Worst-Case Analysis of Tandem Queueing Systems Using Network Calculus

Anne Bouillard, Giovanni Stea

CHAPTER 1

WORST-CASE ANALYSIS OF TANDEM QUEUEING SYSTEMS USING NETWORK CALCULUS

ANNE BOUILLARD, [1], GIOVANNI STEA, [2]

[1]Department of Informatics at ENS/INRIA, 45 Rue d'Ulm, 75230 Paris CEDEX 05, France (Anne.Bouillard@ens.fr)

[2]Department of Information Engineering at University of Pisa, Largo L. Lazzarino 1, 56122, Pisa, Italy (g.stea@iet.unipi.it)

Abstract. In this chapter we show how to derive performance bounds for tandem queueing systems using Network Calculus, a deterministic theory for performance analysis. We introduce the basic concepts of Network Calculus, namely arrival and service curves, and we show how to use them to compute performance bounds in an end-to-end perspective. As an application of the above theory, we evaluate tandems of network nodes with well-known service policies. We present the theory for two different settings: a simpler one, called "per-flow scheduling", where service policies at each node discriminate traffics coming from different flows and buffer them separately, and "per-aggregate scheduling", where schedulers manage a small number of traffic aggregates, and traffic of several flows may end up in the same queue. We show that, in the latter case, methodologies based on equivalent service curves cannot compute *tight* delay bounds and we present a different methodology that relies on input-output relationships and uses mathematical programming techniques.

Keywords. Network Calculus, Worst-case Delay, Performance Bounds.

1.1 Introduction

Many of today's networked applications rely on the underlying network providing Quality of Service (QoS) guarantees. For instance, playback applications (such as video or voice) require bounds on the worst-case traversal time of packets, on an end-to-end basis, the deadline being the playback instant of the video/voice packet. Moreover, there is a growing interest in new types of networked applications requiring firm end-to-end delay guarantees. For instance, those based on remote sensing and control, favored by the emerging of the Machine-to-Machine (M2M) communication paradigm. Some of these are safety-critical, e.g., assisted driving, smart electrical grid control, factory automation, telemedicine, etc., hence packets cannot miss their deadlines without serious consequences. A common trait of all the above application is that they are expected to run in a *multi-hop* networked environment, where their traffic will experience multiple queueing, contending for resources (i.e., bandwidth and buffers space) with traffic from heterogeneous applications.

Despite the abundance of literature on the matter, in the past decades the problem of QoS guarantees has been tackled by network providers mainly through blind *overprovisioning*: overdimensioning links with respect to the traffic they had to carry, achieving utilization factors far below saturation, was in fact enough to ensure, at least statistically, that end-to-end delays were small enough to allow delay-sensitive applications to run smoothly. This trend cannot go on forever for several reasons: first, bandwidth availability traditionally spawns bandwidth-hungry applications, leading to an arms race which will settle on links being close to saturation before a new technological breakthrough occurs. Second, network overprovisioning is not cost-effective, as it requires both larger investments and higher operational expenditures. With respect to the latter, network energy efficiency has recently become an issue, attracting itself a noticeable amount of research: one of the keys to energy efficiency is to disable the portions of a network that are not strictly necessary to ensure a given level of service, which amounts to keeping the rest of the network much closer to saturation than an overprovisioning strategy would otherwise have. Last, but not least, some services just cannot do without firm, a priori guarantees, as opposed to measurement-based, a posteriori statistical assurances. To begin with, even non-critical applications such as high-definition IPTV may require such guarantees: for instance, if they are to be *sold* to a large consumer audience which might not tolerate even occasional video glitches. Moreover, it is self evident that, with remote sensing/control or safety-critical applications, delay guarantees define the very correctness of the applications' behavior.

Network Calculus (NC) is a theory that allows one to compute bounds on significant quantities (most notably, the end-to-end delay) in a queueing network. It has been devised in the '90s, thanks to the seminal works of Cruz [21, 22], Chang [19], and Le Boudec and Thiran [26]. It relies on a deterministic service and traffic characterization (as opposed, for instance, to Markov Chains, which rely on stochastic characterizations), hence it is particularly useful for assessing worst-case measures, such as the maximum delay. NC has already been used in several domains. As far as the Internet is concerned, the Guaranteed Service of the IP IntServ architecture

[15], standardized in 1994, is based on delay bounds computed through NC. More recently, NC has been used to assess the performance of AFDX avionic networks [1], which require a certification of the maximum end-to-end delay before being made operational. Furthermore, it has been applied to design or assess the performance of Network-on-Chips [24], Systems-on-Chips, e.g., [18], Wireless Sensor Networks [25, 32, 39], Wireless Mesh Networks [16, 17], and industrial Ethernet installations [37].

Performance analysis through Network Calculus is normally carried out with respect to a single flow of traffic, which traverses a tandem network from its source to its destination, and contends for bandwidth with similar flows at each hop (e.g., the output port of a switch or router). Contention is usually arbitrated by a scheduling policy (e.g., non-preemptive strict priority, round-robin, etc.), in either of the following settings: a *per-flow scheduling* approach, where each flow has a dedicated FIFO queue, and the scheduler determines the flow that gets access to the output bandwidth at each time, or an *aggregate-multiplexing* approach, where traffics from different flows may be buffered in the same queue (which may or may not also compete with others for access to a link's bandwidth). In this last case, embodied in Internet standards such as IP DiffServ architecture [5], the buffering policy becomes relevant as well: if flows are buffered FIFO, then what gets in will eventually get out, hence the delay of a packet will stay finite as long as the queue does not grow indefinitely, although it will depend on the arrival profile of all the flows that are buffered in the same queue. On the other hand, if non-FIFO queueing is adopted, a packet may sit forever in the buffer, being constantly pushed at the back by other incoming packets, even if the queue length stays finite.

It turns out that the main watershed in NC modeling is whether a per-flow or an aggregate-multiplexing network is analyzed: in the former case, it is easy to derive service guarantees for single flows at a node, and to compose these guarantees along a path to compute an end-to-end guarantee. Perhaps the most interesting feature of NC is that – in this case – the composition of service guarantees preserves *tightness*: if the individual service guarantees allow one to compute the true worst-case delay at each node, then their composition will yield the true end-to-end worst-case delay, and not just a bound on the latter: in other words, all the relevant information is preserved when composing service guarantees. In the second case, i.e., aggregate-multiplexing, per-node guarantees are harder to obtain, to begin with, and composing them along a multi-hop path leads to a loss of tightness [2, 34]: the end-to-end delay that one gets by composing per-node service guarantees is often a *loose*, pessimistic upper bound on the worst-case delay. Thus, in this case, a different method is required, which does not exploit composition, but relies instead on mathematical programming techniques.

This chapter presents the theory of Network Calculus in a tutorial way, focusing on its practical implications whenever possible, using examples that a reader who is mildly familiar with packet scheduling will find easy to understand. It is divided into two parts: in the first part we explain the basics of NC, describing the concepts of service curve and arrival curve. Then, we show how to derive performance bounds for a network flow in a per-flow scheduling approach. The second part builds on top of these basic concepts to explain how to compute the worst-case delay in an

aggregate-multiplexing network, describing both the cases of FIFO and non-FIFO queueing.

Despite the fact that this chapter is written using computer networks as a case study, most of the modeling shown herein can be applied – with few modifications – to other contexts where there is contention for resources and scheduling, notably distributed systems.

1.2 Basic Network Calculus Modeling: per-flow scheduling

Assume that you are observing the traffic of a given flow at the *input* and *output* of a network element, e.g., a scheduler. That network element may, of course, be traversed by several flows simultaneously, however we are interested in what happens to a particular flow, which we call the *tagged flow*. Denote with $A(t)$ and $D(t)$ the functions of time that count how much traffic belonging to the tagged flow has been observed in $[0, t)$, respectively at the input and output of the network element. We call these the Cumulative Arrival Function (CAF) and the Cumulative Departure Function (CDF) for the flow. Assume that the system is time-continuous, i.e., arrivals and departures can occur at any time, unlike in a slotted system (different, simpler models can be used to describe the latter). CAFs and CDFs need not be continuous, to reflect that a discrete quantity of traffic (e.g., a burst) can arrive and leave at once.

Furthermore, assume that the network element is lossless, that it does not generate traffic, and that it serves the traffic of the tagged flow in FIFO order. For a network element to be lossless, traffic must be *buffered*, and the buffer must be large enough not to overflow. For now, we will just state that overflows do not occur, and later on we will quantify the buffer required in order for this hypothesis to hold.

Obviously enough, both $A(t)$ and $D(t)$ – being cumulative functions of time – must be *wide-sense increasing*. Furthermore, for the element to be causal it must be $A(t) \geq D(t)$. When there is no ambiguity, we will write $A \geq D$ to denote that the latter holds for any time instant t. Unless stated otherwise, all cumulative functions R are defined as $R : \mathbb{R}_+ \to \mathbb{R}_+$, are left-continuous and such that $R(0) = 0$.

If one can measure the CAF and CDF of a network element, then it is fairly easy to compute some quantities, namely the delay of a bit and the backlog at any one time. As Figure 1.1 shows, the *vertical* distance $B(\tau) = A(\tau) - D(\tau)$, which is always non-negative, is the element's backlog at time τ. Conversely, the *horizontal* distance between point (τ, a) on A and point (τ', a) on D, i.e., $d(\tau) = \tau' - \tau$, is the delay of the bit that enters the element at time τ. Note that, if the CDF has a plateau at quota a, then the definition still holds, provided that we set $\tau' = \inf\{s : D(s) \geq a\}$. More formally, the delay of a bit arriving at time τ is equal to:

$$d(\tau) = \inf\{s \geq 0 : A(\tau) \leq D(\tau + s)\}.$$

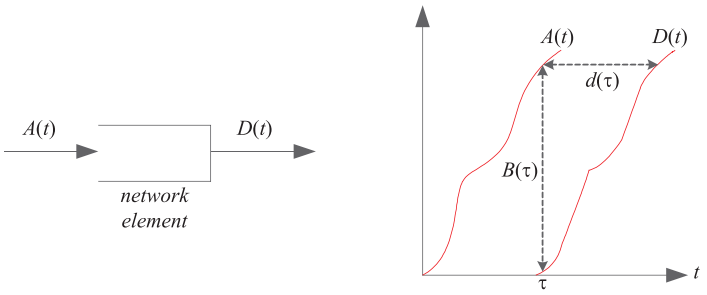

Figure 1.1 CAF and CDF at a network element

1.2.1 Service curve

If a network element provides some QoS guarantees, as most schedulers do (e.g., a minimum departure rate), then it stands to reason that – given an input CAF A – the possible output CDFs D should be *lower-bounded*, and that the lower bound should be a function of *both* the CAF *and* some inherent property of the network element itself. In fact, we say that the network element can be modeled through the *service curve* β if:

$$\forall t \geq 0, \quad D(t) \geq \inf_{0 \leq s \leq t} (A(s) + \beta(t-s)). \tag{1.1}$$

The right-hand side of eq. (1.1) is the lower bound to the CDF D we were just talking about, and in this case the flow is said to be guaranteed the (minimum) service curve β. The infimum at the right side of eq. (1.1), as a function of t, is called the *min-plus convolution* of A and β, and is denoted by $A \otimes \beta$. The name "min-plus convolution" stems from the fact that the operation resembles systems theory convolution, if one replaces the sum with the infimum operator and the product with the sum. The service curve of a network element is not something that can be observed by just measuring one CAF and the related CDF. It is instead a property of the network element, such that eq. (1.1) holds for *any* CAF. In fact, a network element may be stateful (schedulers usually are), hence one trajectory alone cannot provide much information regarding worst-case behavior.

