Real-Time Network Processing: An Investigation

Yunfei Wu, Stephan Wong, and Stamatis Vassiliadis
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Delft, The Netherlands
{yunfei,stephan,stamatis}@Dutepp0.ET.TUDelft.NL
http://ce.et.tudelft.nl

Abstract—**In this paper, we introduce a benchmark suite on RTP/RTCP processing. The suite consists of three benchmarks, namely RTP Sender, RTP Receiver and RTCP Processing. For each benchmark, three aspects are specified: functions, measurements and environments. After implementing the benchmarks, we perform profiling in order to determine the time-critical functions and investigate the architectural characteristics. Finally, the results on architectural characteristics show that the proposed benchmarks differ significantly from MediaBench and NetBench.**

Keywords—**Benchmarking, real-time network processing, RTP/RTCP, profiling, architectural characteristics.**

I. Introduction

The continuous evolution of the Internet requires support from network devices that are capable of handling all the necessary services, applications, etc.. This imposes two requirements on the network devices: performance and flexibility. Performance is needed, because network devices are expected to process data at wire-speed in order to eliminate network bottlenecks. Flexibility is needed, because network devices are expected to be easily adaptable to support emerging applications. In designing network devices for flexibility, an obvious choice would be to utilize general-purpose processors (GPPs) which can be adapted to rapidly changing network protocols. However, they usually lack the performance to handle data at wire-speed. Traditionally, as the performance became increasingly more important, application-specific integrated circuits (ASICs) were introduced and substituted GPPs. However, ASICs lack the flexibility to be easily updated in order to support new features. Consequently, a new kind of processor is needed to meet the two requirements. Such a need has sparked the emergence of the network processor.

A network processor (NP) is a programmable device incorporating specialized hardware designed specifically to process data at wire-speed. Network processors can provide speed improvements through architectures, such as parallel distributed processing and pipeline processing designs. The programmability of network processors can enable easier migration to new protocols and technologies. While network processors are designed having both flexibility and performance in mind, there is still a broad spectrum of trade-offs between these two requirements. The advantage is that it allows network processor vendors to distinguish their products from others by targeting slightly different application areas. The main disadvantage is that the existence of many architectures and implementations of network processors makes it difficult to compare them in terms of performance. Therefore, a key challenge is to find an adequate way to evaluate the performance of the heterogeneous collection of network processors (both current and possible future ones). Consequently, benchmarks are needed to measure the performance of network processors. Currently, several network processing benchmarks have been introduced that mainly focus on some specific applications in the networking domain. However, there is still a need for a new set of benchmarks in order to more accurately reflect the constantly evolving field of network processing. More specifically, in this paper we create a benchmark suite that allows the investigation of network processing on real-time delivery that has not been introduced in existing benchmarks. We also perform profiling on the benchmarks in order to find the time-critical functions in each benchmark, which could be implemented in hardware in the future to obtain a performance gain. Finally, we investigate the architectural characteristics on the benchmarks and compare them with the results from NetBench [6] and MediaBench [3] to understand the features of the created benchmarks.

This paper is organized as follows. Section II describes the background on benchmarking and the protocols for real-time delivery. Section III presents the implementation of the benchmarks on real-time delivery from three aspects: function, measurement and environment. Section IV describes the benchmarks results. Section V presents the conclusions of the paper.

II. BACKGROUND

This section presents two backgrounds on the topic of this paper: benchmarking and the protocols for real-time network processing.

A. Benchmarking

Simply put, a benchmark is a standard for judging system performance in a target application. Generally, a good benchmark helps to measure system performance in a deterministic and reproducible manner. Like the well-known SPEC benchmarks [13], they have been widely utilized to evaluate the performance of modern computer systems. However, they are not suitable for benchmarking network processors since network processors target specific applications related networking. [9] discusses the requirements and challenges of benchmarks for network processors. Recently, some benchmarks for network processors, NetBench [6], CommBench [1], EEMBC [5], and NPF [10], have been introduced. There is still headroom for benchmarking network processing due to the following reason. With new applications emerging, no benchmark exists to evaluate the performance of network processors on these new applications. It is necessary for benchmarks to cover some of the untargeted applications that the existing benchmarks do not cover.

