Time Driven Priority Router Implementation and First Experiments

Mario Baldi, Guido Marchetto, Fulvio Risso
Dipartimento di Informatica, Politecnico di Torino, Italy
Email: {baldi, marchetto, risso}@polito.it

Giulio Galante, Riccardo Scopigno, Federico Stirano
Networking Lab, Istituto Superiore Mario Boella, Torino, Italy
Email: {galante, scopigno, stirano}@ismb.it

***Abstract*— This paper reports on the implementation of Time-Driven Priority (TDP) scheduling on a FreeBSD platform. This work is part of a TDP prototyping and demonstration project aimed at showing the implications of TDP deployment in packet-switched networks, especially benefits for real-time applications. This paper focuses on practical aspects related to the implementation of the technology on a Personal Computer (PC)-based router and presents the experimental results obtained on a testbed network.**

The basic building blocks of a TDP router are described and implementation choices are discussed. The relevant results achieved and here presented can be categorized into two types: *qualitative* results, including the successful integration of all needed blocks and the insight obtained on the complexity related to the implementation of a TDP router, and *quantitative* ones, including measures of achievable network utilization and of jitter experienced on a fully-loaded TDP network. The outcome demonstrates the effectiveness of the presented implementation while confirming TDP points of strength.

I. Introduction

Circuit-switched networks are well suited for transporting constant-bit-rate (CBR) real-time traffic with deterministic delay guarantees owing to their strictly synchronous operation, but may be highly inefficient for carrying bursty data traffic.

Packet-switched networks were designed for achieving high resource utilization even with bursty traffic by exploiting the statistical-multiplexing capability resulting from their asynchronous operation. Unfortunately, they are pretty inefficient in transporting real-time traffic, i.e., the network cannot be fully loaded with such traffic. In fact, the currently widespread approach for supporting delay and jitter-sensitive real-time traffic relies on (*i*) differentiating traffic into at least a couple of classes, (*ii*) giving higher priority to delay-sensitive traffic, and (*iii*) keeping the amount of high-priority traffic small with respect to the total network capacity [1]. Routers handle high-priority packets expeditely so that they are not dropped and experience a short queueing delay, which results in low end-to-end delay and jitter. Nevertheless, this approach, besides leading to network resources' under-utilization as a consequence of (*iii*), cannot guarantee deterministic service to individual real-time traffic flows (such as a videoconference or a phone call).

Several solutions, including asynchronous transfer mode (ATM) and Integrated Services [2], have been proposed for providing flow-level quality-of-service (QoS) guarantees over packet-switched networks, but they involve complicated packet-scheduling algorithms (such as Weighted Fair Queuing) and resource reservation procedures that, when implemented on a per-flow basis, impose a significant performance and cost toll on the entire network.

The Time-Driven Priority (TDP) packet-scheduling technique combines the efficiency of statistical multiplexing typical of asynchronous packet-switched networks with the predictability of synchronous circuit-switched networks. The result is a packet-scheduling technology that can guarantee deterministic end-to-end delay performance even when heavily loaded with real-time traffic, while still being able to let bursty traffic dynamically share available network resources.

Nevertheless, TDP has only been investigated by means of theoretical and simulative tools: the lack of a practical implementation is currently a critical breakpoint in the development of this technology. The primary objective of our work has been then to implement TDP on a PC-based router in order to demonstrate feasibility and provide a prototype for measurements and further studies based on experiments.

The rest of the paper is organized as follows. Section II introduces TDP. Section III provides a description of the TDP implementation on a Personal Computer (PC) running FreeBSD. Section IV describes the experimental testbed and compares the performance of TDP and First-In First-Out (FIFO) packet scheduling in terms of jitter when the network is fully loaded. Finally, Section V concludes the paper.

II. Time-Driven Priority: an Overview

Implementing TDP for real-time packet scheduling requires routers to be synchronized according to a *common time reference* (CTR). The reminder of this section briefly describes the CTR and how it is leveraged by TDP, and envisages an incremental deployment scenario, which would allow for the coexistence of TDP and legacy network technologies. An extensive and detailed description of TDP is outside the scope of this paper and is available in the literature [3], [4].