Before delving deeper into the properties of the convolution operation, which are important to understand the rest of the chapter, we show how to derive a service curve in a couple of practical cases, namely strict-priority and round-robin schedulers.

1.2.1.1 *Example – strict-priority scheduler*

Assume that the tagged flow is being scheduled by a *non-preemptive strict-priority* scheduler. The latter serves one packet from the highest-priority backlogged flow, and waits for a packet to be transmitted before making another decision (hence being non-preemptive). Assume that the tagged flow is the top-priority one, and that there are other flows with lower priorities. Let C be the speed of the link managed by the scheduler, and let M be the Maximum Transfer Unit on the link (i.e., the maximum-sized packet allowed on the

link). Assume that the tagged flow sends an infinite amount of traffic in a single burst at time $t = 0$, i.e., $A(t) = \delta_0(t)$, and let us compute what the lower bound on its CDF will be. Notation $\delta_x(t)$ denotes a function which is null for $t \leq x$, and infinite for $t > x$. In a worst-case scenario, at time 0 the scheduler is busy serving some other lower-priority flow. The server may be at it for a maximum time equal to $T = M/C$, then – by the very definition of strict priority – it must switch to serving the tagged flow. Once it starts (since the tagged flow is always backlogged), it will keep transmitting traffic from it. Therefore, the CDF for the tagged flow will be $D(t) = (C \cdot t - M)_+$. Notation x_+ means $\max(x, 0)$. Figure 1.2 depicts the above scenario. Now, it is $D(0) = A(0) = 0$, hence we can write:

$$D(t) \geq A(0) + (C \cdot (t - 0) - M)_+ \tag{1.2}$$

By setting $s = 0$ in eq. (1.1) , eq. (1.2) implies that $\beta(t) = (C \cdot (t - 0) - M)_+ = C \cdot (t - T)_+$ is a service curve for the tagged flow. The shape of this service curve is very common in practice, as we shall see, and its name is *rate-latency* curve: the rate is the long-term slope, i.e., C, and the latency is its horizontal offset, i.e., T.

Note that, to keep the parallel with classical systems theory, the service curve has been computed by giving an *impulse* as an input to the system and measuring its worst-case output. Therefore, the service curve can be thought of as a *worst-case impulse response* for a network element.

Finally, note that the above reasoning cannot be generalized to flows having lower priority. In fact, unless *all* higher-priority flows are somewhat *regulated*, i.e., prevented to keep their queues always backlogged, lower-priority flows will simply starve. This means that (without calling in additional mechanisms, at least) the worst-case impulse response for a lower-priority flow in a strict-priority scheduler is flat.

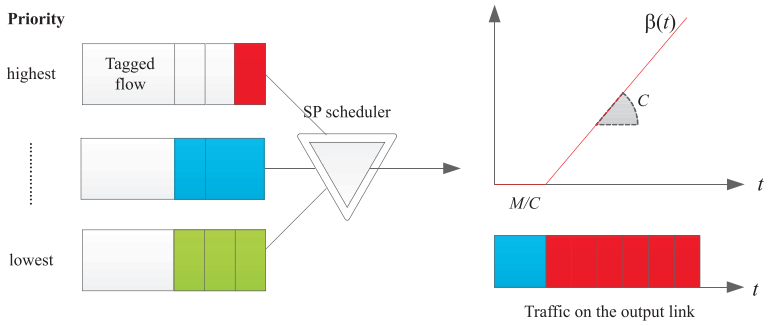

Figure 1.2 Strict-priority scheduling scenario

1.2.1.2 Example – round-robin scheduler Assume that the tagged flow is served by a weighted round-robin scheduler. The scheduler manages N queues, one for each flow, and each flow i has a quantum ϕ_i, a positive quantity representing the amount of time that the server spends in transmitting traffic from queue i before moving on

to queue $(i+1) \mod N$. Let the link speed be equal to C again, and assume for simplicity that traffic may be arbitrarily fragmented, so that each flow may fill the quantum entirely if it has enough backlog. Without loss of generality, assume that the tagged flow is flow 1, and let its CAF be again the impulse at time 0, $A(t) = \delta_0(t)$. The worst-case CDF is observed when: a) all flows $2 \dots N$ are always backlogged, and b) the server is at the beginning of the service period for flow 2 at time 0. In this case, the CDF of tagged flow 1 will be null until time $T_1 = \sum_{i=2\dots N} \phi_i$, and then increases with a slope C until time $P = T_1 + \phi_1 = \sum_{i=1\dots N} \phi_i$. The same pattern repeats indefinitely, as shown in Figure 1.3.

Using the same reasoning as in the previous example, one may conclude that the worst-case impulse response CDF is again the system's service curve for the tagged flow. Note that the long-term guaranteed rate of the tagged flow r_1 can be expressed as a proportion of the quanta of the various flows, i.e., $r_1 = C \cdot \phi_1 / (\sum_{i=1\dots N} \phi_i)$. This holds, rather obviously, for any flow j being scheduled.

Note that the shape of the service curve for a round-robin scheduler is not a rate-latency one. However, it is easy to see that one rate-latency curve exists that bounds that service curve from below: it is the one with a latency equal to T_1 and a rate equal to r_1, i.e., $\beta(t) = r_1 \cdot (t - T_1)_+$. Curve β still verifies eq. (1.1), since it bounds from below a curve that does. Moreover, it is the largest rate-latency one that does, since it touches the worst-case impulse response CDF at abscissas $T_1 + k \cdot P, \forall k \geq 0$. Therefore, one may use β as a service curve for the tagged flow as well. We will see that this makes computations simpler, but it comes with a price.

Finally, let us observe what happens if all the quanta are doubled. On one hand, the minimum guaranteed rate r_i of each flow will remain the same, since it depends on the *ratio* of the quanta. On the other hand, the *latency* of the β service curve will double: this hints at the fact that each flow (including the tagged one) will have the same throughput, but – in general – higher delays, something that we will prove formally later on.

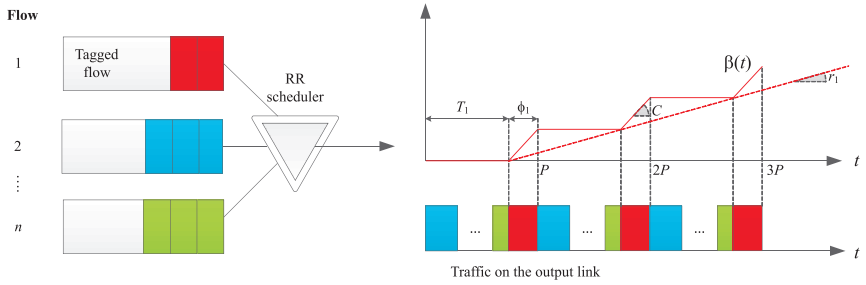

Figure 1.3 Round-robin scheduling scenario

1.2.1.3 Other types of service curves Links and nodes (i.e., routers or switches) can also be seen as network elements providing service-curve guarantees. More specifically, following the same approach used in the previous two cases, it is fairly

easy to observe that a link with a constant rate C (often called a *constant-rate server*) has a service curve $\beta(t) = C \cdot t$, which we also represent by λ_C. By the same token, a link with a *minimum* rate C (e.g., a wireless link that can switch its rate between a minimum and a maximum) has the *same* service curve. The two are clearly not equal in all respects: with the former, we can obtain the exact CDF given the CAF, whereas with the latter we can only know a *lower bound* on the CDF. However, as far as service guarantees are concerned, the two can be described in the same way. It can also be observed that several *fair queueing* schedulers, including Weighted Fair Queueing [31], Deficit Round Robin [36, 28] and others [38] all exhibit rate-latency service curves.

Some elements (e.g., network switches) exhibit a bounded transit delay, at least under appropriate testable hypotheses (e.g., in the absence of overload). This is also the case, for instance, of deadline-based schedulers, such as Earliest Deadline First scheduling, if the admission control (or schedulablity) test is passed. For these elements, the service curve is the *delayed impulse* δ_T.

Figure 1.4 reports both the above service curves. It is interesting to observe that both a minimum-rate and a delay service curve are special cases of a rate-latency service curve. More specifically, if $\beta_{R,T}$ denotes a rate-latency curve with a rate R and a latency T, it is $\lambda_R = \beta_{R,0}$ and $\delta_T = \beta_{\infty,T}$.

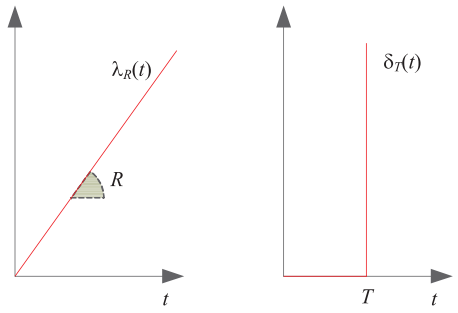

Figure 1.4 A constant/minimum-rate-server service curve (left) and a delay-element service curve (right)

1.2.1.4 Useful properties of the convolution operator

The result of the convolution operation $A \otimes \beta$ can easily be found graphically. Recalling eq. (1.1), it is easy to see that D can be found by sliding the service curve over the CAF, as shown in Figure 1.5, and taking the minimum of the results at each time instant.

Some properties of convolution that will be used later on in this chapter are the following:

- convolution is commutative and associative: $A \otimes B \otimes C = (B \otimes A) \otimes C$;
- the neutral element with respect to convolution is the impulse δ_0: $A \otimes \delta_0 = A$;

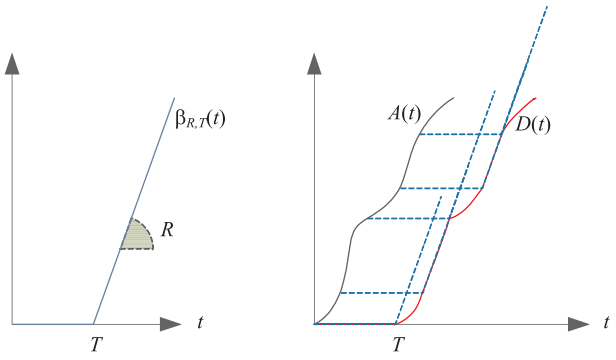

Figure 1.5 Graphical interpretation of the convolution operation

- conversely, convolution with a delayed impulse δ_T just shifts a function right by T: $(A \otimes \delta_T)(t) = A(t-T)$;
- the convolution of two *concave* curves taking value 0 at 0 is equal to their *minimum*: if A, B are concave, $A \otimes B = \min\{A, B\}$.

