Benchmarking and Profiling the RSVP Protocol

Ying Zhao and Stephan Wong
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Delft, The Netherlands
{yingzhao,Stephan}@Dutepp0.ET.TUDelft.NL
http://ce.et.tudelft.nl

Abstract— **In this paper, we introduce a Resource Reservation Protocol (RSVP) benchmark to investigate the RSVP protocol through profiling. The benchmark is created by modifying an existing C implementation of RSVP. By running the modified benchmark in a simulation environment, two kinds of profiling results were gathered. First, performance-related results were produced to identify the time-critical functions. Second, the results on architectural characteristics of the benchmark were captured to assist the architectural design of future network processors.**

.

Index Terms—**Network processor, benchmarking, RSVP, profiling, architecture characteristics**

I. INTRODUCTION

Rapidly evolving applications on the Internet increasingly demand high performance and programmable packet processing capabilities in network devices. In early-day networks, with relatively low data rates and simple protocols, software running on general-purpose processors (GPPs) could be used to provide these packet processing capabilities. However, in modern-day networks, GPPs are no longer capable of processing packets at wire speed. Therefore, application-specific integrated circuits (ASICs) have been considered. Unfortunately, they can not flexibly adapt to increasingly rapid changes in network protocols. Consequently, Network Processors (NPs) [1] emerged, combining the flexibility of general-purpose programmable processors and performance of ASIC chips. Generally speaking, a network processor is a software programmable device with architectural features and/or special circuitry for packet processing. It is similar to a microprocessor, except that it has been optimized for use in applications involving network routing and packet processing.

In recent years, there has been a tremendous increase in the number of multimedia applications on the Internet, which not only require large amount of bandwidth, but also require specialized services to be implemented on the underlying network devices. Examples of such multimedia applications are Voice-over-IP (VoIP) and IP videoconferencing. They deliver delay-sensitive information consisting of video and audio data from end-user to end-user. Due to the real-time and multimedia features, VoIP and IP videoconferencing are being increasingly widely used. However, all IP traffic on the Internet is typically delivered on a best-effort basis. This delivery method does not address the requirement of real-time multimedia applications. To provide predictable performance, such applications require significant bandwidth with minimal delay, jitter and packet loss. Consequently, a quality of service (QoS) signaling protocol, the Resource Reservation Protocol (RSVP) [2], with the goal to address the performance needs of such applications, must be supported in network processors. In order to evaluate the performance requirements of RSVP processing, we hence introduce a new benchmark by modifying the existing implementation of RSVP distributed by ISI [3]. Using this benchmark, we can not only identify the time-critical functions in the RSVP protocol, but also gather profiling results on architectural characteristics. Both results can be utilized to assist the design of future network processors.

This paper is organized as follows. Section II describes the background on benchmarking and QoS mechanism, especially the RSVP protocol. Section III presents the detailed implementation of the RSVP benchmark from three aspects: function, environment, and measurement. Section IV discusses the profiling results for the RSVP benchmark. Section V presents the conclusions of this paper.

II. BACKGROUND

This section presents the background of this paper from three aspects: benchmarking, QoS mechanism and the RSVP protocol.

A. Benchmarking

Generally speaking, a benchmark is a kind of standard for measurements. For the particular network processor (NPs) systems, since increasingly more NP architectural designs emerge, benchmarks hence play a critical role in the evaluation and comparison of existing architectures and also can assist the future architectural development. According to [4], the results of a good benchmark should be comparable, representative, and indicative. In order to generate such results, a benchmarking methodology for network processors is proposed from three separate specifications: functional, environment, and measurement. The *Functional* specification describes the algorithmic details of the benchmark as well as required functional parameters. The *Environment* specification specifies the test-bench (input/output) of this benchmark, and the *Measurement* specification defines the performance metrics for evaluating the benchmark performance.

B. QoS Overview

Providing quality of service is a way to handle contention for network resources when the network is intended to service widely varying types of traffic. In order to provide differentiated service, QoS mechanisms provide following four modules to manage the utilization of network resources. The *Packet Classification* module is responsible for classifying incoming traffic into different classes. The *Queuing and Scheduling* module queues and forwards the packets according to the promised QoS. The *Admission Control* module determines whether the network has sufficient resources for the requested QoS, and the *Policy Control* module determines whether the application user has administrative permission to make the reservation. Based on these four QoS modules there are two popular QoS architectures, namely Differentiated Services (DiffServ) and Integrated Services (IntServ). DiffServ uses implicit signaling or the marking field, which is carried in the packet itself, to differentiate different classes of traffic, while IntServ utilizes explicit signaling to request QoS in contrast to the implicit signaling for DiffServ [9].

