Fluid-Flow Analysis of the Packet Timed Token Service Discipline

L. Lenzini, E. Mingozzi, G. Stea

Dipartimento di Ingegneria della Informazione - University of Pisa
Via Diotisalvi 2, 56126 Pisa - Italy
{l.lenzini, e.mingozzi, g.stea}@iet.unipi.it

Technical Report, December 2001

Abstract – Integrated services networks face the challenge of managing several traffic classes at the same time. Service disciplines devised for integrated services networks therefore need to be flexible, i.e. able to provide different types of service, in order to accommodate different traffic classes efficiently. In this paper we focus on the integration of rate-guaranteed and best-effort traffic, and we argue that service disciplines based on the Generalized Processor Sharing paradigm, which schedule flows according to their weights, lack the flexibility needed to efficiently manage both classes at the same time. We propose that a different service paradigm, the Dual-Class paradigm, which considers the two traffic classes at the same time, be used as a reference to devise flexible and efficient service disciplines for integrated services networks. We then present an innovative Packet Timed Token Service Discipline, which approximates a Dual-Class paradigm. This is shown by means of a fluid-flow analysis on PTTSD.

Index terms – Packet scheduling, Quality of Service, Integrated Services, Timed Token Protocol.

I. INTRODUCTION

The evolution of packet-based network technologies over the last decade, as well as the wide spreading of an instance of these technologies, i.e. the Internet, as a global and commercial communication infrastructure, are challenging the traditional network service models, such as the original TCP/IP's best effort model. As a result, new, alternative service models and traffic management schemes are needed in order to improve Quality of Service (QoS) provision. ATM-based network architectures naturally embody a QoS concept. This is done by specifying (six) standard service categories in terms of a set of parameters characterizing both the traffic presented to the network, and the QoS required from the network [11]. As regards the Internet, two main QoS architectures are considered by the IETF: IntServ, which provides end-to-end QoS on a per-flow basis, and DiffServ, which supports QoS for traffic aggregates [14], [15].

A great challenge for QoS-enabling architectures results from the integration of services. When traffic flows pertaining to multiple traffic classes, each bearing different QoS requirements, coexist in a network, resource management needs to be flexible enough to efficiently provide each flow with a different type of service according to its class, at a feasible computational complexity.

A key component of QoS-enabling architectures are the packet service disciplines (or *scheduling algorithms*) implemented at each switch, selecting which next packet to transmit on an output link, and when, on the basis of several expected performance results. This research area has been extensively investigated in recent years, as testified by the abundance of available literature [13]. It has been observed that a strict priority service discipline is not suitable for an integrated services network, since – although it provides different services for the different priority classes – it offers no means of controlling the service of each priority class [19]. Over the last decade, the ideal fluid-flow service discipline known as Generalized Processor Sharing [9], and its many packet-based derivatives, have received much attention. These service disciplines allocate bandwidth to flows in proportion to their *weights*. A GPS paradigm is well suited for servicing best-effort traffic, since it distributes bandwidth fairly among flows. On the other hand, when weights are selected proportionally to the rate requirements and admission control is enforced, GPS has also been proved to be suitable for servicing rate-guaranteed traffic [9].

Rate-guaranteed traffic and best-effort traffic have very different QoS requirements: rate-guaranteed traffic requires a minimum rate, regardless of the network conditions, whilst best-effort traffic needs no such guarantee. From an economic perspective, rate-guaranteed traffic flows are supposed to be billed according to their minimum guaranteed rate, whilst best-effort traffic can be charged in proportion to the amount of service it receives. Therefore, servicing as much best-effort traffic as possible, while still maintaining the negotiated guarantees for rate-guaranteed traffic could also maximize the profits of service providers.

GPS-based service disciplines lack the flexibility to efficiently account for *both* rate-guaranteed and best-effort traffic at the same time. For instance, when flows

of both traffic classes are scheduled, they are all guaranteed a minimum rate; this is clearly not necessary for best-effort flows; on the other hand rate-guaranteed flows can be serviced at more than the minimum guaranteed rate when the switch is lightly loaded, whilst sharing the whole excess bandwidth *only* among the best-effort traffic would be desirable. These problems are due to the fact that the GPS paradigm only considers *one* type of flows.

In order to achieve a better efficiency in the link capacity utilization, we suggest that a service discipline should approximate an alternative service paradigm, the Dual-Class paradigm, which explicitly considers the two traffic classes at the same time. In the DC paradigm, rate-guaranteed flows are entitled a *fixed* rate, equal to the requested rate, regardless of the network conditions, and best-effort flows share all the *residual bandwidth* (i.e. the bandwidth which has not been reserved for rate-guaranteed flows, plus the bandwidth instantly unused by idle rate-guaranteed flows) according to a GPS paradigm. The DC paradigm maximizes the service provided for best-effort traffic while still meeting the guarantees for rate-guaranteed traffic. Thus, a service discipline based on the DC paradigm seems particularly suitable for an integrated services network.

We then introduce the *Packet Timed Token Service Discipline*, an innovative packet scheduling discipline which approximates the Dual-Class paradigm at a feasible computational complexity. PTTSD manages two distinct types of traffic flows, *synchronous* (i.e. rate-guaranteed) and *asynchronous* (i.e. best-effort), enforcing minimum rate guarantees on the first and servicing the second in a proportional way. More specifically, when synchronous flows are active, they are entitled to use a given amount of bandwidth, while asynchronous flows share most of the residual capacity proportionally. In order to determine the amount of traffic to serve from each synchronous and asynchronous flow, PTTSD applies at the output link of a switch rules based on those used to control medium access by the Timed Token Protocol [1], currently implemented in many ring-based data networks (e.g. FDDI). In this paper, we discuss how PTTSD approximates the Dual-Class paradigm; this is done by means of a fluid-flow analysis of PTTSD.

The rest of the paper is organized as follows. Section II introduces the Dual-Class paradigm and discusses possible ways to approximate it, while Section III describes the PTTSD. In Section IV we derive the fluid-flow model of PTTSD, whilst in Section V we show how PTTSD approximates the Dual-Class Paradigm. Finally, Section VI draws some conclusions.

II. DUAL-CLASS PARADIGM

In this section we formally define the Dual-Class paradigm, and we then discuss possible ways to obtain a service discipline that approximates the DC paradigm.

Let us focus on the output link of a switch, whose capacity is C bits/s. We assume that input traffic is organized into flows, which are grouped into two sets: the first set includes R rate-guaranteed flows, each one requiring a minimum rate $r_1, r_2, ..., r_R$ such that $\sum_{i=1}^{R} r_i \le C$; the other set includes B best-effort flows, each one competing for the available bandwidth with a weight $w_1, w_2, ..., w_B$, according to a GPS paradigm. Figure 1 shows the two flow sets.

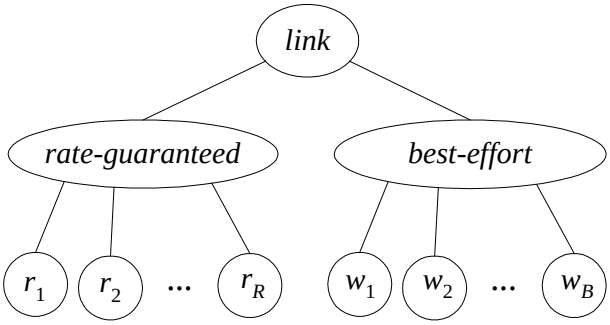

Figure 1 – Flow sets in the DC paradigm

Let us denote with $B_R(t)$ and $B_B(t)$ the set of backlogged rate-guaranteed and best-effort flows respectively at time t, and let us denote with $R_i(t)$ the instantaneous service rate of a backlogged flow at time t.