The proof of the above properties is trivial and is thus left to the reader. Another useful property of convolution is given by the following lemma:

Lemma 1 *[6] If f and g are left-continuous and wide-sense increasing, then for all $t \geq 0$, there exists $s \leq t$ such that $f \otimes g(t) = f(s) + g(t-s)$.*

1.2.1.5 Composition of service curves Suppose now that the tagged flow traverses a *tandem* of n network elements: for instance, a multi-hop path in a network domain, at each hop of which the flow is scheduled by some scheduler, as shown in Figure 1.6. Suppose that, at each network element i, the tagged flow is guaranteed a service curve $\beta^{(i)}$. Clearly, the CDF at node i of the path will be equal to the CAF at node $i+1$, $\forall i < n$. Therefore, we simply use $F^{(i)}$ to denote the CDF at node i, and assume that $F^{(0)}$ is the CAF at node 1. The service curve property ensures that $F^{(i)} \geq F^{(i-1)} \otimes \beta^{(i)}$. Therefore, by the associativity of convolution, we get:

$$F^{(N)} \geq F^{(0)} \otimes \left\{\beta^{(1)} \otimes \beta^{(2)} \otimes \ldots \otimes \beta^{(n)}\right\} = F^{(0)} \otimes \left\{\bigotimes_{i=1}^{n} \beta^{(i)}\right\} \tag{1.3}$$

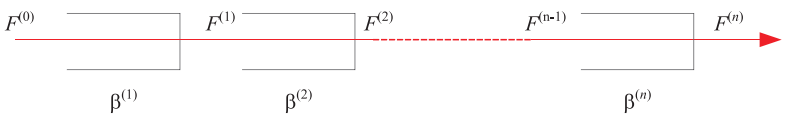

Figure 1.6 A tandem of network elements traversed by a flow

The last term in eq. (1.3) can be regarded as the service curve that *the whole tandem* offers to the tagged flow. In fact, it computes a lower bound to the CDF *at the exit* of the tandem, given the CAF at the ingress. Note that no hypothesis is required on each network element, other than that it provides the tagged flow with a service curve guarantee. For instance, the above result holds for a two-hop path where the tagged flow traverses a strict-priority scheduler and a round-robin scheduler. Therefore, the only thing that is needed to allow for multi-node analysis is to compute the convolution of the service curves at each node.

Convolution of rate-latency service curves is particularly easy to compute. Let us compute the convolution of $\beta^{(i)} = \beta_{R_i,T_i}$, $i = 1, 2$. Without loss of generality, assume that $R_1 \geq R_2$ (convolution is in fact commutative). Recalling Figure 1.5, one has to "slide" one rate-latency curve along the other and consider the minimum ordinate for each abscissa. As shown in Figure 1.7, this implies that the resulting curve will be null until $T = T_1 + T_2$. Then, the result will increase with a rate equal to $R = \min\{R_1, R_2\}$.

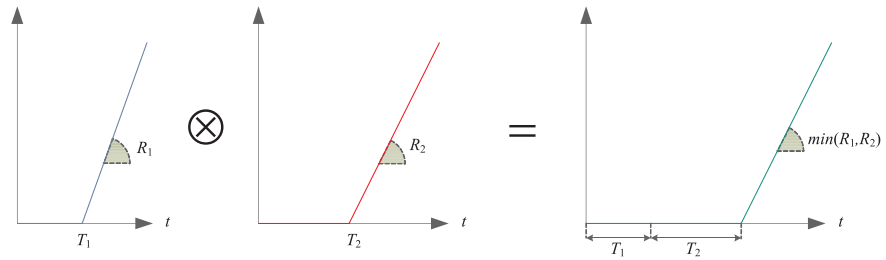

Figure 1.7 Convolution of two rate-latency service curves

By iterating the reasoning n times, we obtain that the convolution of n rate-latency service curves is as follows:

$$\bigotimes_{i=1}^{n} \beta_{R_i,T_i} = \beta_{\min_i\{R_i\},\sum_{i=1}^{n} T_i} \tag{1.4}$$

Equation (1.4) makes sense intuitively: the minimum guaranteed rate for a flow traversing a tandem of node is the *minimum* among those guaranteed at each node. Furthermore, in a worst-case scenario, the flow will experience the maximum latency at each node, hence latencies should add up.

Finally, we observe that a rate-latency service curve $\beta_{R,T}$ may be obtained as the convolution of a minimum-rate service curve λ_R and a delay service curve δ_T.

1.2.1.6 Strict service curves In some cases, network elements provide tighter service guarantees than those captured by the service curve property. A common guarantee is that of the *strict service curve*. We say that a network element offers a strict service curve β to a flow if, for any period $]s, t]$ during which the flow is

backlogged, then it is:

$$D(t) - D(s) \geq \beta(t - s). \tag{1.5}$$

It is straightforward to prove that if β is a strict service curve, then it is also a service curve. In fact, since eq. (1.5) holds for any backlogged period, it also holds if s is the beginning of a backlogged period, which we denote $start(t)$. In this case, it is $A(s) = D(s)$ by definition, hence we have found one instant when $D(t) \geq A(s) + \beta(t - s)$, which implies eq. (1.1). The reverse, however, is false, thus the strict service curve property is – in fact – a stricter guarantee than the service curve.

With reference to the examples of the previous subsections, the service curves of a strict-priority and round-robin scheduler, and of a constant- and minimum-rate server, are also *strict* service curves. On the other hand, the delay element service curve is not a strict service curve. The proof of this result can be found in [26, Chapter 7].

An interesting – though unfortunate – result is that the strict service curve property is *not* preserved through composition: if $\beta^{(1)}$ and $\beta^{(2)}$ are strict service curves offered by two nodes traversed by the tagged flow, then we cannot say that the two-node tandem offers a strict service curve equal to $\beta = \beta^{(1)} \otimes \beta^{(2)}$. In fact, it turns out that β *is* a service curve (because strict service curves are service curves, in any case), but *not a strict one*.

Over the years, several other proposals of service guarantees have appeared in the literature, with the aim to find stricter guarantees than the service-curve ones, which are still preserved through composition – unlike the strict service curve property. A comprehensive description on the subject can be found in [6]. Strict service curves will come again into play later in this chapter, when we examine aggregate-multiplexing architectures.

1.2.2 Arrival Curve

The theory expressed so far allows us to compute a lower bound on the CDF of a tagged flow at the exit of a tandem of network elements, given its CAF and the service curve of each element. Furthermore, by plotting the CAF and the CDF on the same reference, we can assess the delay of each bit, and the amount of traffic in transit (i.e., the backlog) at each time instant. To compute the maximum delay that a bit of the tagged flow experiences, we must find the maximum horizontal distance between the CAF and CDF. Obviously, the maximum delay will depend on the CAF itself: in general, assuming a rate-latency service curve, we can say that the delay increases whenever the slope of the CAF exceeds the minimum guaranteed rate of the service curve, whereas it decreases when the opposite is true. If we can *limit* the rate of the arrivals, then, in such a way that – in a long term – the arrival rate will not exceed the minimum guaranteed rate, we should be able to compute a finite bound on the maximum delay. A common way to represent constraints on the arrivals of a

flow in Network Calculus is the concept of *arrival curve*.[1] A wide-sense increasing function α is said to be an *arrival curve* for a flow characterized by a cumulative function A (or, equivalently, A is α-*upper constrained*) if:

$$\forall \tau \leq t, \quad A(t) - A(\tau) \leq \alpha(t - \tau). \tag{1.6}$$

The alert reader may check that eq. (1.6) is equivalent to $A \leq A \otimes \alpha$. Graphically speaking, the constraint can be visualized by sliding the arrival curve over the CAF: if the CAF never crosses one arrival curve, then eq. (1.6) holds. This is shown in Figure 1.8.

A commonplace network element that enforces an arrival curve is the *leaky-bucket shaper*. The latter is often placed at the ingress of a network path, in order to limit the amount of traffic injected by the flow, thus preventing it from causing excessive queueing in the network. A leaky-bucket arrival curve is characterized by a *sustainable rate* ρ and a *burst size* σ, and its expression is $\gamma_{\sigma,\rho}(t) = \sigma + \rho t$ if $t > 0$ and $\gamma_{\sigma,\rho}(0) = 0$. Roughly speaking, it means that the flow is allowed to inject traffic at a rate up to ρ. It can only exceed that rate by a maximum of σ bits over *any* interval of time. This means that a flow cannot buy any extra credit by not sending traffic at its maximum allowed rate for some time. This is evident in Figure 1.8, where, at time t_1, the CAF of the flow is considerably below $\alpha(t)$, but is still subject to the tighter upper constraint represented by $\alpha(t - t_1) + A(t_1)$ in any case.

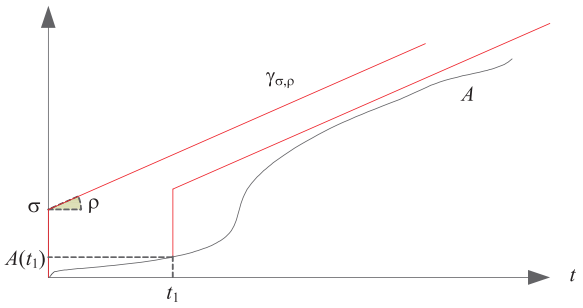

Figure 1.8 Leaky-bucket arrival curve

In most practical cases, arrival curves are concave (such as the leaky-bucket one). It is not uncommon to find *piecewise-linear, concave* arrival curves (taking value 0 at 0), which are practically implemented by using multiple leaky-bucket shapers. For instance, the Guaranteed Service of the IntServ architecture [15] provides a flow with delay guarantees under the hypothesis that its traffic is shaped by a double leaky-bucket, whose arrival curve is depicted in Figure 1.9: a first stage limits the *peak rate* to p with a burst size of one packet M, and a second stage allows for a smaller *sustainable rate* ρ, with $\rho \leq p$ and a burst σ, with $\sigma \geq M$. This means that only the traffic that obeys *both* the peak-rate and the sustainable-rate constraints will

[1] Mind the distinction between cumulative arrival *function* and arrival *curve*.

be allowed in. A piecewise-concave arrival curve can be constructed as the *minimum* of affine curves. For instance, the one in Figure 1.9 can be easily observed to be $\alpha = \min\{\gamma_{M,p}, \gamma_{\sigma,\rho}\}$. Therefore, by the properties of convolution, we can also write $\alpha = \gamma_{M,p} \otimes \gamma_{\sigma,\rho}$. This means that any piecewise concave arrival curve can be written as the convolution of as many affine arrival curves as its linear pieces.