A. Common Time Reference Structure

The global common clock to which all packet switches are synchronized has a basic time period called *time frame* (TF). The TF duration may be derived, for example, as a fraction of the coordinated-universal-time (UTC) second received from

a time-distribution system such as the global positioning system (GPS), or other equivalent systems, such as the global navigation system (GLONASS), the two-way satellite time and frequency transfer (TWSTFT), and, in the future, Galileo. Time frames are grouped into *time cycles* (TCs) and TCs are further organized into *super cycles*, each of which typically lasts one UTC second. TFs are partially or totally reserved to each flow during a resource reservation procedure. The TC provides the basis for a periodic repetition of the reservation, while the super cycle offers a basis for reservations with a period longer than a TC. This results in a periodic schedule for packets to be switched and forwarded, which is repeated every TC or every super cycle.

For example, in Fig. 1, the 250-μs time frame T_f is obtained by dividing the UTC second by 4000; sequences of 100 time frames are grouped into one time cycle, and runs of 40 time cycles are comprised in one super cycle.

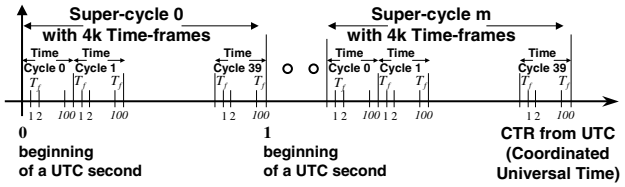

Fig. 1. Common time reference structure

B. Periodic Forwarding

The basic TDP operation is regulated by two simple rules: (*i*) all packets that must be sent in TF t by a node must be in its output ports' buffers by the end of TF $t-1$, and (*ii*) a packet p transmitted in TF t by a node n must be transmitted in TF $t+d_p$ by node $n+1$, where d_p is an integer constant called *forwarding delay*, and TF t and TF $t+d_p$ are also referred to as the *forwarding TF* of packet p at node n and node $n+1$, respectively. The value of the forwarding delay is determined at resource-reservation time and must be large enough to satisfy (*i*).

As exemplified in Fig. 2, which depicts the journey of a packet from node A to node D along three TDP switches, the forwarding delay may have different values for different nodes, due to different propagation delays on different links (e.g., T_{ab}, T_{bc}, and T_{cd}), and different packet-processing times in heterogeneous nodes (e.g., T_{bb} and T_{cc}). Moreover, two variants of the basic TDP operation are possible. When node n deploys *immediate forwarding*, the forwarding delay has the same value for all the packets transmitted by node n. When implementing *non-immediate forwarding*, node n may use different forwarding delays for packets belonging to different flows.

The periodic scheduling within each node results in a *periodic packet forwarding* across the network, which is also referred to as *pipelined forwarding* for the ordered, step-by-step fashion, with which packets travel toward their destination. TDP guarantees that reserved real-time traffic experiences:

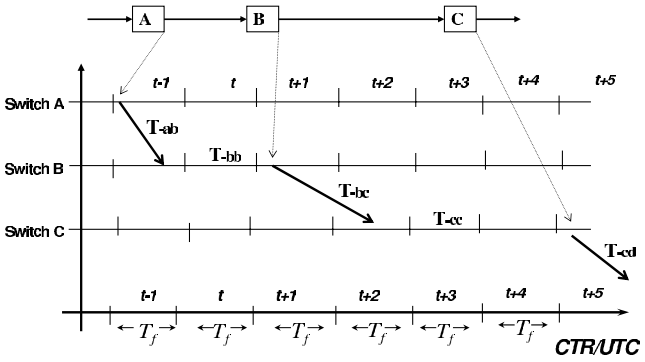

Fig. 2. Basic TDP operation

(*i*) bounded end-to-end delay [4], (*ii*) delay jitter lower than two TFs[1] [4], and (*iii*) no congestion and resulting losses.

Best-effort traffic can be transmitted during any unused portion of a TF, i.e., links can be fully utilized even if flows with reserved resources generate fewer packets than expected.