Definition 1:

We define the (non-work conserving) Dual-Class paradigm as the one for which, at any time instant t:

$$R_i(t) = \begin{cases} r_i & i \in B_R(t) \\ \dfrac{w_i}{\sum_{j \in B_B(t)} w_j} \left(C - \sum_{j \in B_R(t)} r_j \right) & i \in B_B(t) \end{cases} \qquad (1)$$

It is clear that Definition 1 defines a non-work conserving paradigm; in fact, only a non-work conserving paradigm can upper bound the rate of rate-guaranteed flows when no best-effort flow is backlogged.

We can restrict rate-guaranteed flows to be serviced at a fixed rate only when at least one best-effort flow is

backlogged, and define a work-conserving version of the DC paradigm.

□

Definition 2:

We define the (work-conserving) Dual-Class paradigm as the one for which, at any time instant t:

$$R_i(t) = \begin{cases} \dfrac{r_i}{\sum_{j \in B_R(t)} r_j} \cdot C \quad i \in B_R(t),\ B_B(t) = \varnothing \\ r_i \quad i \in B_R(t),\ B_B(t) \neq \varnothing \\ \dfrac{w_i}{\sum_{j \in B_B(t)} w_j} \left(C - \sum_{j \in B_R(t)} r_j \right) \quad i \in B_B(t) \end{cases} \tag{2}$$

In the work-conserving version of the DC paradigm, to which we will refer hereafter, rate-guaranteed flows fairly share the link capacity on the basis of their required rates when no best-effort flow is backlogged.

□

It is clear that (2) can only hold in an ideal fluid-flow service discipline, in which multiple flows can be serviced at the same time. Therefore it seems natural to ask if either GPS or Hierarchical GPS [17], both of which are fluid-flow service disciplines, exhibit the same behavior. We show that this is not the case by means of a simple example.

Example

Suppose that two rate-guaranteed flows i and j and best-effort flow k are scheduled under GPS, according to any choice of weights which enables flows i and j to be served exactly at the required rate when all flows are backlogged. When flow i is not backlogged, part of its bandwidth will go to flow j even if k is backlogged, which is in contrast to (2). Suppose now that the same flow set is scheduled under a two-level H-GPS and that weights are selected as to provide exactly the required rate to i and j when all flows are backlogged. From Figure 2, which reports the only two possible cases, it is clear that when flow i is not backlogged, at least part of its bandwidth still goes to flow j even if k is backlogged, which is in contrast to (2).

□

The concern of managing different traffic classes is also a key issue of link-sharing [16]. Far from being an alternative to link-sharing, a DC-based service discipline can instead be integrated into a link-sharing framework to improve efficiency where sharing between different traffic classes is required.

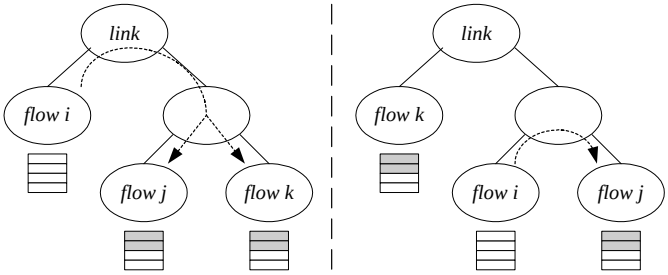

Figure 2 – Excess bandwidth redistribution in H-GPS

It has been also observed that the integration of different traffic classes could be accomplished by employing different schedulers (e.g. one for each traffic class) on a time-sharing or priority basis [13], [16]. This approach is sometimes called *multi-level* scheduling, since it employs a top-level scheduler to decide which bottom-level scheduler is active when. While multi-level scheduling would certainly increase the cost of a switch, to the best of our knowledge no general rules exist concerning what service disciplines to use at each level, and no general method of analysis is known.

Two scheduling discipline which explicitly take into account two traffic classes, namely *latency-critical* (i.e. real-time) traffic and best-effort traffic have been proposed as extensions of Deficit Round-Robin. The first one, called DRR+ ([20]), was shown in [21] to be unfit to schedule latency-critical traffic in multi-hop networks, since it demotes it to best-effort as soon as any burstiness is accumulated. The second one, called DRR++, and presented in [21], also accounts for bursty latency-critical flows. To the best of our knowledge, no formal analysis of DRR++ has been carried out in order to explore what real-time guarantees can be enforced under what conditions.

A unified framework which schedules two sets of tasks on a processor according to (2) was first proposed in [12], though in a different context from that of network scheduling. It has been proposed therein that the DC paradigm be approximated by coupling a dynamic weight management policy with the GPS-based service discipline EEVDF. In this context, flows still receive service on a weight basis, but weights are dynamically varied according to the backlog state of the flows in order to approximate (2). The weight

management policy requires the whole set of weights to be recomputed at a packet timescale. Moreover, since a virtual time function is used to sort packets, weight recomputation should also imply packet resorting. Similar arguments can be used to show that the same problem also arises when other GPS-based service disciplines are employed, both in a flat or in a hierarchical framework. Therefore, DC paradigm approximation through a GPS-based service discipline does not seem to be a computationally feasible solution.

Though devised for the very different context of medium access control in a token ring network, the Timed Token Protocol presents some features that make it a good starting point for developing a packet service discipline which approximates the DC paradigm. Specifically, TTP considers two traffic classes, namely synchronous and asynchronous. On each token visit to a node, synchronous traffic can be transmitted for a bounded time, while asynchronous traffic transmission time adapts to the token pace in order to keep the inter-token time constant[1]. It has been proven in [2]-[6] that synchronous service can be used for transmitting real-time traffic, and several simulative studies [8] show that the TTP offers to each node an identical opportunity to transmit asynchronous traffic. Moreover, asynchronous traffic effectively takes advantage of the bandwidth not used by synchronous traffic. A packet service discipline which services flows according to either the synchronous or asynchronous TTP service can be obtained at the same computational complexity of a simple round-robin scheduler.

III. Packet Timed Token Service Discipline

In this Section we present the Packet Timed Token Service Discipline, and we analyze its properties.

PTTSD is a work-conserving frame-based service discipline, which manages two types of flows[2]: *synchronous flows*, for which a bounded transmission period (*synchronous bandwidth*[3]) is reserved in each frame, and *asynchronous flows*, which transmit their traffic depending on the frame duration.

We will describe and analyze PTTSD under the assumption that it manages a set of N_S synchronous flows, each of which is assigned a synchronous bandwidth H_h, $h = 1...N_S$, and a set of N_A asynchronous flows. We will denote with S and A the two sets. We will assume that each flow enqueues its packets to a separate queue. Packets from synchronous flow i have a maximum transmission time of τ_i seconds; we denote $\tau_{\max} = \max_{i=1..N_S} (\tau_i)$.

In PTTSD, a reference frame duration *TTRT* (*Target Token Revolution Time*) must be selected. This time is used by the asynchronous flows to bound their transmissions.

a. Algorithm Definition

In PTTSD, synchronous and asynchronous flows are considered for transmission in a fixed order on a round-robin basis. Within a single round (or *revolution*), synchronous flows are considered first, followed by the asynchronous flows.

Ideally, a backlogged synchronous flow i should be serviced for a time H_i on each revolution. The presence of variable length packets – which require atomic transmission – clearly makes it impossible to exactly fill the synchronous bandwidth on each revolution. In order to minimize the impact of packet lengths on the synchronous transmission times PTTSD uses a *service lag* variable and a two-cycle mechanism similar to those used by Carry Over Round Robin [10]. The latter is a service discipline for rate-guaranteed ATM connections, which considers fixed length cells. However, our PTTSD analysis also shows that the same mechanism can also handle variable length packets. Each synchronous flow is associated with a variable Δ_i (*service lag*), which measures the difference between the service time that a backlogged synchronous flow required and the service time it actually received. Each Δ_i is managed as follows:

- reset to zero when synchronous flow i is idle;
- incremented by H_i on each revolution if flow i is backlogged;
- decremented by the service time flow i receives when it is being serviced.