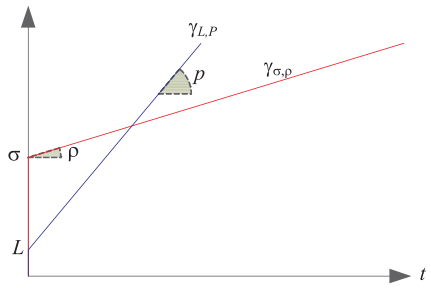

Figure 1.9 Double leaky-bucket arrival curve

1.2.3 Delay and backlog bounds

Knowing both the arrival curve α and service curve β of a flow allows one to compute a bound on the backlog and delay. The bounds are the following (proofs are omitted, and the interested reader can find them in [26, Chapter 1]):

- a bound on the delay is given by the maximum horizontal deviation between α and β:
$$h(\alpha, \beta) = \max_{t \geq 0}\{\inf s \geq 0 : \alpha(t) \leq \beta(t+s)\};$$
- a bound on the backlog is given by the maximum vertical deviation between α and β:
$$v(\alpha, \beta) = \max_{t \geq 0}\{\alpha(t) - \beta(t)\}.$$

The meaning of the above expressions (also shown in Figure 1.10) is the following: given a CAF that conforms to the arrival curve α, fed as input to a system whose service curve is β, the maximum delay (backlog) that a bit experiences will not exceed the ones reported above. The backlog bound can thus be used to dimension the queue at a node so that no overflows occur.

For instance, for a leaky-bucket-shaped flow, whose arrival curve is $\gamma_{\sigma,\rho}$, traversing a rate-latency network element whose service curve is $\beta_{R,T}$, the bounds on the delay and backlog are:

$$d = \begin{cases} +\infty & R < \rho \\ \sigma/R + T & R \geq \rho \end{cases}, B = \begin{cases} +\infty & R < \rho \\ \sigma + T \cdot \rho & R \geq \rho. \end{cases} \tag{1.7}$$

In fact, if we allow the flow to send traffic *faster* than the minimum guaranteed rate (i.e., $R < \rho$), then queues may build up indefinitely, hence no finite bound on the delay and the backlog can be enforced. This also means that the maximum finite delay bound that a flow with a $\gamma_{\sigma,\rho}$ arrival curve can be enforced is $d_{\max} = \sigma/\rho + T$, and that the delay bound can be controlled by overallocating the rate to the flow *or* by reducing the flow's latency. Conversely, B is largely determined by the flow's burst σ, which also bounds B from below, and it can only be controlled by reducing the latency (overallocating rate has no effect). From now on, unless specified otherwise, we will always assume that $R \geq \rho$, so that bounds are finite.

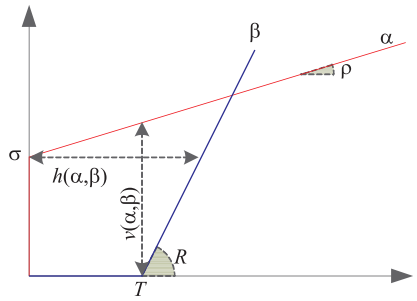

Figure 1.10 Backlog and delay bounds

Note that, since a service curve may represent either a network element, or a *tandem* thereof, end-to-end bounds can also be computed. For the delay, we can compute a bound on the worst-case traversal time of the tandem. In the case of a backlog bound, if the service curve represents a tandem of network elements, then we lack the information on where exactly that backlog is located (i.e., how it is partitioned among the various hops). An obvious workaround is to assume that a maximum backlog equal to B may occur at each node in the tandem, but we can do better than that. Let us introduce the concept of *output arrival curve*, i.e., an arrival curve that constrains the CDF of a flow. The following result is proved in [26, Chapter 1]:

- the output arrival curve is $\alpha^*(t) = \alpha \oslash \beta(t) = \min_{s \geq 0}\{\alpha(t+s) - \beta(s)\}$.

For instance, a leaky-bucket flow traversing a rate-latency service curve will have an output arrival curve equal to:

$$\alpha^*(t) = \gamma_{\sigma,\rho} \oslash \beta_{R,T}(t) = \gamma_{\sigma+T\cdot\rho,\rho}. \tag{1.8}$$

The $\oslash$ operator is called *min-plus deconvolution*. Note that, in this case, the output arrival curve has a *larger* burst than the (input) arrival curve, and – more specifically – that burst is equal to the worst-case backlog at the node. This is not fortuitous: in fact, it may well happen that the server behaves like an "infinite" server, i.e., starts serving traffic at an infinite speed, at the time at which the maximum backlog occurs (we will show in a minute that B is actually the *maximum* backlog). This implies

that, at the output, a burst equal to the maximum backlog must be observable in practice. This also confirms a remarkable phenomenon, often observed in networking: *queueing may increase the burstiness of a flow*. In fact, as a flow traverses a network, the alternance of periods where it is served at full speed and periods where it is not served (typical, for instance, of round-robin schedulers), creates bursts even when the original traffic is smooth. We can use the above result to compute tighter backlog bounds at each node in a tandem, where the CDF at a node is the CAF at the subsequent one. Call $\alpha^{(j)}$ the arrival curve at the output of node j of an n-node tandem, and assume $\alpha^{(0)} = \gamma_{\sigma,\rho}$ to be the arrival curve at the input. Then, assuming that $\rho \leq \min_{1 \leq i \leq n}\{R_i\}$, the backlog bound at node j is the burst of the following arrival curve:

$$\alpha^{(j)} = \gamma_{\sigma,\rho} \oslash \left(\bigotimes_{i=1}^{j} \beta_{R_i,T_i} \right) = \gamma_{\sigma+\sum_{i=1}^{j} T_i \cdot \rho,\rho}. \tag{1.9}$$

Moreover, the backlog bound for the whole system is equal to $\sigma + \sum_{i=1}^{n} T_i \cdot \rho$, i.e., to the backlog bound at node n. This means that the buffer space required for lossless operation increases from node 1 to node n, and at node n it is equal to the maximum amount of traffic in transit at any time in the whole tandem.

We now show that the above bounds are *tight*. We do this by constructing a *worst-case scenario*, i.e., a trajectory of the CAF and CDF of the system such that their maximum horizontal/vertical distances are those predicted by the equations. We limit ourselves to a leaky-bucket-shaped flow traversing a rate-latency service curve, hence making reference to eq. (1.7). However, the result holds for arbitrary arrival and service curves. Assume that the CAF is the *greedy* function, i.e., $A = \gamma_{\sigma,\rho}$. In other words, the flow sends as much traffic as allowed by its arrival curve, starting at time $t = 0$. Furthermore, assume that the network element (e.g., a strict-priority scheduler) is *lazy* (or *exact*), meaning that it behaves so that $D = A \otimes \beta$, with $\beta = \beta_{R,T}$. The CDF can be computed algebraically using some of the properties of the convolution operator that we have explained earlier, i.e.:

$$D = \gamma_{\sigma,\rho} \otimes \beta_{R,T} = \gamma_{\sigma,\rho} \otimes \delta_T \otimes \gamma_{0,R} = \min\{\gamma_{\sigma,\rho}, \gamma_{0,R}\}(t - T). \tag{1.10}$$

Figure 1.11 shows the CAF and the CDF. By visual inspection, it is straightforward to observe that $d = d(0^+)$ and $B = A(T) - D(T)$ when $R \geq \rho$. Conversely, when $R < \rho$, then the horizontal and vertical distance between A and D are increasing functions of t, hence the bounds are infinite. Thus, we have a scenario where the bounds of eq. (1.7) are attained.

Therefore, in this case, B is the *worst-case backlog*, and d is the *worst-case delay* (WCB/WCD for short, hereafter). Note that the worst-case scenario needs not be unique. For instance, the WCD is the same for: i) any CAF A such that: $A(0^+) = b$, and ii) any CDF D such that $D \leq A$, $D \geq A \otimes \beta$ and $D(d) = b$. The alert reader can easily construct an infinity of scenarios that verify the above two properties. Furthermore, we observe that the WCB needs not be experienced by the same bit that experiences the WCD. The output arrival curve is a tight constraint as well. In fact, we can obtain a CDF $D = \gamma_{\sigma+T\cdot\rho,\rho}(t + T)$ by giving as input to the node the greedy CAF $A = \gamma_{\sigma,\rho}$ and assuming that the node serves traffic at an "infinite" speed after sleeping for its latency T, as already anticipated.

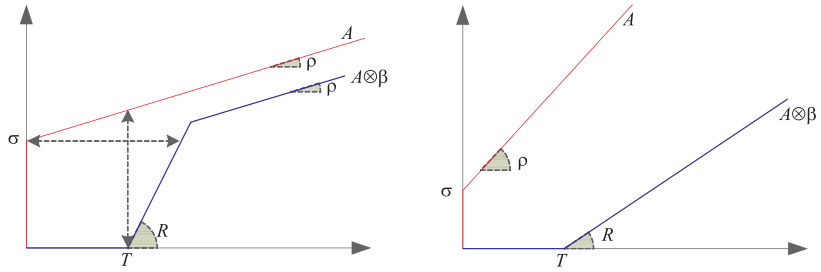

Figure 1.11 Worst-case scenario for a leaky-bucket-shaped flow traversing a rate-latency service curve element. Left: $R \geq \rho$. Right: $R < \rho$.

Another example, this time involving a multi-node scenario, is shown in Figure 1.12. A leaky-bucket-shaped flow traverses a tandem of n rate-latency service curves $\beta^{(i)} = \beta_{R_i,T_i}$, with $\rho \leq \min_{1 \leq i \leq n}\{R_i\}$ so as to ensure that the delay bound is finite. Eq. (1.4) tells us that the tandem is equivalent to a rate-latency service curve, hence a delay bound is:

$$d = h(\gamma_{\sigma,\rho}, \bigotimes_{i=1}^{n} \beta_{R_i,T_i}) = \frac{\sigma}{\min_i\{R_i\}} + \sum_{i=1}^{n} T_i. \tag{1.11}$$

A bit of a CAF experiences a delay equal to d in the following scenario:

- the CAF is greedy, hence $A = F^{(0)} = \gamma_{\sigma,\rho}$;
- node $j = \arg\min\{R_i\}$ is *lazy*, i.e., $F^{(j)} = F^{(j-1)} \otimes \beta^{(j)}$;
- each node $i \neq j$ serves traffic at an *infinite* slope after sleeping for its latency T_i. In other words, it translates a CAF $F^{(i-1)}(t)$ into a CDF $F^{(i)}(t) = F^{(i-1)}(t - T_i)$.