C. RSVP Overview

RSVP is the network control protocol that allows the data receiver to request a special end-to-end quality of service for its data flows. It is a main component of the Integrated Service architecture which can provide a both best-effort and real-time service. When an application in a host requests a specific quality of service for its traffic flow, it utilizes RSVP to deliver its request along the flow paths. RSVP is not only responsible for negotiating QoS parameters with routers to setup reservations, but it also takes charge of maintaining flow states in hosts and routers to provide the requested service.

RSVP defines a “session” to be a data flow with a particular destination and transport-layer protocol. It creates and maintains a reservation state by periodically sending control messages in both directions along the data paths for a session. RSVP messages are sent as IP datagrams and are captured and processed in each node to establish, modify, or refresh the reservation states. The Path and Resv messages are two primary RSVP message types. A path message is initiated by a sender and travels along the direction of data flow, addressed to the multicast or unicast destination address of the session, to create a path state in each node. A Resv message is sent periodically by each receiver host toward the senders to request reservations. In addition, RSVP can use teardown messages (PathTear, ResvTear) to remove the path and reservation states explicitly without waiting for the cleanup timeout. There are also error (PathErr and ResvErr) and confirmation (ResvConf) messages to report errors in processing corresponding Path and Resv messages, and indicate the confirmation of successful reservation. As defined in [5], an RSVP message consists of a common header, followed by a body with a variable number of objects. Figure 1 and Figure 2 illustrate the format of common header and object of RSVP message.

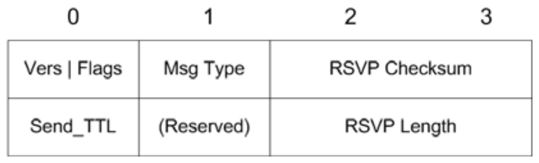

Fig. 1. An RSVP Common Header Format

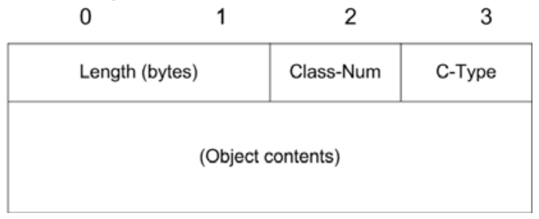

Fig. 2. An RSVP Object Format

To maintain the path and reservation states in routers or hosts, RSVP takes a “soft state” approach. The RSVP soft state is created and must be periodically refreshed by Path and Resv messages. If no matching refresh messages arrive before the expiration of a “cleanup timeout” interval, the state will be deleted. By using a soft state, RSVP can easily handle changing multicast memberships and flow routes.

III. RSVP BENCHMARK

According to the benchmarking methodology described in [4], benchmark implementation can be divided into three parts. The first one is to determine possible functions, which are to be realized in the benchmark, and rewrite those functions to make them measurable. The second part is called environment, it specifies the simulation environment for the benchmark and also designs input data for the benchmark. Third, verifying correctness of the benchmark and defining profiling measurements, that is the metrics for evaluating performance.

A. Function Specification

The RSVP processing in a router is responsible for receiving RSVP messages, parsing these messages for their correctness, and doing corresponding operations. In this benchmark, four functions are investigated.

1) Input Message. Due to incompatibility of SOCKET functions in the simulation environment (SimpleScalar Tool Set, Version 3.0), we manually define internal RSVP messages for RSVP processing routine as input data, and store them in an input buffer. The RSVP processing routine then can take messages from the input buffer to perform corresponding operations. For each RSVP input message, a common header must be constructed first, then the following variable objects should be defined, referring to [5] for field definitions, like Class-Num, C-Type values in the standard object header, and also parameters in object contents. For input message types, suppose there are no Error messages in the input data, the remainder types of messages should be generated following a specific sequence. First generate a Path message, then a Resv message, followed by RConf and Teardown messages. Note that this sequence cannot be freely altered because the latter RSVP messages are dependent to previous ones. Furthermore, in order to maintain a reservation soft state in routers, refresh messages also need to be generated to emulate the periodical sending from senders and receivers. Since it is not possible to mimic every case in real world, we only take the VoIP application case as an example, and utilize the statistic data for conversations on telephone to represent it. We generate the number of refresh messages from 0 to 6 because in reality, the duration of more than 89.5% conservations on telephone are normally within 3 minutes [6] and the default refresh interval defined in RSVP protocol is 30 seconds. Finally, we can determine that in average 4 Path messages, 4 Resv messages, 1 RConf message, 1 RTear message, and 1 PTear message are needed for one complete reservation procedure of one session.