Within a single revolution, PTTSD makes two consecutive *sub-cycles* servicing the synchronous flows before servicing the asynchronous ones. In the first sub-cycle, called the *major cycle*, each synchronous flow i is serviced for a time interval less than or equal to Δ_i (i.e. a packet transmission is not allowed to start unless it is completed within that interval). In the subsequent sub-cycle, called the *minor cycle*, only synchronous

[1] The algorithm performed by each node in a TTP ring is reported in more detail in Appendix A.

[2] The mechanisms by which incoming traffic is classified into flows are beyond the scope of this paper.

[3] It must be noted that, according to the TTP terminology, which will be broadly reused for PTTSD, the word bandwidth denotes a time interval.

flows with $\Delta_i > 0$ are eligible for transmission, and only *one* packet per flow may be transmitted. PTTSD stops transmitting synchronous flows packets when all the eligible flows have been served in the minor cycle, or when a time equal to or greater than $\sum_{h \in S} H_h$ has elapsed since the beginning of the revolution, whichever occurs first. Note that synchronous flows can be serviced until at most $\sum_{h \in S} H_h + \tau_{\max}$ units of time have elapsed since the major cycle has started. The following constraint (*protocol constraint*) is therefore required in order to guarantee that all synchronous flows can be properly serviced within a *TTRT* time interval:

$$\sum_{h \in S} H_h + \tau_{\max} \leq TTRT \quad (3)$$

After the minor cycle ends, PTTSD considers the asynchronous flows before starting a new revolution. Each asynchronous flow is associated with a variable L_j (*lateness*), which records the delay accumulated in the previous revolutions. A backlogged asynchronous flow j calculates the available transmission time (*asynchronous bandwidth*) when it is visited, according to the following algorithm: it computes the elapsed time t of the *previous* revolution duration (i.e. the one which started from the last time it was visited), subtracts *TTRT* and adds the result to its lateness. If the latter is non-negative, the flow does not transmit packets (since this would delay the revolution for the synchronous flows); otherwise, the asynchronous bandwidth of the flow is assigned the absolute value of lateness. In this last case, lateness is reset to zero and the flow transmits its packets without exceeding its bandwidth. The following assertion is proved in Appendix B:

Proposition 1

"The algorithm performed by PTTSD to compute the available bandwidth for an asynchronous flow is equivalent to that performed by a TTP node with no synchronous traffic (in the absence of failures)."

As regards the initialization of the state variables related to new synchronous or asynchronous flow activation, the following simple operations are needed:

- for a synchronous flow: set $\Delta_i = 0$; select the synchronous bandwidth H_i according to the flow's requirements ;
- for an asynchronous flow: add the flow at the end of the asynchronous flow set; $L_j = 0$ and assume that the "previous visit" occurred at the start time of the ongoing revolution.

The pseudo-code for PTTSD is shown in Figure 3.

```
Sync_Flow_Init (synchronous flow i)
Δi=0;
Select_synchronous_bandwidth Hi

Async_Flow_Init (asynchronous flow j)
Lj = 0 ;
last_visit_timej = start_of_curr_revolution;

Major_Cycle_Visit (synchronous flow i)
Δi+=  Hi;
q=first_packet_transmission_time;

while ((Δi>=q)  and (q > 0))
{     transmit_packet (q);
      Δi -= q;
      elapsed_time+= q;          }
if (q=0) Δi=0;

Minor_Cycle_Visit (synchronous flow i)
q=first_packet_transmission_time;
if (q > 0)
{     transmit_packet (q);
      Δi -= q;
      elapsed_time += q;         }
if (q=0) Δi=0;

Async_Flow_Visit (asynchronous flow j)
t = current_time;
earlyness = TTRT-Lj - (t-last_visit_timej);
if ( earlyness > 0 )
{     Lj = 0;
      q=first_packet_transmission_time;
      while ((earlyness>=q) and (q > 0))
      {      transmit_packet (q);
             earlyness -= q;     }      }
else Lj = - earlyness;
last_visit_timej = t;

PTTSD revolution ()
  elapsed_time=0;
  for (i=1 to NS) Major_Cycle_Visit (i);
  i = 1;
  while((elapsed_time<sum(Hh)) and (i<=NS))
      {      if (Δi>0) Minor_Cycle_Visit (i);
             i ++;               }
  for (j=1 to NA) Async_Flow_Visit (j);
```

Figure 3 - Pseudo-code for PTTSD

IV. FLUID FLOW ANALYSIS OF PTTSD

In order to show how PTTSD approximates the DC paradigm, we derived the fluid-flow model of PTTSD. Fluid-flow PTTSD has been obtained under the following assumptions:

a) packets are infinitely divisible;

b) The limits $\lim_{TTRT \to 0} (H_i / TTRT)$ exist and are finite $i = 1..N_S$.

Under assumption a), the behavior of PTTSD can be greatly simplified:

- during a major cycle, a synchronous flow is serviced for a time equal to its synchronous bandwidth (provided it has enough backlog). No service lag is accumulated;
- no traffic can serviced during the minor cycle, since every synchronous flow (either backlogged or idle) has a null service lag at the beginning of the minor cycle;
- each asynchronous flow is serviced for a time equal to its asynchronous bandwidth, which is computed at the server visit on the flow;
- the protocol constraint (3) can be reformulated as follows:

$$\sum_{h \in S} H_h \le TTRT \qquad (4)$$

This simplified model of PTTSD (which we will refer to as "TTSD" hereafter, since it does not consider packets) can be described by the following pseudo-code.

```
Sync_Flow_Init (synchronous flow i)
Select_synchronous_bandwidth Hi;

Async_Flow_Init (asynchronous flow j)
Lj = 0 ;
last_visit_timej = start_of_curr_revolution;

Major_Cycle_visit (synchronous flow i)
B = current_backlog(i);
Transmit (min (B,Hi));

Async_Flow_Visit (asynchronous flow j)
t = current_time;
earliness = TTRT- Lj - (t-last_visit_timej);
if ( earliness > 0 )
{     Lj = 0;
      B = current_backlog(j);
      Transmit (min (B,earliness));
}
else Lj = - earliness;
last_visit_timej = t;

PTTSD revolution ()
  for (i=1 to NS) Major_Cycle_Visit (i);
  for (j=1 to NA) Async_Flow_Visit (j);
```

Figure 4- pseudo-code for TTSD

A system model of TTSD, reported in Figure 5 can be built considering the parallel between the token circulation in a ring and the sequence of server visits in TTSD. We have proved in the Appendix that the algorithm performed when computing the bandwidth which the serviced asynchronous flow is entitled to is equal to that performed by the asynchronous subsystem in a TTP node. Fairly obviously, the same can be said for a synchronous flow. Therefore, it is possible to analyze the system model as it was a peculiar logical TTP ring.

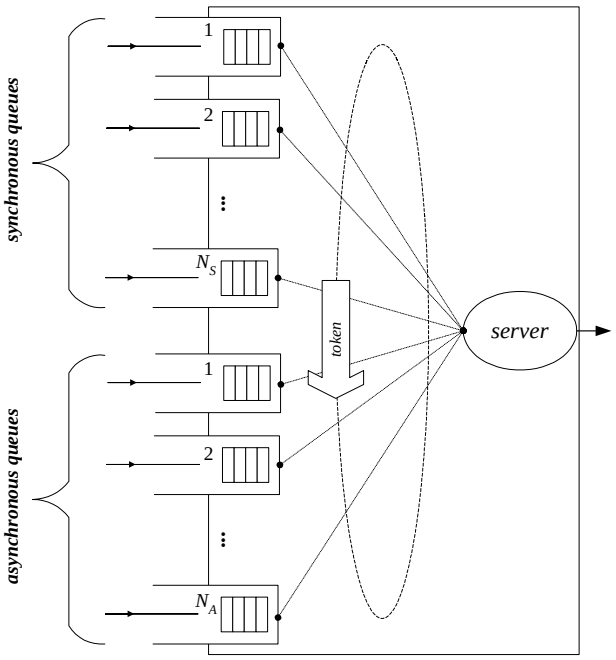

Figure 5 – system model

To construct the fluid-flow model abstraction, we start from the system model, and we let $TTRT \to 0$. Note that this limit operation is possible since we have assumed $\tau = 0$. In order for the protocol constraint to hold, the previous limit operation requires $H_k \to 0$, $\forall k \in S$. This is why we have stated assumption b) at the beginning of this Section. As a consequence of this limit operation, in any finite interval time Δt, the token performs an infinite number of revolutions, each infinitely quickly, and each time the token visits a backlogged queue this receives an infinitesimal service according to TTSD.