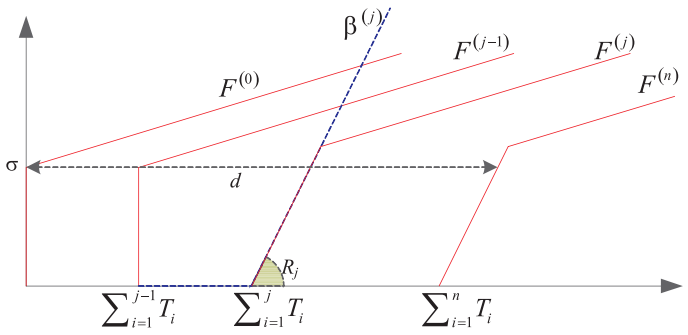

Figure 1.12 Worst-case scenario in a multi-node traversal.

Figure 1.12 shows that the delay of bit $(0^+, \sigma)$ of the CAF is exactly d. The following observations are in order: first of all, the position of node j does not matter:

it can be any node in the tandem, including the first/last. This is in perfect accord with the fact that convolution is commutative and associative. Second, once more, this is not the only worst-case scenario. The alert reader can easily check that the same delay would have been experienced by the same bit, if *all* nodes had been lazy, or if any subset of them including node j had been lazy. This means that node j, acting as a *bottleneck*, is the one whose behavior matters the most. Increasing its rate, for instance, will reduce the WCD, whereas increasing some other node's rate will not. Equation (1.11) has a deep significance. In fact, a bound on the end-to-end delay could also be computed – in principle – by adding up the WCD that the flow can experience at each node. Call $d^{(j)}$ the WCD experienced at node j. The latter can be computed as $d^{(j)} = h(\alpha^{(j)}, \beta^{(j)})$, with $\alpha^{(j)}$ being given by eq. (1.9). After some straightforward algebraic manipulations, we obtain:

$$d' = \sum_{i=1}^{n} d^{(i)} = \sum_{i=1}^{n} (T_i + \frac{\sigma + \sum_{j=1}^{i-1} T_j \cdot \rho}{R_i}). \tag{1.12}$$

It is straightforward to observe that $d < d'$ if $n \geq 2$. Furthermore, the gap between d and d' increases with the number of nodes in the path. Therefore, d is the WCD, and d' is a *loose*, pessimistic bound on the WCD. The pessimism is due to the fact that d contains *one* burst term $\sigma/(min_i\{R_i\})$, whereas d' sums up n terms σ/R_i. In other words, summing up per-node delay bounds does not give you a tight end-to-end delay bounds, *even though the per-node bounds themselves are tight* (which they are, in this case). This is because, by summing up per-node delay bounds, you are implicitly assuming that the traffic of the tagged flow experiences *simultaneously*, and at each node, *both* the scenario that leads to the WCD *and* the one that leads to the output arrival curve. This is clearly impossible: with reference to the previous examples, the latter assumes infinite speed when the former requires the node to be lazy. The principle according to which a tight delay bound should include only *one* burst term (instead of n) is called *Pay Burst Only Once* (PBOO), and is practically embodied in the IP IntServ architecture [15]. Figure 1.13 shows how using the PBOO principle improves on summing up per-node delay bounds, assuming that a flow characterized by a burst $\sigma = 1\,\text{Mb}$ and a rate $\rho = 0.67\,\text{Mbps}$ traverses a tandem of n identical servers, with $T_i = 0.1\,\text{s}$ and $R_i = 10\,\text{Mbps}$. The improvement becomes more significant as the number of servers increases.

As a final observation, we remark that the tightness of the bounds depends on the fact that arrival and service curve are *good* models of the traffic and service constraints for the tagged flow. For instance, given a round-robin scheduler, we might model it using either the periodic staircase service curve shown in Figure 1.3, or its rate-latency lower bound. The latter is clearly easier to manage, as far as computations are concerned, but it allows one to obtain lower bounds on the CDFs that cannot be observed at the output in practice. With reference to Figure 1.14, if we use the rate-latency service curve, we obtain an *upper bound* on the WCD, and not necessarily the WCD itself.

As another example, consider a flow whose arrivals are constrained by the *sporadic traffic model* [14]: the latter consists of a constraint on the maximum packet length M, and on the minimum interarrival time τ. Any sequence of packets no

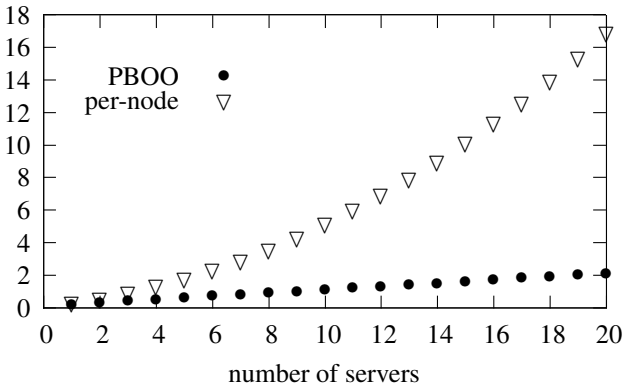

Figure 1.13 Computing the delay using the Pay Burst Only Once principle vs. summing up per-node delays.

longer than M spaced at least τ from each other verifies the constraint. It is straightforward to observe that a leaky-bucket arrival curve $\gamma_{M,M/\tau}$ is an arrival curve for a sporadic flow. However, there is no way that we can get (e.g.) a greedy CAF $A = \gamma_{M,M/\tau}$ without violating the sporadic constraint. Therefore, using a $\gamma_{M,M/\tau}$ arrival curve to model a sporadic flow will allow you to compute *bounds* on the WCD, but not necessarily the WCD itself.

The important lesson is that – under per-flow scheduling – Network Calculus manipulations *do not introduce any pessimism* themselves. If the modeling is exact, then the bounds you compute will be tight.

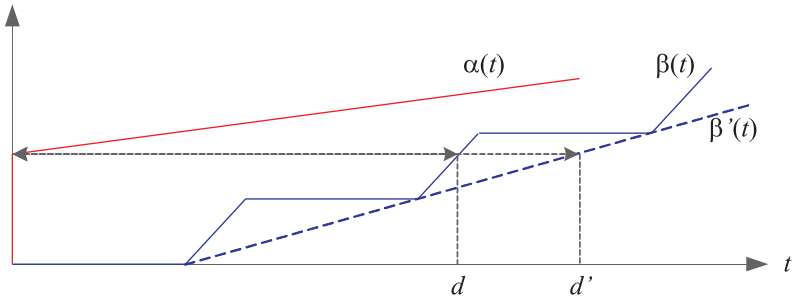

Figure 1.14 Two different delay bounds with a round-robin scheduler.

1.2.4 Numerical examples

We now instantiate some of the above results in a case study. Assume a tagged flow characterized by a burst σ and a rate $\rho = 1$ Mbps traverses a round-robin scheduler, which is shared by N flows (including the tagged one), and manages a link whose

speed is $C = 10$ Mbps. Assume $M = 12$ kbit (roughly corresponding to an Ethernet MTU), and that $\phi_i = M/C$. Note that the above settings imply that all flows have the same long-term rate $R = C/N$, which means that the tagged flow will have a finite bound only as long as $C/N \geq \rho$, i.e., $N \leq 10$. Figure 1.15 shows the delay bound as a function of σ, for several values of N. The figure shows that the delay bound increases in sharp steps whenever the burst surpasses an integer multiple of the quantum. Furthermore, the delay bound depends on the number of cross-flows, which add to the overall latency.

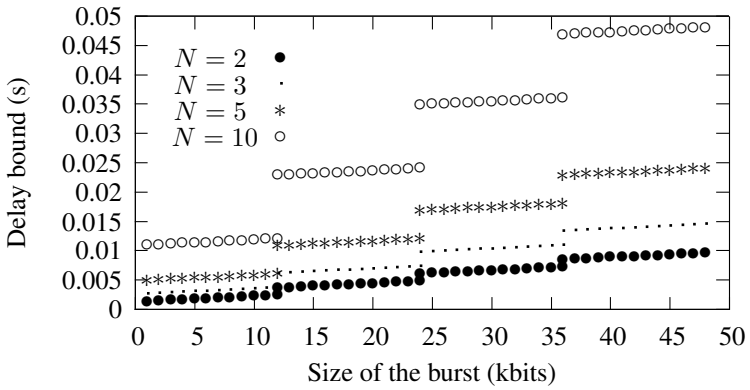

Figure 1.15 Delay bound as a function of the number of flows being scheduled

Assume now instead that $\phi_i = k \cdot M/C$, with $k \geq 1$. Figure 1.16 shows the delay bound as a function of k, for several values of N and $\sigma = 10M$. The figure shows a jaggy behavior, with local minima when the burst is evenly divided by the quantum: this makes sense intuitively, since as soon as the burst exceeds an integer number of quanta, one more round is required to transmit it entirely.

1.3 Advanced Network Calculus modeling: aggregate multiplexing

So far we have described properties of Network Calculus that hold if a flow has a private FIFO queue, a paradigm known as per-flow scheduling (or queueing). Per-flow queueing is, however, not the only option. Aggregate-multiplexing architectures have received an increasing attention in the last fifteen years, following the standardization of the IP DiffServ architecture [5]. In DiffServ, in fact, per-flow queueing is abandoned due to scalability reasons: identifying flows through their 5-tuple, in fact, requires too many operations (e.g., memory reads, hashing, etc.). This becomes a problem when the speed of the link allows only few nanoseconds to make a decision. Moreover, the complexity of advanced schedulers (e.g., Weighted Fair Queueing, WFQ [31]) grows with the number of flows, making it impossible to arbitrate thousands or millions of flows in a packet transmission time at the current link speeds. Instead, in DiffServ, a flow is mapped to a *Class of Service* (CoS) based on a single

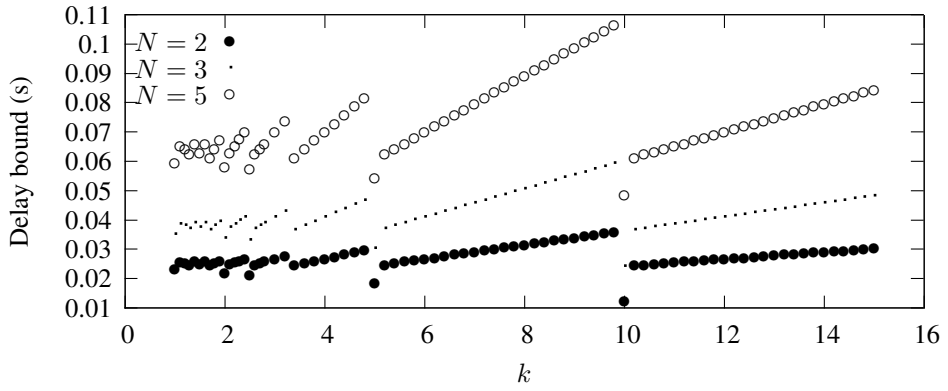

Figure 1.16 Delay bound as a function of the quantum $\phi = k \cdot M/C$.

field in the IP packet (which does away with the first problem, allowing classification with a single memory read operation), and packets of the same CoS get queued in the same queue, regardless of the flow they belong to. Then, a scheduler arbitrates among few CoSs (typically fewer than ten), which allows scheduling decisions to be taken in few nanoseconds. Packets of the same CoS should – in theory – be queued FIFO. However, in practice, this may be difficult to guarantee. In fact, depending on the architecture of the network node, packets arriving at different input lines may traverse different internal paths, with variable delays, before getting queued at the output link. Thus, even if the traffic of each single flow is queued FIFO (which it normally is), at an aggregate level the queueing discipline may appear to be non-FIFO to an external observer.