C. Incremental Deployment Scenario

Today, the Internet is mostly based on asynchronous packet switches. Thus, especially in an initial deployment phase, TDP routers would have to coexist and interoperate with current asynchronous packet switches as depicted in Fig. 3. Synchronous *edge routers* would control the access to the synchronous TDP backbone by policing and shaping the incoming traffic flows. Although TDP provides a maximum delay bound when deployed end-to-end, it could also be beneficial when confined to subnetworks: edge routers at the ingress of each TDP subnetwork would eliminate the jitter experienced by packets in the asynchronous network; then packets would benefit from the controlled-delay service provided by the synchronous subnetwork.

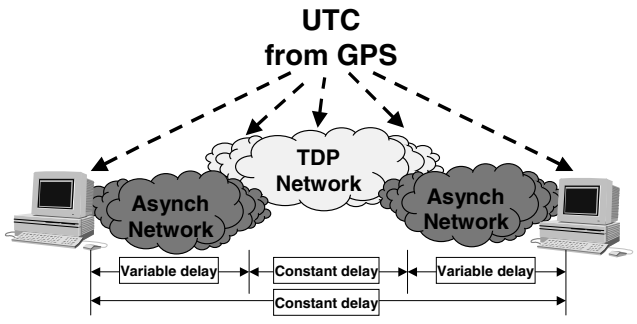

Fig. 3. Interoperation between TDP networks and asynchronous networks

D. Data Plane Operation

In any generic router data plane, packets are moved from input ports to output ports going through three modules that perform *input processing*, *forwarding*, and *output processing*.

[1]In the context of this paper the jitter is the difference between the maximum and the minimum delay experienced from sender to receiver by packets belonging to the same flow.

This section discusses the operations to be performed by a TDP router in each module.

The input module is responsible for determining (1) whether a packet should be scheduled according TDP and, if that is the case, (2) its forwarding TF. Ultimately, the forwarding TF at node n is determined based on (*a*) the forwarding TF at node $n-1$, and (*b*) the forwarding delay fixed at resource-reservation time. In the incremental deployment scenario, edge routers need to shape asynchronous traffic entering the TDP network. Consequently, their input module also comprises mechanisms to classify incoming packets, identify the data flow they belong to, and select a forwarding TF, which determines the output buffer where they will be stored by the output module.

The forwarding module processes packets according to the technology on which TDP is deployed, which does not require any modification for supporting TDP. So, routing may indifferently be based on conventional Internet-Protocol (IP) destination-address-based routing, or multiprotocol label switching (MPLS), or any other technology of choice. This stems from the fact that TDP is just a packet-scheduling technique that can be deployed in the context of any packet-switching technology, without any requirements or impact on the forwarding-module operation.

The output module deploys a per-TF, per-output queuing system, where packets to be forwarded during the same TF through the same interface are buffered in the same queue. The queue in which each packet is stored is determined by both the input module, which decides the forwarding TF, and the forwarding module, which selects the output interface. The output module is also responsible for the timely transmission of all the packets stored in the queues to be flushed in the current TF.

Summarizing, implementing TDP requires only very-limited and straightforward modifications to the input and output modules of currently-deployed routers.

E. Control Plane Operation

A signaling protocol must be chosen for performing resource reservation and selecting the TF in which packets belonging to a given flow should be forwarded by each router. Existing standard protocols and formats should be used whenever possible. Many solutions have been proposed for distributed scheduling in TDP networks [4], [5] and the generalized MPLS (G-MPLS) control plane provides signaling protocols suitable for their implementation. In the most popular provisioning models, such as ATM User-Network Interface and Integrated Services [2], applications signal their QoS requirements to the network for each flow (usually called *micro-flow*); queuing algorithms used in these cases have to maintain status information for each micro-flow, which is not scalable. TDP immediate forwarding does not require per-micro-flow status in intermediate nodes, thus having similar provisioning scalability as the DiffServ model [1], where micro-flows are aggregated in the network to improve scalability.

III. Implementation

One of the most common objections to TDP is that its implementation is rather costly and critical because its working principles are fundamentally different from those of existing (and well-known) asynchronous packet-switched networks. One of the objectives of the prototype is to demonstrate that TDP has very-low computational complexity, its implementation requires a negligible effort when compared to the implementation of a whole router and, moreover, TDP can be easily integrated within the operating system of an existing software router.