Before deriving the fluid-flow model of the system model, we need to investigate the timing properties of the latter.

a. Timing Properties of TTSD

In a generic TTP network, with n nodes that can transmit both synchronous and asynchronous traffic, the following Lemma holds:

Lemma

For any integers $x \geq 1$; $l \geq 1$; $c > 1$; $1 \leq y \leq n$; $1 \leq i \leq n$; and $1 \leq m \leq n$; if $h_{x,y} = H_y$ for all x,y, $(x,y \geq l,m)$, then, under the protocol constraint (4):

$$\sum_{x,y=c,i}^{c+1,i} a_{x,y} \leq TTRT - \sum_{h=1}^{n} H_h - \tau\ ,\ c,i \geq l+1,m \qquad (5)$$

Proof

See [5].

□

Note that the hypothesis of the Lemma requires synchronous transmissions to be constant starting from at least token visit (c-1, i).

For TTSD, the above Lemma can be reformulated taking into account the following differences with the model analyzed in [5]:

- $\tau = 0$;
- left summation accounts for N_A+1 consecutive token visits on asynchronous queues;
- regardless of the way synchronous and asynchronous queues are interleaved, if the token starts and ends on the same asynchronous queue, it also visits all the N_S synchronous queues in the meantime.

Therefore, the above Lemma is reduced to:

Lemma

In our system model, for any integers $x \geq 1$; $l \geq 1$; $c > 1$; $1 \leq y \leq N_A$; $1 \leq i \leq N_A$; and $1 \leq m \leq N_S$; if $h_{z,w} = H_w$ for all z,w, $(z,w \geq l,m)$, then, under the protocol constraint (4),

$$\sum_{x,y=c,i}^{c+1,i} a_{x,y} \leq TTRT - \sum_{h \in S} H_h\ ,\ \ c,i > l+1,m \qquad (6)$$

□

From the above Lemma the following corollary can be derived:

Corollary 1

In our system model, for any integers $x \geq 1$; $l \geq 1$; $c > 1$; $1 \leq y \leq N_A$; $1 \leq i \leq N_A$; and $1 \leq m \leq N_S$; if $h_{z,w} = H_w$ for all z,w, $(z,w \geq l,m)$, then, under the protocol constraint (4)

$$\sum_{x,y=c,i}^{c+N_A+1,i-1} a_{x,y} \leq N_A(TTRT - \sum_{h \in S} H_h)\ ,\ c,i > l+1,m \quad (8)$$

Proof

In a system model with N_A asynchronous queues, (N_A+1) token revolutions include $(N_A+1)\cdot N_A$ token visits on asynchronous queues. Therefore, by applying N_A times inequality (6), the proof follows straightforwardly.

□

Note

If no traffic is transmitted from φ asynchronous queues out of N_A from visit (c,i) to $(c+N_A-\varphi+1,\ i\text{-}1)$ (because the token is late on every visit, or because the queues are not backlogged), the asynchronous bandwidth will be shared among $N_A'=N_A-\varphi$ asynchronous queues, and therefore Corollary 1 can be rewritten replacing N_A with N_A'.

□

The above results will be used to prove Theorem A and Theorem B, which then lead to the following:

- if there are late asynchronous queues and the protocol constraint inequality holds in a strict sense, the token will be early on all the asynchronous queues after a finite number of token revolutions;
- if the token arrives early on all the asynchronous queues, it keeps arriving early as long as the hypothesis of the Lemma holds.

Theorem A

In our system model, for any integers $x \geq 1$; $l \geq 1$; $c > 1$; $1 \leq y \leq N_A$; $1 \leq i \leq N_A$; and $1 \leq m \leq N_S$; if:

- $h_{z,w} = H_w$ for all z,w, $(z,w \geq l,m)$;
- the protocol constraint (4) holds;
- for any given asynchronous queue i the token is early during the c-th visit, where $c,i > l+1,m$

then the token will be early on $(c+1)$-th visit too.

Proof

it follows from (6).

$$t_{c+1,i} - t_{c,i} = \sum_{x,y=c,i}^{c+1,i-1} a_{x,y} + \sum_{h \in S} H_h \leq \sum_{x,y=c,i}^{c+1,i} a_{x,y} + \sum_{h \in S} H_h \leq$$
$$\leq (TTRT - \sum_{h \in S} H_h) + \sum_{h \in S} H_h = TTRT$$

If the token is early on its c-th visit to the asynchronous queue i at time $t_{c,i}$, $t_{c+1,i} - t_{c,i} \leq TTRT$ is a sufficient condition for it being early on its (c+1)-th visit $t_{c+1,i}$.
□

Theorem B

In our system model, for any integers $x \geq 1$; $l \geq 1$; $c > 1$; $1 \leq y \leq N_A$; $1 \leq i \leq N_A$; and $1 \leq m \leq N_S$; if:

- $h_{z,w} = H_w$ for all z,w, $(z,w \geq l,m)$;
- the protocol constraint (4) holds;
- at time t_0 the token is late on at least one asynchronous queue,

then, at worst after the following number of token revolutions:

$$\eta = N_A \cdot \left\lceil \frac{\sum_{h \in S} H_h}{TTRT - \sum_{h \in S} H_h} \right\rceil \quad (9)$$

the token will not be late on any asynchronous queue.

Proof

Let us define $L_{c,i}$, *"lateness"* on the token's c-th visit to asynchronous queue i, the quantity:

$$L_{c,i} = \max \left\{ 0, \left(t_{c,i} - t_{c-1,i}\right) - TTRT + L_{c-1,i} \right\}$$

where $L_{0,i}=0 \ \forall i \in A$.

The timing properties of the TTP ensure that – when the network operates correctly and under the protocol constraint – we have:

$$0 \leq L_{c,i} \leq \sum_{h \in S} H_h$$

While the lower bound for $L_{c,i}$ is obvious from the definition, the rightmost inequality is an immediate consequence of "Johnson and Sevcik's Theorem" [2].

Let us assume that at time $t_{c,I}$ the token is late on at least the asynchronous queue i. This means that *at least* one out of N_A asynchronous queues is not transmitting anything. Therefore, from Corollary 1 it follows that:

$$t_{c+N_A,i} - t_{c,i} = N_A \cdot \sum_{h \in S} H_h + \sum_{x,y=c,i}^{c+N_A,i-1} a_{x,y} \leq$$

$$N_A \cdot \sum_{h \in S} H_h + \left(N_A - 1\right) \cdot \left[TTRT - \sum_{h \in S} H_h \right] =$$

$$N_A \cdot TTRT - \left[TTRT - \sum_{h \in S} H_h \right]$$

The above result implies that:

- the time duration of N_A token revolutions is less than or equal to $N_A \cdot TTRT$;
- after at most N_A token revolutions, a time interval equal to $TTRT - \sum_{h \in S} H_h$ has been recovered, i.e. queue i lateness decreases by $TTRT - \sum_{h \in S} H_h$.

Since the maximum lateness that a queue can experience is upper bounded by $\sum_{h \in S} H_h$, the derivation of (9) is straightforward.
□

It is therefore possible to say that the system model with constant synchronous transmissions will always reach a state in which:

- if $\sum_{h \in S} H_h < TTRT$, then the token is early on each queue visit after at most γ revolutions ;
- if $\sum_{h \in S} H_h = TTRT$, then a null quantity of asynchronous traffic can be transmitted.