The drawback of aggregate-multiplexing architectures is that – since no provision can be made for a single flow at a node – it becomes much harder to predict the performance of a flow, especially in a multi-hop path. It should be self-evident, in fact, that the queueing a flow is subject to at the various nodes does not depend on its arrival function alone, as it was under per-flow scheduling, but also on the arrival functions of all the flows that share the same queue. Thus, the least that we can expect is that computations will get more involved in this framework. Moreover, the composition of *aggregates* (i.e., the set of flows that share the same queue at a node) changes from one node to the other, due to the fact that their respective paths merge and diverge based on the destination of each flow. This means – in practice – that we cannot compose the service curves of two neighboring hops of a tagged flow, since we cannot guarantee that the two-hop tandem is *lossless* and *does not create traffic*: flows that leave after the first hop will in fact count as losses, and flows that enter at the second hop will count as created traffic.

However, it also stands to reason that, if all the flows of an aggregate are regulated by some arrival curve (e.g., a leaky bucket), and their overall rate does not exceed

the one of the aggregate service curve, then it must be possible to compute bounds on the backlog and the delay at that node. Furthermore, if the same happens at all the nodes of a tandem, then it should also be possible to compute bounds on the *end-to-end* delay as well. However, the issue of whether these bounds are *tight* becomes relevant.

In the rest of this chapter we show what Network Calculus has to offer for the analysis of aggregate-multiplexing tandem networks. We begin by recalling the existing Network Calculus theorems, and clearly point out why using these does not allow one to compute tight bounds. We then present an alternative method, based on mathematical programming, which allows one to analyze both FIFO and non-FIFO aggregate-multiplexing networks, and that always computes the WCD, though at the price of complex computations. We terminate this chapter with a review of the related work and some numerical examples showing the effectiveness of our approach.

1.3.1 Aggregate-multiplexing schemes

We now present the baseline Network Calculus results related to aggregate-multiplexing networks. We start with "blind" multiplexing, i.e., a multiplexing scheme where we can make no assumptions on what the queueing policy is. Under blind multiplexing, traffic of a given flow may be treated, for instance, at the lowest priority, i.e., be delayed whenever some other flow's traffic is also backlogged. Then, we move to FIFO multiplexing, which is – rather counterintuitively – slightly more complex.

Blind Multiplexing Under *blind multiplexing*, traffic of flows belonging to the same CoS are served in an arbitrary order. This policy then encompasses every possible service policy, and can be used when the latter is not known, hence the name *blind*. Intuitively, the worst-case scenario for a tagged flow will happen when it is given the lowest priority. As a consequence, the service elements are requested to offer *strict* service curves: indeed, in case of stability (i.e., when the aggregate input rate is strictly smaller than the service curve's rate), a strict service curve guarantee ensures that backlogged periods have finite duration. This, in turn, guarantees that the the tagged flow will receive some service. For this reason, it is possible to compute an *equivalent service curve*, that holds for a single flow, based on the aggregate service curve: it suffices to remove from the service offered by the network element the maximum service that can be taken by the other flows, i.e., the sum of their arrival curves. This is shown in the following theorem:

Theorem 1.1 *[26, Chapter 6.1] Consider a node serving two flows,* 1 *and* 2*. Assume that the node guarantees a minimum strict service curve* β *to the aggregate of the two flows and that flow* 2 *has* α_2 *as an arrival curve. Then*

$$\beta^1(t) = [\beta(t) - \alpha_2(t)]_+$$

is a minimal (equivalent) service curve for flow 1*.*

Note that the equivalent service curve is not a strict service curve anymore. However, if the service policy is known to be strict priority (i.e., flow 1 has the lowest

priority and flow 2 the highest), then β^1 and $(\beta - M)_+$ are, respectively, *strict* service curves for flows 1 and 2, where M is the maximum size of a packet.

FIFO Multiplexing When FIFO multiplexing is in place, traffic of flows that have the same CoS are buffered First-Come-First-Served in the same queue. Therefore, a bit of the tagged flow arriving at time t is transmitted only when all the traffic arrived *before* time t (belonging to any flow traversing that node) has been transmitted. Network Calculus allows one to derive *equivalent service curves* for individual flows as well, through the following theorem.

Theorem 1.2 *[26, Chapter 6.2] Consider a node serving two flows,* 1 *and* 2*, in FIFO order. Assume that the node guarantees a minimum service curve β to the aggregate of the two flows and that flow* 2 *has α_2 as an arrival curve. Define the family of functions:*

$$\beta_\tau^1(t) = [\beta(t) - \alpha_2(t-\tau)]_+ 1_{t>\tau},$$

For any $\tau \geq 0$ such that $\beta_\tau^1(t)$ is wide-sense increasing, then flow 1 *is guaranteed the (equivalent) service curve $\beta_\tau^1(t)$.*

Unfortunately, Theorem 1.2 does not lend itself to an intuitive interpretation, as for the blind multiplexing case. The above theorem states that a flow is guaranteed an *infinity* of service curves, each obtained by giving τ a nonnegative value. This implies that a delay bound for flow 1 as $d = h(\alpha, \beta_\tau^1(t))$, is itself a function of τ. Hence, the best delay bound is the *minimum* value of that function, computed on all the values of $\tau \geq 0$ [4].

However, the fact that the node is FIFO (henceforth referred to as the *FIFO hypothesis*) is indeed a strong hypothesis. In fact, it allows one to compute the CDFs of *single flows*, given their CAFs, *without* resorting to equivalent service curves. All it takes is the *aggregate* CDF, or at least - if only a service curve is known - a lower bound on that CDF, as per eq. (1.1). The operations required for computing the CDF of a tagged flow at a node are:

- *FIFO multiplexing* of several CAFs at the entrance of a node, so as to compute the aggregate CAF;
- *Input-output transformation* from the aggregate CAF to the aggregate CDF, according to a node's service curve (i.e., to eq. (1.1));
- *FIFO de-multiplexing* of flows at the exit of a node, i.e., computation of per-flow CDFs from the aggregate CDF, exploiting the FIFO hypothesis.

The procedure is exemplified in Figure 1.17, using two piecewise-linear CAFs, A_1 and A_2, and a rate-latency service curve β (this can obviously be generalized to any number and shape of CAFs and any service curve). FIFO multiplexing (bottom left) is a summation of CAFs: $A = A_1 + A_2$. I-O transformation (top) corresponds to computing an aggregate CDF which is wide-sense increasing and satisfies eq. (1.1), e.g., the one obtained by assuming that equality holds in eq. (1.1). FIFO de-multiplexing (bottom right) exploits the FIFO hypothesis: more specifically, for all

$t \in R_+$, there is a unique $\tau \leq t$ such that $A(\tau) \leq D(t) \leq A(\tau^+)$. Then, $D_1(t)$ and $D_2(t)$ must satisfy $A_i(\tau) \leq D_i(t) \leq A_i(\tau^+)$, $i \in \{1, 2\}$ (and $D_1(t) + D_2(t) = D(t)$). However, $D_1(t)$ and $D_2(t)$ may not be uniquely defined, when neither A_1 nor A_2 are continuous in τ. In that case, any wide-sense increasing D_1 and D_2 satisfying the above equalities are possible CDFs.

If A_2 is discontinuous in τ (e.g., $A_2(\tau^+) = A_2(\tau) + \sigma$), but A_1 is not, then on some non trivial interval D_1 is constant, while D_2 has the same slope as D. If in the interval $[t_1, t_2]$, D is affine with slope R, and on the *corresponding* interval $[\tau_1, \tau_2]$ (i.e., the interval when the bits that depart in $[t_1, t_2]$ arrive at the input), A_i is affine with slope ρ_i, $i \in \{1, 2\}$, then D_i is affine on the interval $[t_1, t_2]$ with slope $\frac{\rho_i}{\rho_1+\rho_2} R$.

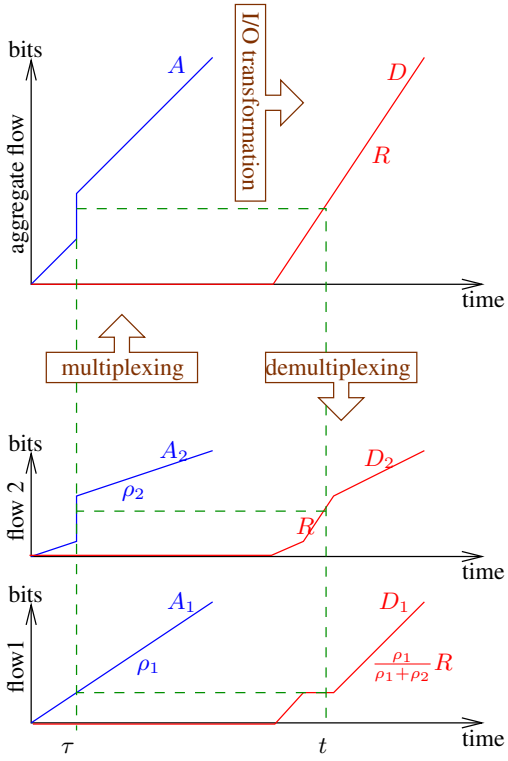

Figure 1.17 Input-output relationship at a FIFO node.

We now show how to compute the WCD in both blind- and FIFO-multiplexing tandems traversed by several flows.

1.4 Tandem systems traversed by several flows

1.4.1 Model

We analyze a tandem of n *nodes*, numbered from 1 to p, connected by forward links from node h to $h+1$, $1 \leq h < n$. The tandem is traversed by p *flows*, i.e., distinguishable streams of traffic. For each flow i, $1 \leq i \leq p$, we set $fst(i)$ the node at which this flow enters the network, and $lst(i)$ the node at which it departs. In other words, flow i traverses all nodes from $fst(i)$ to $lst(i)$ included, and then departs. Throughout the chapter, the exponent h corresponds to server h and the index i corresponds to a flow. We note $h \in i$ or $i \ni h$ if flow i traverses node h.

