. Currently we are building the prototype board and developing related software to set up a demonstration system.

ACKNOWLEDGMENT

We thank Integrated Device Technology, Inc. for providing us with free samples of TCAM, SRAM, and FIFO devices, and the related simulation models.

REFERENCES

- [1] T. Russell, Signaling System #7, 2nd edition, McGraw-Hill, New York, 1998.
- [2] The ATM Forum Technical Committee, "Private Network-Network Specification Interface v1.0F (PNNI 1.0)," af-pnni-0055.000, March 1996.
- [3] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels," IETF RFC 3209, Dec. 2001.
- [4] L. Berger (editor), "Generalized Multi-Protocol Label Switching (GMPLS) Signaling, Resource Reservation Protocol-Traffic Engineering (RSVP-TE) Extensions," IETF RFC 3473, Jan. 2003.
- [5] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource reSerVation Protocol (RSVP) Version 1 Functional Specification," IETF RFC 2205, Sept. 1997.
- [6] S. K. Long, R. R. Pillai, J. Biswas, T. C. Khong, "Call Performance Studies on the ATM Forum UNI Signalling," http://www.krdl.org.sg/Research/Publications/Papers/pillai_uni_perf.pdf.
- [7] P. Boyer and D. P. Tranchier, "A Reservation Principle with Applications to the ATM Traffic Control," Computer Networks and ISDN Systems Journal 24, 1992, pp. 321-334.
- [8] T. Hellstern, "Fast SVC Setup," ATM Forum Contribution 97-0380, 1997.
- [9] L. Roberts, "Inclusion of ABR in Fast Signaling," ATM Forum Contribution 97-0796, 1997.
- [10] P. Pan and H. Schulzrinne, "YESSIR: A Simple Reservation Mechanism for the Internet," IBM Research Report, RC 20967, Sept. 2, 1997.
- [11] M. Veeraraghavan, G. L. Choudhury, and M. Kshirsagar, "Implementation and Analysis of Parallel Connection Control (PCC)," Proc. of IEEE Infocom'97, Kobe, Japan, Apr. 7-11, 1997.
- [12] I. Baldine, G. N. Rouskas, H. G. Perros, D. Stevenson, "JumpStart: A Just-in-Time Signaling Architecture for WDM Burst-Switched Networks," IEEE Communication Magazine, pages 82-89, Feb. 2002.
- [13] H. Wang, M. Veeraraghavan, R. Karri, "A Hardware Implementation of a Signaling Protocol," Proc. of Opticomm 2002, July 29-Aug. 2, 2002, Boston, MA.
- [14] H. Wang, M. Veeraraghavan, T. Li, "Specification of a Subset of GMPLS RSVP-TE for a Hardware-Accelerated Implementation," Nov. 2002, <http://eeweb1.poly.edu/networks/papers/rsvp-subset.pdf>.
- [15] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, S. Molendini, "RSVP Refresh Overhead Reduction Extensions," IETF RFC 2961, April 2001.
- [16] K. Sklower, "A tree-based routing table for Berkeley unix," presented at the 1991 Winter Usenix Conf., Dallas, TX..
- [17] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink, "Small forwarding tables for fast routing lookups," Proc. of ACM SIGCOMM'97, pages 3-14, 1997.
- [18] M. Waldvogel, G. Varghese, J. Turner, B. Plattner, "Scalable High-Speed IP Routing Lookups," Computer Comm. Rev., vol. 27, pp. 25-36.