Our reference implementation is based on PC hardware equipped with Ethernet NICs. We decided to use the FreeBSD open-source operating system because it is a reference for networking-related projects for historical reasons (the TCP/IP network stack was first developed on the BSD platform), it is well documented in [6], and comes with high-quality traffic-management tools, such as the alternate queueing (ALTQ) framework [7], [8]. Because ALTQ includes many packet-scheduling policies such as FIFO, Weighted Fair Queuing (WFQ), Class Based Queuing (CBQ), Hierarchical Fair Service Curve (HFSC), TDP can be simply implemented as just another scheduling discipline.

The current development version supports only immediate forwarding and does not include signaling functions, so that TFs and TCs are allocated statically by manual configuration. Nevertheless, the prototype does not loose in generality and we paid the utmost attention to accurately avoid any design choice that could prevent us from adding signaling functions and implementing non-immediate forwarding in the future.

A. Common Time Reference

Several CTR solutions can be envisioned and have been proposed for TDP in the literature. The most appealing is probably GPS because it is well known, it has worldwide coverage, it is widely deployed, and economies of scale drive costs down. Therefore, in our design, each router includes a Symmetricom GPS-receiver Peripheral Computer Interconnect (PCI) card that can generate interrupts at a programmable rate ranging between less than 1 Hz and 250 kHz. Such interrupts can be used to make sure that TFs begin exactly at the same time at every router[2]. Two global variables, updated whenever GPS-interrupts occur, are added to the FreeBSD kernel to track the current TF and TC number.

B. Router Input Processing

The input module needs to determine the forwarding TF of each packet at the previous router in order to compute its forwarding TF at the current node. This could be achieved in various ways, among which (*i*) attaching some sort of time stamp to each packet, (*ii*) including a TF delimiter within the

[2]A certain error in the relative beginning of TFs is tolerated and is way beyond the accuracy provided by the GPS receivers deployed in the prototype. A more detailed discussion of synchronization issues and tolerance on synchronization errors is outside the scope of this paper and will be addressed by subsequent publications.

data stream, (*iii*) precisely measuring both the propagation delay and the arrival time of each packet. Among these solutions, (*i*) is deployed in the presented implementation, using the differentiated-service (DS) field to carry a compressed time stamp (encoded on two bits). Each TDP packet transmitted on a link with a TDP router at the other end is tagged with one of the following non reserved DS codepoints [1]: 0x0c, 0x1c, 0x2c, and 0x3c. Bits 0x0c are set in all TDP packets to distinguish them from the others (i.e., those receiving a best-effort or differentiated service), bit 0x10 is set to 1 (0) in packets transmitted during odd (even) TFs, and bit 0x20 toggles its state every TC. This results in an alternating-bit protocol for TF and TC identification, whose simplicity and robustness properties are well known[3].

Two variables maintained by TDP routers for each input interface contain the number of the TF and TC during which the last received packet was transmitted by the upstream node. Each of these variables is updated every time the corresponding bit in the DS codepoint of a packet has a different value with respect to the one of the previous packet. When a node has no packets (including non-TDP packets) to transmit during a TF on a given link, it sends sequences of *padding* IP packets[4] with TF and TC marking for keeping the router at the other end "synchronized"[5]. This solution is elegant and effective since (*i*) it does not require any new standard or protocol, (*ii*) introduces very-limited computational overhead and no transmission overhead, and (*iii*) is resilient to packet losses.

Dummynet [9] is a firewall extension for selecting packets using programmable rules and pass them through objects called *pipes*, which are used to emulate bandwidth and resource limitations, propagation delays and packet losses. In the testbed, edge routers use Dummynet to classify incoming asynchronous data flows and store packets until one TF before they are fetched by the output module for transmission.

C. Router Output Processing

Let d be the maximum forwarding delay (in TFs) at a node, T_f the TF duration (in seconds), C the output link capacity (in bit/s). Given that the prototype has a minimum packet-processing and switching time and the links deployed in the testbed have negligible propagation delay, each output interface should be equipped with $n_{\mathrm{buf}} = d + 1$ queues, each having a capacity of $T_f \cdot C$ bits. Adding one extra queue simplifies the output-scheduler implementation as it ensures that packets are never written into the queue from which the scheduler is retrieving packets for transmission. Hence, TDP buffering requirements are moderate: for example, less than 16 KB per output interface in the experiments presented in Section IV ($d = 4$, $C = 100$ Mb/s, $T_f = 250\,\mu$s).