B. Real-Time Transport Protocol

As the need of multimedia services grows, it is inevitable that the convergence of voice and data will play an important role in future networks. Currently, Voice-over-IP (VoIP) is viewed as an attractive and effective technology. However, the IP network only provides "best-effort" services causing variable delays, packet duplicates, and packet losses. This is usually not a big problem for data applications. However, voice data are delay sensitive meaning that voice packets have to be delivered timely and long packet delivery delays must be avoided. In order to ensure timely voice data delivery, additional protocol support is required. Such protocol must handle two main problems of the voice data delivery over the IP network: out-of-order delivery and jitter.

The Real-Time Transport Protocol (RTP) [11] is designed to provide end-to-end delivery services for data with real-time characteristics. The data transport is augmented by the RTP control protocol (RTCP) to allow monitoring of the data delivery.

The RTP protocol handles the out-of-order problems by assigning a sequence number to each voice packet and delay problems by assigning a timestamp to the packet. The sequence number enables the receiver to process the

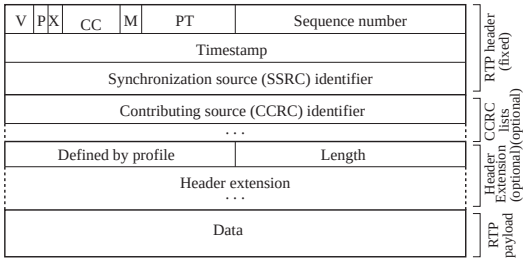

Fig. 1. RTP packet.

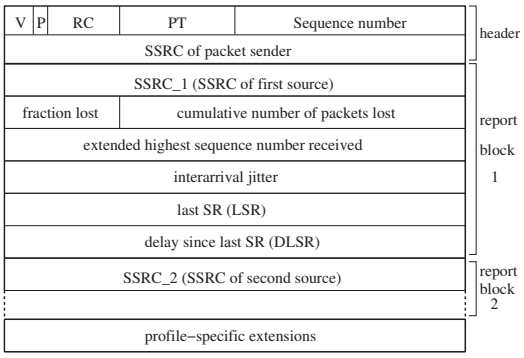

Fig. 2. RTCP Receiver Report (RR) packet.

packets in the same order as they were sent. It also allows the receiver to detect packet losses and duplicates. Once ordered, the receiver determines using the timestamp at which time the data in the packet should be played back. Figure 1 depicts the structure of an RTP packet. The first twelve bytes must be present in every RTP packet, while the CCRC list and RTP header extension are optional. Specific details regarding the use of these header fields, RTP and its profiles are described in [11] and [12].

The RTCP protocol works in conjunction with RTP by providing control information. An important RTCP packet is the RTCP receiver report (RR) which allows all receivers to exchange information on reception conditions and to adapt their reporting rates to avoid using excessive bandwidth and overwhelming the sender. Figure 2 depicts the structure of an RTCP RR. It mainly consists of two sections possibly followed by a third profile-specific extension section if defined. The first section (header) is 8 bytes long. The second section contain zero or more reception report block. Each reception report block conveys statistics on the reception of RTP packets from a single synchronization source.

III. BENCHMARKING

The RTP protocol is responsible for sending and receiving RTP packets and the RTCP protocol is responsible for providing control information. Therefore, the benchmark suite for the RTP and RTCP processing consists of an RTP Sender benchmark, an RTP Receiver benchmark and an

RTCP Processing benchmark. The implementation of the benchmarks are based on existing software codes [7], [4] and [11]. By using the methodology described in [8], each benchmark is highlighted from three aspects. First, the function aspect specifies the core algorithms used in each benchmark, all necessary functions implemented in the benchmark, and the manner to implement these functions. Second, the measurement aspect specifies the verification of the benchmarks and the metrics used to investigate the benchmarks. Third, the environment aspect specifies the interface (input/output) of each benchmark and simulation environment on the benchmarks. For all three benchmarks, the same measurement and the same simulation environment are utilized.

This section is organized as follows. Section III-A describes function specification and environment specification on interface of the RTP Sender benchmark. Section III-B describes the two aspects of the RTP Receiver benchmark. Section III-C describes the two aspects of the RTCP Processing benchmark. Section III-D describes the common aspects for the three benchmarks on measurement specification and simulation specification.