This state, which will be named *stationary state*, is always reached in a finite interval time.

A similar result has been proven [22] for an FDDI network in which the effects of accumulated lateness are neglected, i.e. the *Lc* counters are *always* reset upon a token arrival. Our improvements with respect to [22] are:

- the proof that such a state is always reached in a finite number of token revolutions *even if* lateness is taken into account;
- an upper bound to these revolutions, quantified by (9).

b. Asynchronous vector

We will now focus on the evolution of the asynchronous traffic in the stationary state, under the hypothesis that the asynchronous queues are always

backlogged. The rationale behind it will be explained in Section IV.c

Let (c^*,1) be the c^*-th token visit on asynchronous queue 1 in the stationary state. Having assumed that every asynchronous queue is always backlogged, Lemma ensures that:

$$\sum_{x,y=c^*,1}^{c^*+1,1} a_{x,y} = TTRT - \sum_{h\in S} H_h$$

Let us now define the following vector $\mathbf{V} = [V_0, V_1, \ldots, V_{N_A}]$ of N_A+1 components:

$$\begin{cases} V_i = a_{c^*,i} \quad i = 1...N_A \\ V_0 = (TTRT - \sum_{h\in S} H_h) - \sum_{x,y=c^*,1}^{c^*,N_A} a_{x,y} = a_{c^*+1,1} \end{cases}$$

Notice that all the vector's elements are known at time t_{c^*,N_A}, i.e. within a single token revolution. We will call vector **V** the *asynchronous vector of token revolution c**.

Let us now instantiate (6) to (c^*,2):

$$\sum_{x,y=c^*,2}^{c^*+1,2} a_{x,y} = \sum_{x,y=c^*,1}^{c^*+1,1} a_{x,y} + a_{c^*+1,2} - a_{c^*,1} = TTRT - \sum_{h\in S} H_h$$

Similarly,

$$a_{c^*+1,3} = a_{c^*,2} = V_2, \ \ldots \ a_{c^*+1,N_A} = a_{c^*,N_A-1} = V_{N_A-1}$$

$$a_{c^*+2,1} = a_{c^*,M} = V_{N_A}, \ \ a_{c^*+2,2} = a_{c^*+1,1} = V_0$$

Since $\sum_{x,y=c^*,1}^{c^*+1,1} a_{x,y} = TTRT - \sum_{h\in S} H_h$, it must be $a_{c^*+1,2} = a_{c^*,1} = V_1$.

From the above recurrence formulas, it is straightforward to prove:

Corollary 2

When the system model is in the stationary state, if the asynchronous vector related to token rotation c^* is known, then:

$$a_{c,j} = V_{[j-(c-c^*)] \bmod (N_A+1)}, \ \forall c \geq c^*, \ 1 \leq j \leq N_A \quad (10)$$

Proof

The proof is obvious, and is thus omitted.

□

Equality (10) shows that the sequence of transmissions from each asynchronous queue is *periodic*, with a period of (N_A+1) token revolutions[4].

Corollary 3

When the system model is in the stationary state, the service difference between any two backlogged asynchronous queue over any time interval is upper bounded by:

$$\eta_{i,j} = TTRT - \sum_{h\in S} H_h, \quad 1 \leq j \leq N_A \quad (11)$$

Proof

The proof is obvious, and is thus omitted.

□

Note:

In the stationary state, if we assume that the asynchronous queues are always backlogged, the average token revolution time depends on the number of asynchronous queues N_A: it is clear from Corollary 1 that N_A+1 token revolutions have a duration of:

$$(N_A+1) \cdot \sum_{h\in S} H_h + N_A \cdot (TTRT - \sum_{h\in S} H_h) =$$

$$(N_A+1) \cdot TTRT - (TTRT - \sum_{h\in S} H_h)$$

This means that the average token rotation time is:

$$\frac{(N_A+1) \cdot TTRT - (TTRT - \sum_{h\in S} H_h)}{N_A+1} =$$

$$TTRT - \frac{(TTRT - \sum_{h\in S} H_h)}{N_A+1}$$

Clearly, the bigger N_A is, the closer to *TTRT* the average token rotation time is. Note that the only two cases in which in the stationary state the average token rotation time is equal to *TTRT* are $N_A = \infty$ and $\sum_{h\in S} H_h = TTRT$. For finite N_A values and if inequality holds in the protocol constraint, the average token revolution time is *less* than *TTRT* in the stationary state. Since a synchronous queue k is served for a time H_k on each token visit, keeping the token revolution time close to *TTRT* means keeping the rate at which a

[4] Clearly, if equality holds in the protocol constraint, asynchronous transmissions are always null, and therefore periodical.

synchronous queue can be served close to (but higher than) $H_k / TTRT$.

□

c. Fluid-flow rates

In this section, we first detail the procedure for obtaining the *fluid-flow model*, which, as stated before, requires the limit operation $TTRT \to 0$.

After that, we calculate the instantaneous rates for each synchronous and asynchronous flow. To achieve this, we observe the behavior of the fluid-flow model in a time interval $[t, t+\Delta t)$ in which we assume that the state of each queue (*empty* or *backlogged*) will not change, and we let $\Delta t \to 0$. Obviously, there exists a Δt^* (equal to the minimum backlog among all the queues at time t) such that for any $\Delta t \le \Delta t^*$ the fluid-flow model exhibits such a property. Note that the order in which the above limit operations are carried out is fundamental, since it makes no sense to calculate instantaneous rates in a polling model with finite sojourn times.

- on each token visit, backlogged queues always hold the token for the maximum allowed time;
- the asynchronous queues can be partitioned into two disjoint subsets: one backlogged, which includes queues with indexes $\alpha_1, \alpha_2, ..., \alpha_M$, $1 \le \alpha_1 \le \alpha_2 \le ... \le \alpha_M \le N_A$, and the other includes empty queues;
- the synchronous queues can be partitioned into two disjoint subsets: one backlogged, which includes flows with indexes $\sigma_1, \sigma_2, ..., \sigma_K$, $1 \le \sigma_1 \le \sigma_2 \le ... \le \sigma_K \le N_S$, and the other including empty queues.

Let us denote by $B_A(t) = \{\alpha_1, \alpha_2, ..., \alpha_M\}_{(t)}$ and $B_S(t) = \{\sigma_1, \sigma_2, ..., \sigma_K\}_{(t)}$ the subsets of backlogged queues previously defined; clearly, the cardinality and composition of $B_A(t)$ and $B_S(t)$ are time varying. We denote with $M(t)$ the cardinality of $B_A(t)$.

Since our goal is to find timing properties that hold in the time interval $[t, t+\Delta t)$, queues that are never backlogged in $[t, t+\Delta t)$ need not be considered. Thus, in the proof process of Theorem C, summations on asynchronous/synchronous queues will be performed on subsets $B_A(t)$ and $B_S(t)$ respectively. In order to simplify the notation drastically, we will use the subscript $1...M(t)$ instead of $\alpha_1, \alpha_2, ..., \alpha_M$.

Theorem C

If the limit $C_k = \lim_{TTRT \to 0} \left(H_k / TTRT \right)$ exists and is finite $\forall\ k \in S$, then the instantaneous normalized rates at which synchronous and asynchronous queues are served in the fluid-flow model at time t are, respectively:

$$R_k^{(S)}(t) = \frac{[M(t)+1] \cdot C_k}{M(t) + \sum_{h \in B_S(t)} C_h} \quad \forall\ k \in B_S(t) \tag{12}$$

$$R_j^{(A)}(t) = \frac{1 - \sum_{h \in B_S(t)} C_h}{M(t) + \sum_{h \in B_S(t)} C_h} \quad \forall\ j \in B_A(t) \tag{13}$$

Proof

Let us distinguish two cases: $\sum_{h \in B_S(t)} H_h = TTRT$ and $\sum_{h \in B_S(t)} H_h < TTRT$.