Packets arriving at the output interface are enqueued in a buffer determined as a function of the forwarding TF selected by the input module (TF_{out}), the current TF (TF_{curr}), the number of TFs per TC (n_{TF}), and the buffer being currently flushed (buf_{curr}):

$$((TF_{\mathrm{out}} + n_{\mathrm{TF}} - TF_{\mathrm{curr}}) \bmod n_{\mathrm{TF}}) + buf_{\mathrm{curr}}) \bmod n_{\mathrm{buf}}.$$

The interrupts generated by the GPS card at the beginning of each TF are used for updating the local TF counter, for starting packet transmission from the current output buffer, and, in edge routers, for transferring the packets to be sent at the beginning of the next TF from the corresponding Dummynet queue to the correct output buffer.

D. Implementation Complexity and Limitations

TDP can be implemented by adding about 1000 lines of C code to the FreeBSD kernel — networking subsystem, ALTQ, and Dummynet. Considering that the FreeBSD networking subsystem itself (i.e., with no QoS support) comprises more than 14 000 lines of C code, this demonstrates that TDP enables adding to existing routers support for deterministic QoS for a negligible implementation complexity and cost. Moreover, since it uses simple FIFO queues served in a cyclic way, TDP does not need any network driver changes.

The PC architecture leads to several limitations that make our implementation far from ideal. For example, TDP requires that packets to be transmitted in a given TF are sent out through all interfaces at the beginning of such TF. The prototype, instead, due to the PC's mono-processor and mono-bus architecture, ends up serving all interfaces sequentially, delaying the beginning of transmission on the various links. Although such additional delay has a minor impact since it is equivalent to having longer links, its variations affect the number of bits that can be transmitted before the end of a TF. The variation in the beginning of transmission on different interfaces results from the combination of several latencies that are outside the control of the operating system, such as interrupt-servicing and bus-acquisition time. However, such latencies become increasingly negligible as central processing units (CPUs) and PC components evolve.

IV. Experiments

Presenting experimental results obtained with the TDP testbed has the main objective of validating the correctness of the TDP implementation. These results can also be used to highlight the benefits that might be gained by deploying TDP on the Internet. To this purpose, a conventional (asynchronous) IP network is compared with a TDP-enabled IP network in terms of maximum jitter experienced by several traffic flows, under link-saturation conditions. However, since this comparison is not the primary focus of this paper — extensive TDP evaluations are available in the literature [3]–[5], [10] — the experiments are not aimed at providing a comprehensive and

[3]Such mechanism can be seen as the transmission of a time stamp composed of the TC and TF number. However, in order to reduce the amount of information transmitted, the time stamp is compressed by sending only the least significant bit of both the TF and TC number, and by keeping state information (the number of the last seen TF and TC) on the nodes.

[4]In the current prototype implementation, *padding* IP packets are addressed to the router interface at the other end of the link, and contain an User-Datagram-Protocol (UDP) message to the standard discard port (9). Other implementation possibilities can be envisioned and devising the best solution is left for future work.

[5]Notice that this does not introduce bandwidth waste since the transmission link would anyway be idle.

exhaustive comparison between the performance of the two technologies (i.e., articulated traffic scenarios are not considered). For example, all experiments are run with constant rate flows of fixed-length packets in order to simplify the system performance evaluation and the manual bandwidth reservation in the complex traffic scenario introduced in Section IV-B.

A. Testbed Setup

The results presented in the reminder of this section are obtained on a testbed consisting of four routers, named R_1, R_2, R_3, and R_4 connected in the full-mesh topology shown in Fig. 4. The traffic is generated using an Agilent N2X RouterTester [11] equipped with 16 Fast Ethernet ports.

Each router is implemented with a PC based on the SiS 645 DX chipset, which supports a 32-bit 66-MHz peripheral computer interconnect (PCI) bus version 2.2 as well as a 533-MHz front-side bus (FSB), and is equipped with a 2.4-GHz Pentium IV CPU, and 256 MByte of 266-MHz double-data-rate (DDR) synchronous dynamic random-access memory (SDRAM). Router R_1 is equipped with five network interface cards (NICs): two for exchanging traffic with the RouterTester and three for connecting the other nodes. Routers R_2, R_3, and R_4 feature four NICs each: one connecting to the RouterTester and three connecting to the remaining routers. All NICs are Intel PRO/1000 MT Gigabit Ethernet server adapters operating at 100 Mb/s.