A. RTP Sender

Function specification: Three functions are investigated in the RTP Sender benchmark:

- *Reading data from input* is to obtain a voice data unit as an RTP payload and to allocate a buffer to the data unit.
- *Encapsulating an RTP packet* consists of building an RTP header, prepending the header to the RTP payload to be an RTP packet, and increasing the timestamp and sequence number for the next RTP packet.
- *Writing the RTP packet* is performed in two ways: writing to the memory for the simulation and writing to the hard disk for generating the input of the RTP Receiver benchmark.

Environment specification - interface: The RTP Sender benchmark takes an audio file as the input, the size of which is $4MB$. The input buffer is set into $160\ bytes$ for $20\ ms$ voice. The output buffer is set into $10000\ bytes$ for an RTP packet.

B. RTP Receiver

Function specification: Four functions are investigated in the RTP Receiver benchmark:

- *Input buffer management* The input buffer is utilized to store incoming packets for further processing. If the buffer is full, the new incoming packet will be placed at the beginning of the buffer even if the old packet in the beginning of the buffer has not been played out.
- *Data parsing* A packet is received as a format of a string. This function is to convert this string into the structure of an RTP packet. It takes a pointer to the string saved in the input buffer and returns a pointer to the RTP packet structure.
- *Statistics updating* It includes two procedures: first, processing the sequence number of each coming RTP packet and subsequently, estimating the interarrival jitter. The sequence number processing is done by the algorithm defined in [11]. The interarrival jitter is estimated by the following equations:

$$D_i = (R_i - R_{i-1}) - (S_i - S_{i-1}) \quad (1)$$

Where D_i denotes jitter estimation for the ith packet, R_i and R_{i-1} denote the time of arrival in RTP timestamp units for the ith and $(i-1)$th packet, respectively. S_i and S_{i-1} denote RTP timestamp of the ith and $(i-1)$th packet, respectively.

$$J_i = 15/16 J_{i-1} + 1/16|D_i| \quad (2)$$

Where J_i denotes temporal average of the jitter for the ith packet, J_{i-1} denotes the one for the $(i-1)$th packet.

- *Queue management* It is to manage the RTP packets moving in and out of the queue. It is a sorted queue, known as *the jitter buffer*, based on the sequence number of RTP packets. It includes two procedures: dequeuing and inserting. Dequeuing is done by first-in-first-out. To accommodate jitter, packets must not start to be dequeued until the buffer has been filled past a threshold. Inserting is to compare the sequence number of the incoming packet with sequence numbers of all other packets in the queue and to arrange a location where the packet should be inserted. Therefore, packets can be inserted in the middle of the queue.

Environment specification - interface: The RTP Receiver takes the output of the RTP Sender benchmark as the input. The input buffer is set into $64KB$. The output of the RTP Receiver benchmark is written into memory for simulation and into hard disk for verifying the correctness of the benchmark.

C. RTCP Processing

Function specification: In RTCP Processing benchmark, two functions are implemented: building an RTCP receiver report (RR) and computing the RTCP transmission interval.

- *Building an RTCP RR packets* includes two aspects: generating an RTCP header and calculating each element (depicted in Figure 2) in the receiver block.

• *Computing the RTCP transmission interval* is performed by the following equation:

$$T_d = max(T_{min}, CL(t)) \quad (3)$$

Where T_{min} is $2.5s$ for the initial packet, and $5s$ for all other packets, C is the average RTCP packet size divided by 5% of the session bandwidth, $L(t)$ represents the number of users within a multicast group that have been heard from at time t, and the initial value at time 0, $L(0) = 1$ when the user joins the group.

Environment specification - interface: The RTCP Processing benchmark works in conjunction with the RTP Receiver benchmark, hence the same input data are used in the RTP Receiver benchmark. The input buffer of the RTCP Processing benchmark is set into $100\,bytes$ that can contain one report block in an RTCP RR packet. The output of the benchmark is stored in the memory.

D. Measurement And Simulation Environment

This section specifies the common aspects for the created benchmarks in this paper on measurement and simulation environment.