2) *Parse Message.* After receiving the RSVP messages, a processing routine is called to parse messages and do corresponding operations including version checking and checksum verification [7] of incoming packet, performing object format checking, parsing the sequence of objects in the message, etc.

3) *Process Message.* Decompose arriving RSVP messages into components and process depending upon the type of message. Several common sub-functions can be abstracted from processing procedures for each type of RSVP message, as depicted in Figure 3. It is easy to observe that, during the processing of each type of RSVP messages, appropriate state blocks have to be located, created, modified, or deleted. The primary function of message processing is "state maintenance". Usually, before state maintenance, the appropriate SESSION object has to be determined and created if no matching SESSION is found. We call this function "session". After state maintenance, immediate refresh messages may be generated and sent, or copy of the received messages may be forwarded to particular destinations. This function is called "generate and send message".

4) *Timeout Processing.* According to the soft state approach, each state block is associated with a timer and deleted upon timeout or receiving a tear message. Periodic refresh messages restart the timer. The actions resulting from a timeout are similar to those when receiving a PathTear or ResvTear message.

B. *Environment Specification*

The RSVP benchmark takes messages from an input message buffer. The output of this benchmark are also RSVP messages, either new generated refresh messages or copies of original messages that need to be forwarded. The cycle-accurate sim-outorder simulator from the SimpleScalar Tool Set (Version 3.0) [8] is utilized to run the RSVP benchmark.

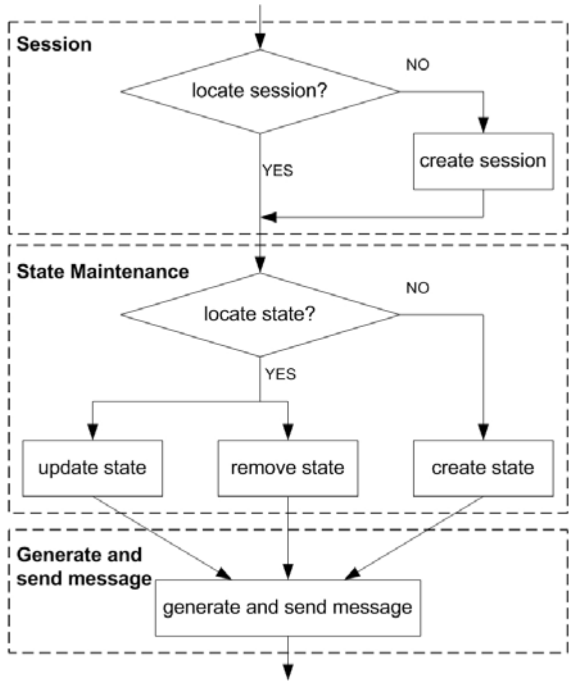

Fig. 3. The flow chart of "Process Message" function

C. *Measurement Specification*

Three aspects are described in the measurement. They are verification of the benchmark, the performance metric and the architectural characteristics.

1) Benchmark Verification. Before measuring the performance of the benchmark, its correctness should be checked. We compare input RSVP messages with output messages generated from the benchmark, and also check the status of reservation states, according to the message processing rules described in [10].

2*) Performance Metric*. The number of clock cycles is utilized as performance metric to evaluate the performance of the benchmark. In order to determine the clock cycles for certain functions, the simulator is extended with an extra operation called instruction annotation detection. At the same time, two "NOP/a" instructions are introduced to signify the start of a function and the end of a function, respectively.

3) Architectural Characteristics. The architectural characteristics of the benchmark are explored to better understand the features of the benchmark, and hence are used to differentiate between different benchmarks. These key architectural characteristics include:

--*Instruction Level Parallelism* (ILP), measured in instruction per cycle (IPC). A high IPC means the instruction dependency with a program is low.

--*Branch Prediction Accuracy*, measured in branch address prediction rate (APR) and branch direction prediction rate (DPR).

--*Instruction Distribution*. We only concentrate on the load/store and branch operations. High load/store frequency shows the application has a data-intensive nature.

--*Cache Behavior*, measured in the number of cache accesses and miss ratios. Since caches have profound effects on the performance of processors, we can also study the cache behavior of the benchmark by varying cache configurations, such as cache sizes, block sizes, set associativity, hit latency, etc.

Profiling results on these four aspects were gathered to investigate the architectural characteristics of the RSVP benchmark.