In the TDP configuration, each PC acts as an edge router since it receives asynchronous traffic from the RouterTester, while handling synchronous TDP traffic received on the other interfaces from TDP routers. The CTR is obtained in each router from a Symmetricom GPS card configured to generate an interrupt at the beginning of each 250-μs TF aligned to UTC (corresponding to a 4000-Hz IRQ-generation rate), and each TC contains 60 TFs. The system clock IRQ frequency is set to 100 Hz, this way, less than 5000 periodic IRQs are received every second, keeping the IRQ-handling overhead incurred by the CPU moderate.

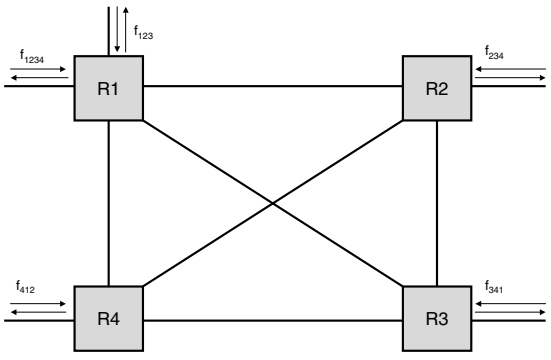

Fig. 4. Experimental testbed setup

B. Traffic Scenarios

In order to have traffic traversing long multi-hop paths and achieve high network utilization, each packet injected in the network loops several times along a circular route before being routed outside the network to the RouterTester. Therefore, all routers run a FreeBSD 4.8 kernel modified to make IP routing decisions based on both the destination address and the time-to-live (TTL) fields.

More in detail, five UDP CBR traffic flows are injected in the testbed network as shown in Fig. 4. The following description contains references to flow-related variables; as with the flow names in Fig. 4, the subscripts indicate the list of routers the corresponding flow traverses. For example, flow f_{1234} is injected on router R_1, loops n_{1234} times along the 4-hop path $R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4 \rightarrow R_1$, and then is routed back to the RouterTester through the same interface on R_1 it came from. Each experiment lasts 10 minutes for guaranteeing statistically-significant measurements.

Since the maximum packet rate on 100-Mb/s Ethernet interfaces ranges between 8127 packets per second (pps) and 148 809 pps as the IP packet size varies from 1500 bytes to 46 bytes, two traffic scenarios with different packet sizes are deployed to assess the impact of the packet rate on the router forwarding performance. In the first scenario, all flows consist of 1002-byte IP packets (resulting in 1020-byte Ethernet frames), whereas, in the second one, all flows comprise 482-byte IP packets (resulting in 500-byte Ethernet frames).

In the first traffic scenario, each flow contributes 6.4 Mb/s and loops 5 times through the network. This produces an overall load of 96 Mb/s on each link traversed by flow f_{1234}, and 32 Mb/s on link $R_1 \rightarrow R_3$ and link $R_2 \rightarrow R_4$. Therefore, each router forwards about 140 Mb/s corresponding to about 17 000 pps.

The second traffic scenario is a little bit more articulated: each of the five flows injects 3.2 Mb/s in the network, but $n_{1234} = 5$, $n_{123} = n_{341} = 10$, and $n_{234} = n_{412} = 15$. Again, each of the links visited by flow f_{1234} transports 96 Mb/s, whereas link $R_1 \rightarrow R_3$ and link $R_2 \rightarrow R_4$ carry 32 and 48 Mb/s, respectively. The traffic forwarded by routers R_1 and R_3 is approximatively 150 Mb/s, corresponding to about 35 000 pps, whereas the traffic forwarded by R_2 and R_4 is approximatively 130 Mb/s, corresponding to about 32 000 pps.

When deploying conventional asynchronous packet switching, all flows are treated equally and a FIFO scheduling policy is applied to each output queue. Given the homogeneity and constant packet rate of the various flows in the experiments run, performance indexes, such as delay and jitter, are not expected to be significantly different from any other work-conserving scheduling algorithm.