Measurement specification: It describes three aspects in measurement. First, *verification of benchmarks* has to be performed before measuring the performance of the benchmarks. The correctness of the benchmarks are verified by comparing the input of the RTP Sender benchmark and the output of the RTP Receiver benchmark. If they are the same, the benchmarks are correct since the final result at the receiver is to obtain the same voice signal with the one at the sender. Second, *the performance metric* for evaluating the performance of all benchmarks is the number of clock cycles, which captures how fast the benchmark is executed and which functions in the benchmark are time-critical. For the time-critical function, it could be implemented in specifical hardware to obtain the performance gain. Third, *the architectural characteristics* for the benchmarks are also investigated. These characteristics are as follows:

• Instruction Level Parallelism (ILP) is measured by instruction per cycle (IPC) which shows data-level parallelism and dependency of the instructions. IPC value is high when the dependency of the instruction with a program is low.

• Branch Prediction Accuracy is measured by branch address-prediction rate (APR) and branch direction-prediction rata (DPR).

• Instruction distribution is measured by determining the frequency of load instructions, store instructions, and branch instructions.

• Cache behavior is measured by the number of cache accesses and miss ratios. Both data cache and instruction cache are simulated: dl1 stands for first level data cache, il1 stands for first level instruction cache and l2 stands for the unified second level caches.

The four architectural characteristics on the created benchmarks mentioned above are compared with the applications from NetBench and MediaBench to understand the features of the benchmarks.

Simulation environment: The *sim-outorder* simulator from the SimpleScalar tool set (Version 3.0) [2] is utilized for all benchmarks. This simulator allows us to measure the number of clock cycles spent in each function of each benchmark and to perform profiling to determine the time critical functions. The measurement is performed by utilizing instruction annotations in the *sim-outorder* simulator. We have introduced a NOP instruction to signify the start of a function and another NOP to signify the end of the function. More specifically, the overall simulation clock cycle (called sim_cycle) is noted in both cases. By subtraction the starting sim_cycle from the ending sim_cycle, the number of clock cycles spent in executing a given function can be calculated.

IV. Simulation Results

In this section, we discuss the benchmarks results from two aspects: performance and architectural characteristics. In order to simulate the benchmarks, we made the following assumptions. The *sim-outorder* simulator entails a 2-way superscalar processor with $64\ KB$ of direct-mapped level 1 ($L1$) data and instruction caches and a $1MB$ unified level 2 ($L2$) caches. The $L1$ and $L2$ cache latencies are set to 1 and 6 cycles, respectively.

Profiling results: The performance results present the number of clock cycles for each function in each benchmark in relation to the total cycles. This allows us to determine the time-critical functions in the benchmarks. The results are depicted in Figures 3, 4, and 5. In order to simulate a more realistic networking environment in which packets may arrive in-order or out-of-order, three different data sets are utilized in the RTP Receiver benchmark (depicted in Figure 4[1]: R_input1 denotes packets are in order, R_input2 denotes packets are slightly out-of-order, and R_input3 denotes packets are largely out-of-order). Figure 3 illustrates that the biggest contributor to the total cycles for the RTP Sender benchmark is the encapsulation function (encap), which takes more than 70% of the total cycles. Figure 4 illustrates that the two biggest contributors for the RTP Receiver benchmark are the statistics updating

[1] Only the biggest value in each function is marked in Figure 4.

function and the data parsing function, which take about 25% and 23% of the total cycles, respectively. Figure 5 illustrates that the biggest contributor for the RTCP Processing benchmark is building RTCP receiver report function (block), which takes about 55% of the total RTCP Processing cycles.

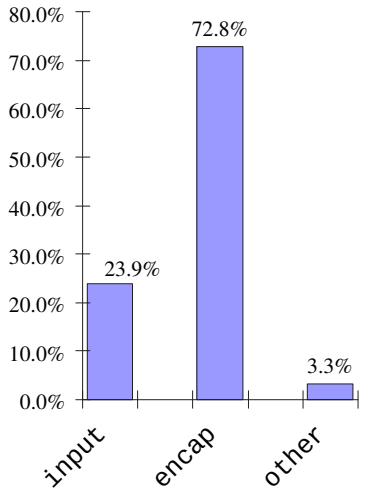

Fig. 3. RTP Sender benchmark results.