IV. EXPERIMENT EVALUATION

In this section, benchmark results on performance and architectural characteristics are discussed. Before running the benchmark on a simulation environment, we made several assumptions. We assumed the benchmark works for RSVP processing in a unicast environment as an RSVP-capable router. Based on the understanding of the RSVP protocol and statistics gathered from the real world, three synthetic input scenarios for the benchmark are defined. Scenario 1 represents the most common situation and consists of Path, Resv, Tear, and RConf messages generated. Scenario 2 and Scenario 3 are both extreme cases. They consist of only Path-related messages and Resv-related messages, respectively.

A. *Results on Functions*

The performance results in clock cycles for different input scenarios are depicted in Figure 4. It can be observed from the figure that the "process message" (process) function consumes the largest amount of execution time, because most operations for setting up reservations are included in this function. The "parse message" (parse) function is the secondary time-consuming one, while the "time-out processing" (time-out) function consumes the least amount of execution time.

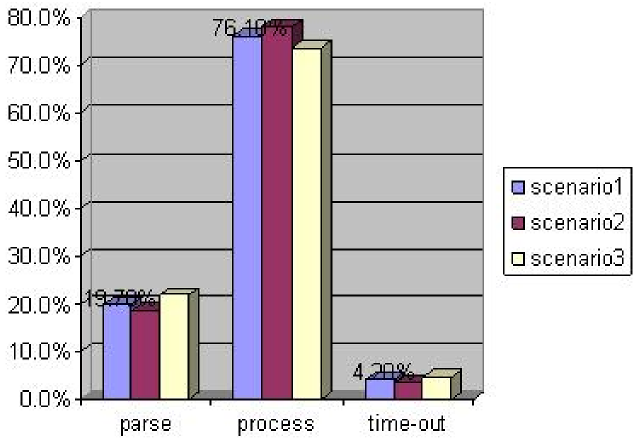

Fig. 4. The combined RSVP benchmark results for main functions

The performance results of sub-functions in "process message" for different input scenarios are depicted in Figure 5. We can observe that the "state maintenance" (state) and "generate and send message" (generate) functions consume the major part of the execution time due to the fact that the processing for these two functions are obviously more than the processing for the session function. Another observation is that the percentages of cycles for "generate and send message" function change greatly in three input scenarios, from 27.8% in scenario 3 to 45.2% in scenario 2. This is mainly due to looking up the routing information for the destination address, which is only called when processing Path-related message.

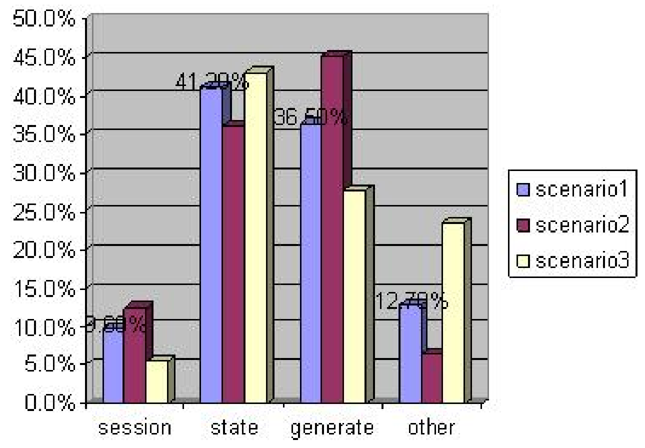

Fig. 5. The combined RSVP benchmark results for sub-functions in "process message" function

B. Results on Architectural Characteristics

The number of input messages is set to 10000, and all three input scenarios are tested with the default parameters[1] [11] of the *sim-outorder* simulator. Table 1 presents the simulation results on instruction level parallelism and branch prediction accuracy. Table 2 presents the simulation results on instruction distribution, we can observe that the load and store instructions occupy about 42% of the total instructions executed. This reveals the benchmark has a data-intensive nature. Table 3 presents the simulation results on cache behavior. Here il1 represents the level 1 instruction cache, dl1 represents the level 1 data cache, and ul2 represents the unified level 2 cache.