On the other hand, under TDP scheduling, all flows are classified as TDP traffic and processed with the immediate forwarding policy. The per-hop forwarding delay is 3 TFs in the first scenario and 4 TFs in the second scenario. The extra time needed in the second case is probably due to the larger overhead incurred by the CPU for processing almost twice as many packets per second as in the first case.

C. Experimental Results

The maximum jitter is measured for each flow. This is computed as the difference between the maximum and the minimum end-to-end delay observed during the whole experiment on packets belonging to the same flow. The results obtained are shown in Table I, for 1020-byte IP packets, and in Table II, for 482-byte IP packets.

TABLE I
END-TO-END DELAY JITTER FOR 1002-BYTE IP PACKETS

Flow name	FIFO [ms]	TDP [ms]
f_{1234}	3.10	0.18
f_{123}	1.97	0.32
f_{234}	2.27	0.34
f_{341}	2.27	0.36
f_{412}	2.31	0.44

TABLE II
END-TO-END DELAY JITTER FOR 482-BYTE IP PACKETS

Flow name	FIFO [ms]	TDP [ms]
f_{1234}	3.58	0.15
f_{123}	4.54	0.46
f_{234}	4.05	0.40
f_{341}	5.10	0.45
f_{412}	3.98	0.42

In both scenarios TDP guarantees a deterministically bounded jitter, that is an order of magnitude smaller (a few hundreds microseconds) than that obtained with asynchronous operation (a few milliseconds).

Furthermore, it is worth noting that no flow experienced any packet loss in any scenario.

V. CONCLUSION

In this paper we provided a description of the first prototypal implementation of a Time-Driven Priority (TDP) router. TDP is a scheduling algorithm that uses a global common time reference (CTR) for shaping packet forwarding inside the network: the implementation of the TDP router was quite simple and it was based on a personal computer with an open source operating system (FreeBSD) equipped with NICs and a GPS card (as CTR source).

The tests performed confirmed the validity of the implementation as it did not show any packet losses or jitter beyond the TDP theoretical bound [4] at full network load. This demonstrates that the time requirements of TDP do not hamper its implementation on an existing routing platforms.

The main outcomes of this work are then a hands-on confirmation of TDP points of strength and, more relevantly, the feasibility of a router supporting TDP over a non real-time architecture such as a PC.

ACKNOWLEDGMENTS

The presentation of this work was supported by the European Union under the E-Next Project FP6-506869. We would like to thank Symmetricom for donating the four GPS cards deployed in the experiments.

REFERENCES

[1] S. Blake *et al.*, *An Architecture for Differentiated Services*, IETF Std. RFC 2475, Dec. 1998.
[2] R. Braden, D. Clark, and S. Shenker, *Integrated Service in the Internet Architecture: an Overview*, IETF Std. RFC 1663, July 1994.
[3] C.-S. Li, Y. Ofek, A. Segall, and K. Sohraby, "Pseudo-isochronous cell switching in ATM networks," *Computer Networks and ISDN Systems*, vol. 30, 1998.
[4] C.-S. Li, Y. Ofek, and M. Yung, "Time-driven priority flow control for real-time heterogeneous internetworking," in *Proc. IEEE (INFOCOM'96)*, vol. 1, Mar. 24–28, 1996, pp. 189–197.
[5] M. Baldi and Y. Ofek, "Fractional lambda switching - principles of operation and performance issues," *SIMULATION: Transactions of The Society for Modeling and Simulation International*, vol. 80, no. 7, July 2004.
[6] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.
[7] "ALTQ." [Online]. Available: http://www.csl.sony.co.jp/~kjc/software.html
[8] K. Cho, "A framework for alternate queueing: Towards traffic management by PC-UNIX based routers," in *Proc. USENIX (Annual Technical Conference'98)*, New Orleans, LA, June 1998.
[9] "Dummynet." [Online]. Available: http://info.iet.unipi.it/~luigi/ip_dummynet
[10] M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," *IEEE/ACM Trans. Networking*, vol. 8, no. 4, pp. 479–492, Aug. 2000.
[11] Agilent, "N2X RouterTester 900." [Online]. Available: http://advanced.comms.agilent.com/n2x