Case 1

If $\sum_{h \in B_S(t)} H_h = TTRT$, then $\sum_{h \in B_S(t)} C_h = 1$, i.e. only synchronous queues are served. As a consequence, $R_k^{(S)}(t) = C_k$, and $R_j^{(A)}(t) = 0$, and therefore (12) and (13) are obviously verified.

Case 2

$\sum_{h \in B_S(t)} H_h < TTRT$. Let us refer to Figure 6, where t is the time at which system observation starts for an interval time Δt. We assume that from t onward synchronous transmissions remain constant, and therefore the Lemma hypothesis holds. In the most general case the system will experience a transitory phase which has been proved to be finite by Theorem B. Let t_1 be the time instant at which the system reaches the stationary state, and let us count from 0 onward the token revolutions beginning from t_1.

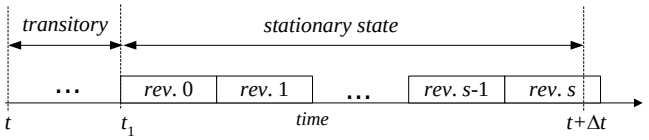

Figure 6 – Transmissions in the time interval $[t, t+\Delta t)$

As proved in Corollary 2, in s token revolutions asynchronous queue j can rely on the following service time:

$$\sum_{c=0}^{s-1} a_{c,j} = \left\lfloor \frac{s}{M(t)+1} \right\rfloor \cdot \left[TTRT - \sum_{h \in B_S(t)} H_h \right] + \sum_{l=0}^{r-1} V_{(j-l) \bmod (M(t)+1)} \quad (14)$$

where:

$$s = q \cdot (M(t)+1) + r,$$

$$q = \left\lfloor \frac{s}{M(t)+1} \right\rfloor,\ q \in \{0, 1, \ldots\},\ 0 \le r \le M(t)$$

From Figure 6 it follows:

$$\Delta t - TTRT \le (t_1 - t) + \sum_{c=0}^{s-1} \left[\sum_{j=1}^{M(t)} a_{c,j} + \sum_{h \in B_S(t)} H_h \right] \le \Delta t$$

After some algebraic manipulations and bearing in mind (14), we obtain:

$$\Delta t - TTRT \le (t_1 - t) + s \cdot \sum_{h \in B_S(t)} H_h + M(t) \cdot \left\lfloor \frac{s}{M(t)+1} \right\rfloor \cdot \left[TTRT - \sum_{h \in B_S(t)} H_h \right] + \sum_{j=0}^{M(t)} \sum_{l=1}^{r-1} V_{(j-l) \bmod (M(t)+1)} \le \Delta t \quad (15)$$

Let us take the limit $TTRT \to 0$ of (15). When $TTRT \to 0$, $s \to \infty$. Since the first and fourth addenda in (15) are O($TTRT$), they both tend to 0. Furthermore, since the second and third addenda are O($s \cdot TTRT$), (15) is reduced to:

$$\lim_{TTRT \to 0} \left\{ s \cdot \frac{M(t) \cdot TTRT + \sum_{h \in B_S(t)} H_h}{M(t)+1} \right\} = \Delta t \quad (16)$$

If $C_k = \lim_{TTRT \to 0} \left(H_k / TTRT \right)$ exists and is finite, (16) leads to:

$$\lim_{TTRT \to 0} \left(s \cdot TTRT \right) = \Delta t \cdot \frac{M(t)+1}{M(t) + \sum_{h \in B_S(t)} C_h} \quad (17)$$

Let $W_k^{(S)}(t, t+\Delta t)$ and $W_j^{(A)}(t, t+\Delta t)$ be the service time received by synchronous queue k and asynchronous queue j respectively in the time interval $[t, t+\Delta t)$. By definition:

$$R_k^{(S)}(t) = \lim_{\Delta t \to 0} \left\{ \frac{\lim_{TTRT \to 0} W_k^{(S)}(t, t+\Delta t)}{\Delta t} \right\} \quad (18)$$

$$R_j^{(A)}(t) = \lim_{\Delta t \to 0} \left\{ \frac{\lim_{TTRT \to 0} W_j^{(A)}(t, t+\Delta t)}{\Delta t} \right\} \quad (19)$$

We have:

$$\lim_{TTRT \to 0} W_k^{(S)}(t, t+\Delta t) = \frac{[M(t)+1] \cdot C_k}{M(t) + \sum_{h \in B_S(t)} C_h} \cdot \Delta t \quad (20)$$

while on the other hand, since :

$$\lim_{TTRT \to 0} W_j^{(A)}(t, t+\Delta t) = \lim_{TTRT \to 0} \left[\sum_{c=0}^{s-1} a_{c,j} \right]$$

from (14) and (17):

$$\lim_{TTRT \to 0} W_j^{(A)}(t, t+\Delta t) = \frac{1 - \sum_{h \in B_S(t)} C_h}{M(t) + \sum_{h \in B_S(t)} C_h} \cdot \Delta t \quad (21)$$

By substituting (20) and (21) into (18) and (19), the thesis follow straightforwardly.
□

d. Worst-Case Timing Properties of TTSD

We now derive the worst-case token return time for a synchronous flow in TTSD.

Generalized Johnson and Sevcik's Theorem for TTSD

Let $k \in S$ be a synchronous queue in our system model; let N_A be the cardinality of the asynchronous queues set A. Then, for any integer $x \ge 1$, $v \ge 1$, under the protocol constraint (4):

$$t_{x+v,k} - t_{x,k} \le v \cdot TTRT + \sum_{h \in S, h \neq k} H_h - \left\lfloor \frac{v}{N_A + 1} \right\rfloor \cdot \left[TTRT - \sum_{h \in S} H_h \right] \quad (22)$$

Proof

The proof process closely follows the one given in [5] for a generic TTP network. A *worst-case scenario* for the token return time at synchronous queue k can be defined as follows:

- no transmission (either synchronous or asynchronous) takes place from $t = t_{x-1,k}$ to $t = t_{x,k}$ inclusive, i.e. during the token revolution that precedes $t = t_{x,k}$.
- synchronous queue k becomes backlogged at the time instant $t = t_{x,k}^{+}$.
- Every synchronous and asynchronous queue is backlogged from time $t = t_{x,k}^{+}$ onwards, and therefore every queue holds the token for the maximum time allowed by the TTP rules.

Such a scenario has been proved to be the worst-case for a TTP network in [6].

Sub-case 1: *v*=1.

Regarding the first token revolution following $t = t_{x,k}$, it is possible to say that:

- if the first queue visited by the token after $t = t_{x,k}$ is an asynchronous one, it will hold the token for a time interval equal to *TTRT* (recall pseudo-code in Figure 4);
- any synchronous queue *h* will hold the token for a time interval H_h;

the above two statements yield:

$$t_{x+v,k} - t_{x,k} = TTRT + \sum_{h \in S, h \neq k} H_h \quad (23)$$

which is only reached in a worst-case scenario, and therefore inequality (22) holds for *v*=1.

Sub-case 2: *v*>1.