We make the following assumptions and will use the following notations in the rest of the chapter:

- $F_i^{(h)}$ is the CDF of flow i at node $h \in [i, j]$;
- $F_i^{(fst(i)-1)}$ represents the CAF of flow i at node $fst(i)$. In some cases, this CAF will also be denoted $F_i^{(0)}$;
- the aggregate CAF at node h is $A^{(h)} = \sum_{i \ni h} F_i^{(h-1)}$ and the aggregate CDF is $D^{(h)} = \sum_{i \ni h} F_i^{(h)}$;
- node h offers a service curve $\beta^{(h)}$ to the aggregate CAF at node h, $A^{(h)}$, and $\beta^{(h)}$ is assumed to be wide-sense increasing, piecewise affine and convex;
- the arrival process of flow i, $F_i^{(fst(i)-1)}$, is α_i-upper constrained, where α_i is assumed to be wide-sense increasing, piecewise affine and concave.

A system is said to be stable if there exists a constant C such that for each server, the backlog is always upper bounded by C. Let $R_h = \lim_{t\to\infty} \beta^{(h)}(t)/t$ and $\rho_i = \lim_{t\to\infty} \alpha_i(t)/t$. We assume that the system is stable, that is, $\forall h \in [1, n]$, $R_h \geq \sum_{i \ni h} \rho_i$ (see [26] for example).

A *scenario* for an n-node tandem described as above is a family of functions $(F_i^{(h)})_{1 \leq i \leq p, h \in i}$ such that:

1. $\forall i, h$, $F_i^{(h)}$ is wide-sense increasing, left-continuous and $F_i^{(h)}(0) = 0$;
2. $\forall i \ni h$, $F_i^{(h-1)} \geq F_i^{(h)}$;
3. $\forall i$, $F_i^{(fst(i)-1)}$ is α_i-upper constrained;
4. $\forall h \in [1, n]$, $D^{(h)} \geq A^{(h)} \otimes \beta_h$ if β_h are simple service curve; conversely, $\forall h \in [1, n]$, $\forall s < t$ in the same backlogged period, $D^{(h)}(t) - D^{(h)}(s) \geq \beta^{(h)}(t - s)$ if β_h are strict service curves.

1.4.2 Loss of the tightness

To illustrate the complexity of getting good bounds, consider the following simple example with two servers and two flows:

- for $h \in \{1, 2\}$, $\beta^{(h)} = \beta_{R_h, T_h}$,
- for $i \in \{1, 2\}$, $\alpha_i = \gamma_{\sigma_i, \rho_i}$, $fst(1) = fst(2) = 1$ and $lst(1) = lst(2) = 2$ and flow 1 is given a higher priority than flow 2.

To compute an upper bound on the delay of flow 2, at least two methods can be used:

(a) first compute the equivalent service curves for flow 2 and compute the convolution of the two curves thus obtained: $\tilde{\beta}_a = [\beta^{(1)} - \alpha_1]_+ \otimes [\beta^{(2)} - (\alpha_1 \oslash \beta^{(1)})]_+$.

(b) first compute the convolution of the two service curves and then compute the equivalent service for flow two. Note that even though the service is not strict anymore, a direct computation can be used to compute the residual service curve, as in [9, 34, 35]. We then find $\tilde{\beta}_b = [\beta^{(1)} \otimes \beta^{(2)} - \alpha_1]_+$.

The first approach is called *separated-flow analysis* (SFA). The second one, instead, uses the principle of *Pay Multiplexing Only Once* (PMOO), introduced in [35]. In the latter, the service impairment due to flow 1 is only counted once in the equivalent service curve, instead of twice as in per-flow analysis. Intuitively, PMOO can thus be expected to lead to smaller delay bounds. In fact, rather counter-intuitively, this is not always the case. After a few straightforward algebraic manipulations, we obtain:

$$\tilde{\beta}_a(t) = (\min(R_1, R_2) - \rho_1)[t - \frac{\sigma_1 + R_1 T_1}{R_1 - \rho_1} - \frac{\sigma_1 + \rho_1 T_1 + R_2 T_2}{R_2 - \rho_1}]_+$$

and

$$\tilde{\beta}_b(t) = (\min(R_1, R_2) - \rho_1)[t - \frac{\sigma_1 + \min(R_1, R_2)(T_1 + T_2)}{\min(R_1, R_2) - \rho_1}]_+.$$

The alert reader can check that, if $\beta_1 = \beta_2$, then $\tilde{\beta}_b \leq \tilde{\beta}_a$. But if $\sigma_1 = 0$, $T_1 = 0$ and $R_2 > R_1$, then $\tilde{\beta}_a \leq \tilde{\beta}_b$. This implies that neither approach is guaranteed to yield *tight* delay bounds in all settings. Moreover, the SFA method, used to compute $\tilde{\beta}_a$, can be generalized to generic feed-forward networks. This is the object of Section 1.4.3. The PMOO method, leading to $\tilde{\beta}_b$, cannot be generalized in such a simple way when several flows interfere with each other. In fact, when flows are not *nested* into one another you cannot convolve service curves (recall than you can only do that when they are traversed by the same set of flows). A different approach, which can also be applied to any feed-forward topology *and* leads to tight results, is to use mathematical programming. In the latter, you first work with the trajectories, and bound using the arrival and service curves only at the last step. This will be discussed in Section 1.5.

1.4.3 Separated-flow analysis

Algorithm 1.1 gives the general way of computing the equivalent service curve for a tagged flow traversing a tandem. It first computes an arrival curve for each flow at each intermediate server: $\alpha_i^{(h)}$ is an arrival curve for $F_i^{(h)}$. Then, it computes the equivalent service curve for each flow at each server: $\beta_i^{(h)}$ is the equivalent service curve of server h for flow i. Finally it computes the end-to-end service curve for the each flow, and a bound on its WCD can be computed using that curve.

Algorithm 1.1

General Separated-flow Analysis Algorithm {

for $h = 1$ to n
{
for each i such that $h \in i$
{
$\beta_i^{(h)} \leftarrow (\beta^{(h)} - \sum_{j \ni h - \{i\}} \alpha_j^{(h-1)})_+$;
$\alpha_i^{(h)} \leftarrow \alpha_i^{(h-1)} \oslash \beta_i^{(h)}$;
}
}
for $i = 1$ to m do
{
$\tilde{\beta}_i = *_{h \in i} \beta_i^{(h)}$
}

Algorithm 1.1 is valid for blind multiplexing (and thus for any service policy) if the servers offer strict service curves. If the service policy is known to be FIFO (in which case the service curves need not be strict), it is also possible to compute $\beta_i^{(h)}$ with the formula of Theorem 1.2 (instantiated for any nonnegative value of τ).

1.5 Mathematical programming approach

In this section, we present a method for computing exact worst-case performance in tandem networks. The method can be generalized to arbitrary acyclic networks, but for sake of notational and algorithmic simplicity, we detail the method only for tandem networks, and give the idea and the additional difficulties that arise in the general case at the end.

1.5.1 Blind multiplexing

Consider a single server traversed by one flow. To compute the WCD at time t, we use the following ingredients:

- strict service curve: $F^{(1)}(t) - F^{(1)}(s) \geq \beta^{(1)}(t-s)$;
- choose $s = start(t)$: $F^{(1)}(s) = F^{(0)}(s)$;

- introduce the arrival date, u, of the bit that departs at time t: $s \leq u \leq t$ and $F^{(0)}(u) \geq F^{(1)}(t)$;
- arrival curve: $F^{(0)}(u) - F^{(0)}(s) \leq \alpha_1(u-s)$.

As we assumed α_1 piecewise affine concave (i.e., the minimum of a finite number of affine curves) and $\beta^{(1)}$ piecewise affine convex (i.e., the maximum of a finite number of affine curves), all those constraints are either linear or easily exploded into a finite number of linear ones. To compute the WCD, the only remaining step is to maximize $t-u$ under these constraints. Therefore, this is a *linear program.*

In a tandem network of n nodes, the WCD can be computed by generalizing the above linear program *backwards*, i.e., starting from server n and going back to server 1.

1.5.1.1 The linear program

Variables Let us first define the variables of the linear program. Note that, to emphasize the meaning of the variables, we denote them the same way as the date of the function value they represent. Thus, they will mainly be named t_k or $F_i^{(h)}(t_k)$.

- *time variables:* we have $n+2$ relevant time instants, i.e., $t_0, \ldots, t_n$ and u, with the following interpretation: consider a bit of data that exists the system at time t_n: then t_{n-1} is the start of the backlogged period of server n at time t_n ($t_{n-1} = start_n(t_n)$) and more generally, $t_{i-1} = start_i(t_i)$. Variable u represents the arrival date of the bit of data considered;
- *functional variables:* the relevant variables are $F_i^{(fst(i)-1)}(t_k)$ and $F_i^{(h)}(t_k)$ for $i \in h$ and $k \in \{h, h-1\}$. Intuitively, $F_i^{(h)}(t_k)$ represents the value of the CAF $F_i^{(h)}$ at time t_k. The important dates for $F_i^{(h)}$ are t_h, i.e., the date at which the bit of interest exits server h and t_{h-1}, i.e., the start of the backlogged period of server h. Variable $F_i^{(fst(i)-1)}(t_k)$ represents the CAF of flow i. The variable $F_i^{(fst(i)-1)}(u)$ will also be used to compute the WCD of flow i.

Linear constraints Without loss of generality, we can assume that $lst(1) = n$, i.e., flow 1 traverses all the tandem and is the one whose WCD we want to compute. We have the following constraints:

- *time constraints* $\forall h, t_{h-1} \leq t_h$;
- *service constraints* $\forall h, \sum_{i \ni h} F_i^{(h)}(t_h) \geq F_i^{(h)}(t_{h-1}) + \beta^{(h)}(t_h - t_{h-1})$;
- *start of backlogged period constraints* $\forall h, \forall i \ni h, F_i^{(h)}(t_{h-1}) = F_i^{(h-1)}(t_{h-1})$;
- *causality constraints* $\forall h$, $\forall i \ni h$ and $k \in \{h, h-1\}$, $F_i^{(fst(i)-1)}(t_k) \geq F_i^{(h-1)}(t_k) \geq F_i^{(h)}(t_k)$;

- *non-decreasing constraints* $\forall i$, $\forall h \ni i$, $F_i^{(fst(i)-1)}(t_h) \geq F_i^{(fst(i)-1)}(t_{h-1})$ and $F_i^{(h)}(t_h) \geq F_i^{(h)}(t_{h-1})$;
- *arrival constraints* $\forall i, \forall k < h \ni i$,$F_i^{(fst(i)-1)}(t_h) - F_i^{(fst(i)-1)}(t_k) \leq \alpha_i(t_h - t_k)$;
- *constraints on* u: $t_0 \leq u \leq t_n$; $F_1^{(fst(1)-1)}(u) - F_1^{(fst(1)-1)}(t_{fst(1)}) \leq \alpha_1(u - t_{fst(1)})$ and $F_1^{(fst(1)-1)}(u) \geq F_1^{(n)}(t_n)$

Objective function: In order to compute the WCD of flow 1, we just need to maximize the distance between the time at which it exits node n and the time at which it enters the tandem, i.e.:

$$\textbf{Maximize } t_n - u.$$

The same variables and constraints, can also be used to compute the maximum backlog at node n. In this case, the objective is:

$$\textbf{Maximize } \sum_{i \ni n} F_i^{(fst(i)-1)}(t_n) - \sum_{i \ni n} F_i^{(n)}(t_n).$$

Note that time variable u and the related functional variables and constraints are not required in this case.