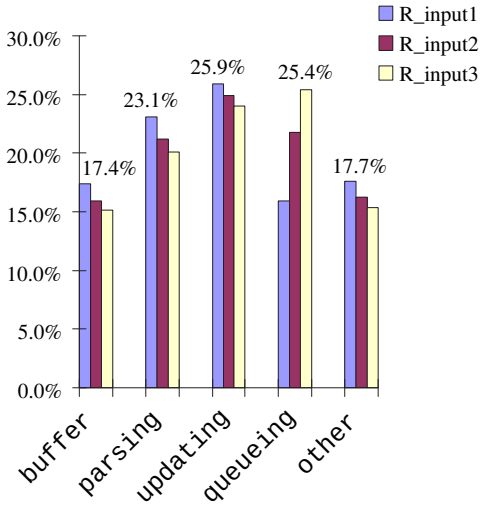

Fig. 4. RTP Receiver benchmark results.

Results on architectural characteristics: The four architectural characteristics are highlighted in Tables I, II, and III. The results on NetBench and MediaBench are the averages of the results presented in [6] due to the large mix of different small and big benchmarks. We are able to use the averages over the results, because there are no significantly big differences between the results, except for the number of cycles (*# of cycles*), the number of instructions (*# of inst.*), and the number of il1 and dl1 accesses (*il1 acc.* and *dl1 acc.*). Therefore, no comparison is made based on these values and they are only shown for completeness. Table I highlights the results on instruction level parallelism and branch prediction accuracy, Table II highlights the results on instruction distribution, and Table III highlights the results on cache behavior. In Tables I, II, and III, 'Average' denotes the arithmetic mean of the RTP Sender, the RTP Receiver and the RTCP Processing benchmark results. The results presented in Table I show that the average IPC of the RTP/RTCP benchmarks is 14.5% and 31% higher than NetBench and MediaBench, respectively. The average APR and the average DPR of the RTP/RTCP benchmarks is 4.9% and 4% higher than MediaBench, respectively. The results presented in Table II show that the average load/strore instruction frequency of the RTP/RTCP benchmarks is higher than NetBench and MediaBench. The results presented in Table III show that the RTP/RTCP benchmarks has a better performance in cache behavior than NetBench and MediaBench because the first level data/instuction caches miss ratios (dl1 and il1) and the second level unified cache miss ratio (l2) are less than the ones of NetBench and MediaBench.

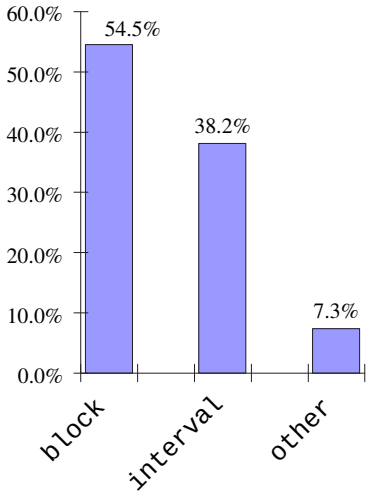

Fig. 5. RTCP Processing benchmark results.

	# of cycles (M)	IPC	APR (%)	DPR (%)
RTP Sender	11.4	1.99	95	95.1
RTP Receiver	16.5	1.9	92.2	92.2
RTCP	33	1.84	93.3	93.3
Average	20.3	1.9	93.5	93.5
NetBench	207	1.66	93.9	94.2
MediaBench	280	1.45	89.1	89.9

TABLE I
COMPARISON IN IPC AND BRANCH PREDICTION VALUES BETWEEN RTP/RTCP BENCHMARKS WITH NETBENCH AND MEDIABENCH.