It is well-known that the cache size can greatly affect the performance of processors. Therefore, we vary the sizes of level

1 instruction cache, level 1 data cache and level 2 unified caches, respectively, to investigate the caches impact on this benchmark. The profiling results show that the instruction cache miss ratios are more affected by the cache size. When increasing the level 1 instruction cache size, the corresponding IPC value also significantly increases. However, the size of

[1] Default il1: 512-set, direct-mapped, 32-byte line, LRU, 1-cycle hit latency, 16KB size. Default dl1: 128-set, 4-way, 32-byte line, LRU, 1-cycle hit latency, 16KB size.Default ul2: 1024-set, 4-way, 64-byte line, LRU, 6-cycle hit latency, 256KB size

level 1 data cache and level 2 unified caches does not have much impact on the IPC value. Subsequently, the experiment of reducing the level 2 cache hit latency, i.e., decreasing the miss penalty of level 1 cache also proves that the instruction cache has a more significant impact on the performance than the data cache. Nevertheless, this conclusion does not conflict with previous results that the benchmark has a data-intensive nature because the data cache access address mainly concentrate on four places but not evenly distributed. Moreover, the experiments of changing cache block size and set associativity reveal that the 128KB instruction cache with 64-byte block and 4-way set-associativity provides the best performance for this benchmark.

Table 1. IPC and branch prediction values for the benchmark with three input scenarios

Input	# of cycles (M)	IPC	APR (%)	DPR (%)
Scenario1	69.7	0.93	89.78	90.94
Scenario2	71.6	0.98	91.27	91.94
Scenario3	68.0	0.90	89.87	90.85

Table 2. Instruction distribution for the benchmark with three input scenarios

Input	# of Inst. (M)	Load (%)	Store (%)	Branch (%)
Scenario1	65.1	27.2	14.9	18.7
Scenario2	70.2	26.6	15.2	18.4
Scenario3	61.2	27.8	14.5	18.9

Table 3. Cache behavior for the benchmark with three input scenarios

Input	il1 acc. (M)	il1 miss (%)	dl1 acc. (M)	dl1 miss (%)	ul2 miss (%)
Scenario1	80.8	8.22	25.4	0.40	0.34
Scenario2	86.2	8.01	28.7	0.36	0.26
Scenario3	75.6	8.70	22.6	0.39	0.36

V. Conclusions

In this paper, we first argued that the RSVP protocol is required in varying network processors and briefly introduced the newly-proposed benchmarking methodology. Subsequently, we described the implementation of the RSVP benchmark from function, environment and measurement aspects. Finally, we discussed the profiling results of the benchmark from two aspects. The performance-related results can be utilized to identify the time-critical functions. The biggest contributor to the total cycles of the RSVP benchmark is the "process message" function, which occupies more than 73% of the total execution cycles. For the sub-functions of "process message", the "state maintenance" and "generate and send message" functions occupy the major part of the total cycles for the "process message" function, which is more than 70%. On the other hand, the results on architectural characteristics were presented by looking up the instruction level parallelism, branch prediction accuracy, instruction distribution and cache behavior. Further investigation on the cache organization shows that the instruction cache has a more significant impact on the benchmark performance than the data cache, and the 128KB instruction cache with 64-byte block and 4-way set-associative will provide the best performance.

References

[1] Niraj Shah, *Understanding Network Processors (Version 1.0)*, (2001).

[2] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala, *RSVP: A New ReSerVation Protocol*, IEEE Network Magazine (1993), no. 5, 8-18.

[3] *RSVP ISI Distribution*, [Online]. Available: http://www.isi.edu/div7/rsvp/release.html.

[4] Mel Tsai, Chidamber Kulkarni, Christian Sauer, Niraj Shah, and Kurt Keutzer, *A Benchmarking Methodology for Network Processors*, Proceedings of 1st Network Processor Workshop (NP-1), 8th International Symposium on High Performance Computer Architectures (HPCA), 2002.

[5] R.Berger, L.Zhang, S.Berson, S.Herzog, and S.Jamin, *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*, Tech. report, Network Working Group, RFC 2205, 1997.

[6] *Duration Distribution for Talks on the Phone*, [Online]. Available: http://www.people.com.cn/GB/it/50/145/20010226/403473.html.

[7] R.Braden and D.Borman abd C. Partridge, *Computing the Internet Checksum*, Tech. report, Network Working Group, RFC 1071, 1998.

[8] Doug Burger and Todd M. Austin, The SimpleScalar Tool Set, Version 2.0, Tech. report, Computer Sciences Department, University of Wisconsin-Madison, 1997.

[9] R.Braden , D.Clark, and S.Shenker, *Integrated Services in the Internet Architecture: An Overview*, Tech. report, Network Working Group, RFC 1633, 1994.

[10] R.Braden, L.Zhang, *Resource ReSerVation Protocol (RSVP) -- Version 1 Message Processing Rules.* Network Working Group, RFC 2209, Septermber 1997.

[11] Matthew Postiff, David Greene, Charles Lefurgy, Dave Helder, and Trevor Modge, *The MIRV SimpleScalar/PISA Compiler*, Tech. report, EECS Department, University of Michigan, 2000.