The time interval $[t_{x+1,k},\ t_{x+v,k})$ includes $(v-1)$ complete token revolutions. Based on the worst-case scenario definition, the following statements can be assessed:

- the overall synchronous transmission within the time interval $[t_{x+1,k},\ t_{x+v,k})$ is:

$$(v-1) \cdot \sum_{h \in S} H_h \quad (24)$$

- starting from the first token visit on an asynchronous queue following time $t_{x+1,k}$, the hypothesis of the Lemma holds; we can then apply it for any group of consecutive $N_A + 1$ token visits on asynchronous flows, for an overall number of visits equal to $(v-1) \cdot N_A - 1$. Therefore, it is possible to state that the overall asynchronous transmission within the time interval $[t_{x+1,k},\ t_{x+v,k})$ is upper bounded by:

$$\left\lfloor \frac{(v-1) \cdot N_A - 1}{N_A + 1} \right\rfloor \cdot \left(TTRT - \sum_{h \in S} H_h \right) = \left[(v-1) - \left\lfloor \frac{v}{N_A + 1} \right\rfloor \right] \cdot \left(TTRT - \sum_{h \in S} H_h \right) \quad (25)$$

Putting (23), (24) and (25) together, after a few algebraic manipulations, we obtain:

$$t_{x+v,k} - t_{x,k} = v \cdot TTRT + \sum_{h \in S, h \neq k} H_h - \left\lfloor \frac{v}{N_A + 1} \right\rfloor \cdot \left[TTRT - \sum_{h \in S} H_h \right] \quad (26)$$

Since (26) has been calculated in a worst-case scenario, for every possible scenario (22) holds.

□

Note that inequality (22) also holds if $N_A = 0$. However, in this particular case, tighter bounds can easily be derived.

V. PTTSD as an Approximation of the DC paradigm

Let us denote with $R_i(t)$ the instantaneous service rate of a generic flow at time t. At any time instant t, under fluid-flow PTTSD the service rate of a backlogged flow is:

$$R_i(t) = \begin{cases} \dfrac{[M(t)+1]\cdot h_i}{M(t)+\sum_{j\in B_S(t)} h_j}\cdot C & i \in B_S(t) \\ \dfrac{1-\sum_{j\in B_S(t)} h_j}{M(t)+\sum_{j\in B_S(t)} h_j}\cdot C & i \in B_A(t) \end{cases} \quad (27)$$

Synchronous flow rates are not constant, since they depend on the number of backlogged asynchronous and synchronous flows. However, it is straightforward to see that, as the number of backlogged asynchronous flows increases, the synchronous service rates approaches a minimum value $h_i \cdot C$. Therefore, the service offered to backlogged asynchronous flows *upper bounds* the synchronous flow rates. On the other hand, each asynchronous flow always obtains an equal share of the available bandwidth.

When $M(t)$ is high, (27) can be rewritten as:

$$R_i(t) \cong \begin{cases} h_i \cdot C & i \in B_S(t) \\ \dfrac{1}{M(t)}\left(1-\sum_{j\in B_S(t)} h_j\right)\cdot C & i \in B_A(t) \end{cases} \quad (28)$$

Expression (28) is formally equal to that obtained from (2) in the particular case where best-effort flows weights are all equal. Note that, when no asynchronous flow is backlogged (i.e. $M(t)=0$), (27) yields:

$$R_i(t) = \frac{h_i}{\sum_{j\in B_S(t)} h_j}\cdot C \quad i \in B_S(t)$$

Therefore, fluid-flow PTTSD embodies the DC paradigm when no asynchronous flow is backlogged and approximates an unweighted version when asynchronous flows are backlogged.

The approximation improves as the number of backlogged asynchronous flows increases. Figure 7 and Figure 8 show the ratio between the (synchronous and asynchronous) rates in fluid-flow PTTSD (27) and in the ideal case (28) against $M(t)$ and for various synchronous loads $\sum_{j\in B_S(t)} h_j$. Both figures show that a good approximation of the DC paradigm (within 10%) is achieved when as few as ten asynchronous flows are backlogged.

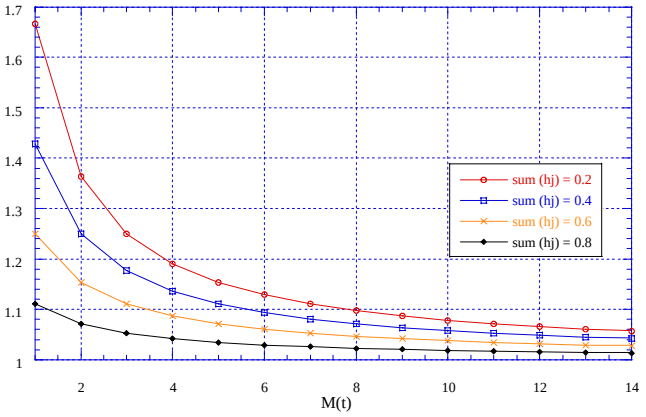

Figure 7 – normalized synchronous rates

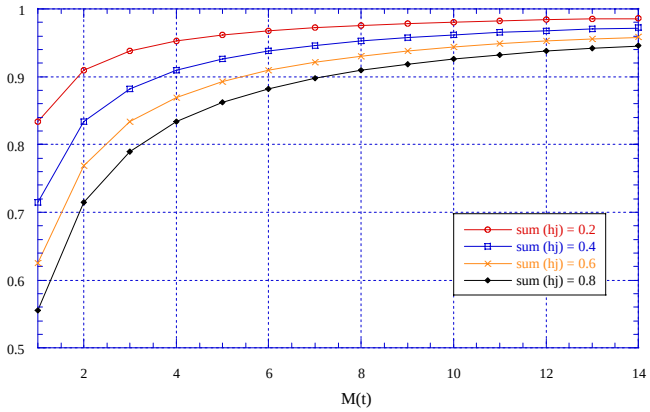

Figure 8 – normalized asynchronous rates

VI. Conclusions

In this paper we have introduced an innovative scheduling paradigm, the Dual Class paradigm, showing that it can be taken as a reference to devise efficient services disciplines for integrated services networks. Specifically, the DC paradigm considers the simultaneous presence of both rate-guaranteed and best-effort traffic, and maximizes the service given to best-effort traffic while meeting the rate guarantees. We have devised a new service discipline, called PTTSD, which manages at the same time both rate-guaranteed (*synchronous*) and best-effort (*asynchronous*) traffic. We have showed that PTTSD approximates the Dual Class paradigm by means of a fluid-flow analysis.

Appendix A: Timed Token Protocol

The Timed Token Protocol has been used as a MAC protocol by the FDDI technology and other network standards. In the following, we describe the TTP as it is implemented in FDDI, and therefore we refer to a

physical ring with *n* nodes attached to it. Transmission rights are granted by the circulation of a special packet, called *token*. The TTP accounts for two distinct classes of traffic on each node: synchronous traffic, which is transmitted for a time duration up to a given maximum on each token visit, and asynchronous traffic, which can only be transmitted if the token arrives earlier than expected.

At ring initialization, a target token rotation time ($TTRT$) is selected. Any node that supports synchronous traffic is then assigned a portion $H_i \geq 0$ of $TTRT$ to transmit it. Each node has two timers: the Token Rotation Timer (TRT), and the Token Holding Timer (THT). The TRT always counts *down* and a node's THT counts *down* only when the node is transmitting asynchronous traffic. If a node's TRT reaches 0 before the token arrives at the node, TRT is reset to $TTRT$ and the arriving token is marked as *late* by incrementing the node's late counter Lc by one. At ring initialization, Lc is set to 0 on each node.

Only the node which possesses the token is able to transmit. When node receives the token, it does the following:

- if $Lc > 0$, sets $Lc := Lc - 1$ and $THT := 0$ Otherwise, $THT := TRT$ and $TRT := TTRT$;
- if it has *synchronous* traffic, it transmits it until either a period H_i has elapsed or all the synchronous traffic has been transmitted, whichever occurs first;
- if it has *asynchronous* traffic, it transmits it until the THT counts down to 0 or until all its asynchronous traffic has been transmitted, whichever occurs first; note that a transmission which is already in progress when THT expires is always completed (*asynchronous overrun*);
- it passes the token to the next node on the ring.

The choice of *synchronous bandwidths* (or *synchronous capacities*) H_h, $h = 1...n$, must obey the following constraint:

$$\sum_{h=1..n} H_h \leq TTRT - \tau$$

called the *protocol constraint*, where (the *latency*) is the portion of token rotation time which is unavailable for traffic transmission, due to protocol and media dependent overheads and to the occurrence of asynchronous overruns [2].