	# of inst. (M)	load (%)	store (%)	branch (%)
RTP Sender	22.8	31.3	21.9	12
RTP Receiver	31.9	30.0	18.8	14.3
RTCP	43.6	29	17.2	1.3
Average	32.8	30.1	19.3	9.2
NetBench	359	27.7		7.2
MediaBench	408	19.8		11.3

TABLE II
COMPARISON IN INSTRUCTION DISTRIBUTION BETWEEN RTP/RTCP BENCHMARKS WITH NETBENCH AND MEDIABENCH.

	il1 acc. (M)	il1 miss ratio (%)	dl1 acc. (M)	dl1 miss ratio (%)	l2 miss ratio (%)
RTP Sender	23.9	0	11.9	0	4.6
RTP Receiver	33.9	0.0	14.9	0.4	0.8
RTCP	45.1	0	19.2	0.3	0.8
Average	34.3	0.0	15.3	0.2	2.1
NetBench	400	0.05	140	0.8	9.7
MediaBench	519	0.4	86	1.8	14.8

TABLE III
COMPARISON IN CACHE BEHAVIOR BETWEEN RTP/RTCP BENCHMARKS WITH NETBENCH AND MEDIABENCH.

The results on architectural characteristics show that the RTP/RTCP processing is significantly different from the multimedia processing (MediaBench), which means that it is necessary to create benchmarks for network processing. The results also show that the RTP/RTCP processing has some common characteristics with NetBench since they all focus on processing in the same networking domain. However, they have some different characteristics since they target on different network processing.

V. CONCLUSIONS

It has been argued in this paper that benchmarking network processing in the higher layer of the TCP/IP model is important since VoIP and quality of services are becoming increasingly popular technologies. It was briefly discussed how RTP/RTCP was utilized to provide end-to-end delivery services for data with real-time characteristics. Subsequently, benchmarks for RTP/RTCP processing were introduced and discussed from three aspects: functions, measurements and environments. Consecutively, these benchmarks were run in a simulation environment in order to obtain profiling information and to investigate architectural characteristics. The profiling results show that the biggest contributor to the total cycles of the RTP Sender Benchmark is the encapsulation function, which takes more than 70% of the total cycles. The biggest contributors to the RTP Receiver benchmarks are the statistics function and data parsing function which take about 25% and 23% of the total cycles. The biggest contributor to the RTCP Processing benchmark is the building RTCP receiver report function which takes about 55% of the total cycles. Finally, the comparison on architectural characteristics between the proposed benchmarks in this paper with other network processing or multimedia benchmarks shows that we differ from them in instruction level parallelism, instruction distribution and cache behavior. These results highlight that we could achieve a 14.5% and 31% higher parallelism than NetBench and MediaBench, respectively. We could also achieve lower first level data/instruction caches miss ratio and lower second level unified caches miss ratios.

REFERENCES

[1] Tilman Wolf and Mark Franklin. CommBench - A Telecommunications Benchmark for Network Processors. `http://ccrc.wustl.edu/ wolf/cb/`, 1999.

[2] Austin, Todd M. and Burger, D. *Simplescalar Tutorial*. University of Wisconsin, 1997.

[3] Chunho Lee and Miodrag Potkonjak and William H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems. `http://www.cs.ucla.edu/~leec/mediabench/`.

[4] Comer, Douglas E. *Internetworking with TCP/IP (Volumn 3)*. Prentice Hall, 2000.

[5] Embedded Microprocessor Benchmark Consortium. `http://eembc.org/`.

[6] Gokhan Memik, B. Mangione-Smith and W. Hu . NetBench: A Benchmarking Suite for Network Processors. Technical report, Compiler and Architecture Research Group (CARES) at University of California, November 2001.

[7] Jonathan Lennox, Jonathan Rosenberg and Dan Rubenstein, editor. *RTP Library*. Lucent Technologies, 1998.

[8] Mei Tsai, Chidamber Kulkarni, Christian Sauer, Niraj Shah, Kurt Keutzer. A Benchmarking Methodology for Network Processors. In 1^{st} *Network Processor Workshop (NP-1), 8th International Symposium on High Performance Computer Architectures (HPCA)*, 2002.

[9] A. Nemirovsky. Towards characterizing network processors: Needs and challenges. Technical report, Xstream Logic Inc., 2000.

[10] Prachant R. Chandra and Seow Yin Lim, editor. *Framework for Benchmarking Network Processor (Revision 1.0)*. Network Processing Forum, 2002.

[11] Schulzrinne, H. A Tranport Protocol for Real-Time Application. In *RFC 1889*, January 1996.

[12] Schulzrinne, H. RTP Profile for Audio and Video Conference with Minimal Control. In *RFC 1890*, January 1996.

[13] SPEC. `http://www.specbench.org`.