The timing properties of the TTP have been widely explored over the last two decades ([2]-[7]). Two very well known results related to the TTP timing properties are as follows:

- under the protocol constraint, the *maximum* token revolution time is upper bounded by $2 \cdot TTRT$, and the *average* token revolution time is upper bounded by $TTRT$ [2];
- it is possible to identify the worst-case scenario for the token return time [6], and to calculate the worst-case v-th token return time for any $v \geq 1$ (*Generalized Johnson and Sevcik's Theorem*) [5].

APPENDIX B: PROOF OF PROPOSITION 1

Proposition 1

"The algorithm performed by PTTSD to compute the available bandwidth for an asynchronous flow is equivalent to that performed by a TTP node with no synchronous traffic (in the absence of failures)."

Proof

Let us suppose that j is a TTP node, and let us denote with $t_{c,j}$ the time instant at which the token reaches node j for c-th time. We will use the same subscript pair for denoting quantities related to node j sampled at the c-th token visit. We now compute the asynchronous bandwidth in of node j at time $t_{c,j}$, under the hypothesis that it has no synchronous traffic, according to the TTP rules reported in Appendix A. We need to distinguish two cases:

Case 1: the token was late on the previous visit at node j

In this case, at time $t_{c-1,j}$ we have $Lc \leftarrow 0$ (note that $Lc = 1$ just before the token arrival) and TRT is not reset. Let us denote with $L_{c-1,j}$ the difference $TTRT - TRT > 0$ at time $t_{c-1,j}$. Since TRT always counts down, the necessary and sufficient condition for the token to be early at time $t_{c,j}$ is that TRT has not expired yet, i.e. $t_{c,j} - t_{c-1,j} \leq TTRT - L_{c-1,j}$. In that case, at time $t_{c,j}$, we have:

$$TRT = TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right)$$

and $THT = TRT$, and the available asynchronous bandwidth for node j is thus:

$$TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right).$$

If the above condition is not verified, at time $t_{c,j}$ the TRT value is $2TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right)$.

Case 2: the token was early on the previous visit at node j

In this case, at time $t_{c-1,j}$ we had $Lc \leftarrow 0$ (note that $Lc = 0$ just before the token arrival) and $TRT \leftarrow TTRT$. Therefore, the same formulas of case 1 also apply to this case, provided that we set $L_{c-1,j} = 0$.

The above two cases can be summarized as follows:

The token is early on its c-th visit to node j if and only if:

$$t_{c,j} - t_{c-1,j} \leq TTRT - L_{c-1,j}$$

In that case the asynchronous bandwidth is:

$$TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right) \geq 0$$

and $L_{c,j} = 0$.

Otherwise, the asynchronous bandwidth is null and:

$$\begin{aligned} L_{c,j} &= TTRT - \left[2TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right)\right] \\ &= -\left[TTRT - L_{c-1,j} - \left(t_{c,j} - t_{c-1,j}\right)\right] \end{aligned}$$

Therefore, assuming that node j can manage two variables L_j and t_j, which are correctly initialized at ring startup, in order to compute the asynchronous bandwidth of a TTP node with no asynchronous traffic on token arrival, the following algorithm can be performed:

a) compute $b = TTRT - L_j - \left(now - t_j\right)$;
b) if the latter is positive, it represents the available asynchronous bandwidth; set $L_j \leftarrow 0$;
c) otherwise, the available asynchronous bandwidth is null; set $L_j \leftarrow -b$;
d) set $t_j \leftarrow now$.

Which is exactly what the `Async_Flow_Visit` procedure in the pseudo-code of Figure 3 does.

VII. REFERENCES

[1] R.M. Grow: "A Timed Token Protocol for Local Area Networks", Proc. Electro '82, Token Access Protocols, Paper 17/3, May 1982.

[2] K.C. Sevcik and M.J. Johnson: "Cycle Time Properties of the FDDI Token Ring Protocol", IEEE Transaction on Software Engineering, Vol. SE13, No. 3, pp. 376-385, March 1987.

[3] G. Agrawal, B. Chen, W. Zhao, S. Davari: "Guaranteeing Synchronous Message Deadlines with the Timed Token Medium Access Control Protocol", IEEE Transaction on Computers, vol. 43, No 3, pp. 327-343, March 1994.

[4] M. Hamdaoui and P. Ramanathan: "Selection of Timed Token Protocol Parameters to Guarantee Message Deadlines", IEEE Transaction on Networking, vol. 3, No 3, pp. 340-351, March 1995.

[5] S. Zhang and A. Burns: "Timing Properties of the Timed Token Protocol", Technical Report, Department of Computer Science, University of York March 1994.

[6] S. Zhang and E.S. Lee: "The Worst-Case Scenario for Transmission of Synchronous Traffic in an FDDI Network", Technical Report, Center for Communications Systems Research, University of Cambridge, UK November 1998.

[7] J.P.C. Blanc, L. Lenzini, "Analysis of Communication Systems with Timed Token Protocols Using the Power-series Algorithm", Performance Evaluation, Volumes 27&28, November 1996, pp. 391-409.

[8] D. Dykeman, and W. Bux: "Analysis and Tuning of the FDDI Media Access Control Protocol", IEEE Transaction on Networking, vol. 6, No 6, pp. 997-1010, July 1988.

[9] A.K. Parekh and R.G. Gallager: "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: the Single Node Case". IEEE Transaction on Networking, Vol. 1, No. 3, pp. 344-357, June 1993.

[10] D. Saha, S. Mukherjee, K. Tripathi: "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks". *IEEE Transaction on Networking*, Vol. 6, No. 6, pp. 779-796 (December 1998).

[11] The ATM Forum, Traffic Management Specification, Version 4.1, March 1999.

[12] I. Stoica, H. Abdel-Wahab, and K. Jeffay, "On the Duality between Resource Reservation and Proportional Share Resource Allocation," Proceedings, Multimedia Computing and Networking 1997, SPIE Proceedings Series, Volume 3020 San Jose, CA, pp 207-214, February 1997.

[13] H. Zhang "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks", Proceedings of the IEEE, vol. 83, No. 10, pp. 1374-1396, October 1995

[14] R. Braden, D. Clark and S. Shenker: "Integrated Services in the Internet Architecture: an Overview", RFC 1633, The Internet Society, June 1994

[15] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss: "An Architecture for Differentiated Services", RFC 2475, The Internet Society, December 1998.

[16] S. Floyd and V. Jacobson: "Link-Sharing and Resource Management Models for Packet Networks", IEEE Transaction on Networking, Vol. 3, No. 4, pp. 365-386, August 1995.

[17] J. Bennett and H. Zhang: "Hierarchical Packet Fair Queueing Algorithms", IEEE Transaction on Networking, Vol. 5, No. 5, pp. 675-689, October 1997.

[18] D. Stiliadis and A. Varma: "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms", IEEE Transaction on Networking, Vol. 6, No. 5, pp. 675-689, October 1998.

[19] C. Dovrolis, P. Ramanathan: "A Case for Relative Differentiated Services and the Proportional Differentiation Model", IEEE Network, Vol. 13, No. 5: pp. 26-34, September 1999.

[20] M. Shreedhar and G. Varghese: "Efficient Fair Queueing Using Deficit Round-Robin", IEEE Transaction on Networking, Vol. 4, No. 3, pp. 375-385, June 1996.

[21] M.H. MacGregor and W. Shi: "Deficits for Bursty Latency-critical Flows: DRR++", Proceedings of the IEEE International Conference on Networks (ICON'00), Singapore , September 5 - 8, 2000.

[22] A. Valenzano, P. Montuschi, L. Ciminiera, "Some Properties of Timed Token Medium Access Protocols", *IEEE Transaction on Software Engineering*, Vol. 16, No. 8, pp. 858-869, August 1990.