Let us denote with Λ the linear program defined above and let d_Λ (resp. b_Λ) its optimal solution, if the objective is the WCD (resp. the maximum backlog). The following theorem holds:

Theorem 1.3 *Consider a tandem network with n servers and p flows. The LP instance Λ has $\mathcal{O}(pn)$ variables and $\mathcal{O}(pn^2)$ constraints and is such that the optimum is the worst end-to-end delay for flow 1 is d_Λ (resp. the maximum backlog at server n is b_Λ).*

The above theorem is formally proved in [10]. Here, we just explain the principle of the proof using an example. It is based on the construction of an admissible scenario that satisfies the solution of the LP.

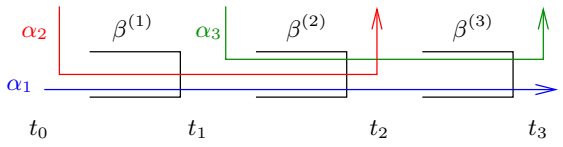

Figure 1.18 Tandem network with three nodes and three flows.

1.5.1.2 Example Consider the tandem of Figure 1.18. The latter has been extensively studied in [34], where it is shown that the exact WCD cannot be computed using the application of (min,plus) algebraic methods. To compute the WCD of flow 1, we have the following linear program:

- *time constraints*
  - $t_3 \geq t_2 \geq t_1 \geq t_0$;
  - $t_3 \geq u \geq t_0$;
- *service constraints*
  - $F_1^{(3)}(t_3) + F_3^{(3)}(t_3) \geq F_1^{(3)}(t_2) + F_3^{(3)}(t_2) + \beta^{(3)}(t_3 - t_2)$;
  - $F_1^{(2)}(t_2)+F_2^{(2)}(t_2)+F_3^{(2)}(t_2) \geq F_1^{(2)}(t_1)+F_2^{(2)}(t_1)+F_3^{(2)}(t_1)+\beta^{(2)}(t_2-t_1)$;
  - $F_1^{(1)}(t_1) + F_2^{(1)}(t_1) \geq F_1^{(1)}(t_0) + F_2^{(1)}(t_0) + \beta^{(3)}(t_1 - t_0)$;
- *start of backlogged period constraints*
  - $F_i^{(3)}(t_2) = F_i^{(2)}(t_2), i \in \{1, 3\}$;
  - $F_i^{(2)}(t_1) = F_i^{(1)}(t_1), i \in \{1, 2, 3\}$;
  - $F_i^{(1)}(t_0) = F_i^{(0)}(t_0), i \in \{1, 2\}$;
- *arrival constraints*
  - $F_1^{(0)}(t_k)-F_1^{(0)}(t_\ell) \leq \alpha_1(t_k-t_\ell), (k, \ell) \in \{(1, 0), (2, 1), (2, 0), (3, 2), (3, 1), (3, 0)\}$;
  - $F_1^{(0)}(u) - F_1^{(0)}(t_0) \leq \alpha_1(u - t_0)$;
  - $F_2^{(0)}(t_k) - F_2^{(0)}(t_\ell) \leq \alpha_2(t_k - t_\ell), (k, \ell) \in \{(1, 0), (2, 1), (2, 0)\}$;
  - $F_3^{(1)}(t_k) - F_3^{(1)}(t_\ell) \leq \alpha_3(t_k - t_\ell), (k, \ell) \in \{(2, 1), (3, 2), (3, 1)\}$;
- *causality constraints*
  - $F_1^{(0)}(t_k) \geq F_1^{(1)}(t_k), k \in \{1, 2, 3\}$;
  - $F_2^{(0)}(t_k) \geq F_2^{(1)}(t_k), k \in \{1, 2\}$;
  - $F_3^{(1)}(t_k) \geq F_3^{(2)}(t_k), k \in \{2, 3\}$;
- *non-decreasing constraints*
  - $F_1^{(0)}(t_0) \leq F_1^{(0)}(t_1) \leq F_1^{(0)}(t_2) \leq F_1^{(0)}(t_3)$; $F_1^{(0)}(t_0) \leq F_1^{(0)}(u)$; $F_1^{(3)}(t_2) \leq F_1^{(3)}(t_3)$; $F_1^{(2)}(t_1) \leq F_1^{(2)}(t_2)$; $F_1^{(1)}(t_0) \leq F_1^{(1)}(t_1)$;
  - $F_2^{(0)}(t_0) \leq F_2^{(0)}(t_1) \leq F_2^{(0)}(t_2)$; $F_2^{(2)}(t_1) \leq F_2^{(2)}(t_2)$; $F_2^{(1)}(t_0) \leq F_2^{(1)}(t_1)$;
  - $F_3^{(1)}(t_1) \leq F_3^{(1)}(t_2) \leq F_3^{(1)}(t_3)$; $F_3^{(3)}(t_2) \leq F_3^{(3)}(t_3)$; $F_3^{(2)}(t_1) \leq F_3^{(2)}(t_2)$;
- *delay at t_3*
  - $F^{(3)}(t_3) \leq F_1^{(0)}(u)$;
- *Objective*

– *Maximize* $t_3 - u$.

Figure 1.19 explains the relation between the linear program and the trajectories. Given an admissible trajectory, the assignment variables can be read from the figure: let t_3 be the exit time of the bit whose the delay is computed. Variables t_i are defined as the beginning of the backlogged period at each server. Then, variables $F_i^{(h)}(t_k)$ are assigned the value of the CAF $F_i^{(h)}$ at time t_k. Those variables satisfy of course all the constraints, that are derived from Network Calculus basics. Conversely, to construct the trajectories from a solution of the linear program, it is enough to linearly interpolate the CAFs: $F_i^{(fst(i)-1)}$ between $t_{fst(i)-1}$ and $t_{lst(i)}$; and $F_i^{(h)}$ between t_{h-1} and t_h. Before t_{h-1}, one can set $F_i^{(h)} = F_i^{(h-1)}$ and after t_h, $F_i^{(h)} = F_i^{(fst(i)-1)}$.

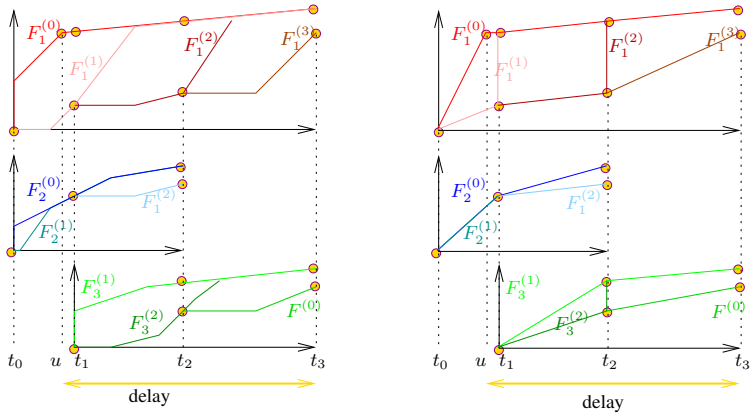

Figure 1.19 From the trajectories to the linear program and back to the trajectories for blind multiplexing.

1.5.1.3 Equivalent service curve The above method allows one to compute the WCD for a tagged flow, under constraints $(\alpha_i)_{i\in F}$ and $(\beta_j)_{j\in S}$. One may also want to measure how the network affects flow 1, in particular whether the flow can be guaranteed some end-to-end service curve. We call a *universal* end-to-end service curve a service curve β which is independent of α_1 (i.e., β remains an end-to-end service curve for any α_1). Precomputing such a universal curve can be useful to quickly compute a bound on the end-to-end delay for flow 1 for several different curves α_1, instead of having to write down and solve a linear program every time.

For tandem networks, it is possible to compute such a universal end-to-end service curve, which is *optimal* in the sense that it is maximal for all the universal service curves. It can be computed as follows: compute the WCD for the network when α_1 is a constant function equal to σ, call it $d(\sigma)$. Then, β_1, the residual service curve for flow 1 is the inverse function d^{-1}.

The LP that we have defined above to compute $d(\sigma)$ can be schematically rewritten as

$$\text{Maximize } AX$$
$$\text{Subject to } BX \leq C(\sigma),\ X \geq 0,$$

where only C depends on σ. Indeed, σ never appears within coefficient matrix B. Then, by the strong duality theorem of linear programming, the following LP has the same optimal solution.

$$\text{Minimize } C^t(\sigma)Y$$
$$\text{Subject to } B^tY \leq A^t,\ Y \geq 0.$$

The constraints of the dual LP are independent of σ, and for any σ, its optimal value is reached at a vertex of the convex polyhedron defined by the constraints $B^tY \leq A^t$ and $Y \geq 0$. Hence, function $d(\sigma)$ is the minimum of a finite set of linear functions (whose cardinality can be exponential in the size of the network).

1.5.1.4 Generalization to feed-forward networks The same method can be applied to general feed-forward networks. In this case, several difficulties arise, making the problem much more difficult. More specifically:

- **The number of relevant dates grows exponentially with the size of the network.** Indeed, starting from the date that marks the beginning of the backlogged period of one given server, several dates must be defined for the beginning of the backlogged period of all the predecessors of this servers. As a consequence, the number of required dates is the number of paths from any node to the last server visited by the tagged flow.
- **Those dates are not totally ordered** and every order compatible with the NC constraints must be generated, leading to a different linear program for each order. Thus, the WCD is the maximum solution of an exponential number of linear programs.

To illustrate this fact, consider the example of Figure 1.20. The departure date of the bit of interest is $t_\emptyset$, and we can define $t_4 = start_4(t_\emptyset)$, $t_{24} = start_2(t_4)$ and $t_{34} = start_3(t_4)$. But then, for server 1, two starts of backlogged period have to be defined: t_{134} and t_{124}. Four orders have to be consider: either t_{24} and t_{34} belong to different backlogged period, in which case we have $t_{124} \leq t_{24} \leq t_{134} \leq t_{34}$ or $t_{134} \leq t_{34} \leq t_{124} \leq t_{24}$, or they belong to the same backlogged period, in which case we have $t_{134} = t_{124} \leq t_{34} \leq t_{24}$ or $t_{134} = t_{124} \leq t_{24} \leq t_{34}$. Date u also has to be inserted in these orders, inducing even more linear programs. The other constraints depending of the order of the dates (arrival, non-decreasing) have to be generated according to these orders.

A reduction of X3C (Exact-three-cover, [23]) to this problem shows that it is in fact NP-hard to compute the WCD under those assumptions.

1.5.1.5 Generalization to other service policies Assuming *blind multiplexing*, or *arbitrary multiplexing* may yield very pessimistic bounds for some networks where the service policy is known. It may not always be possible to find a linear

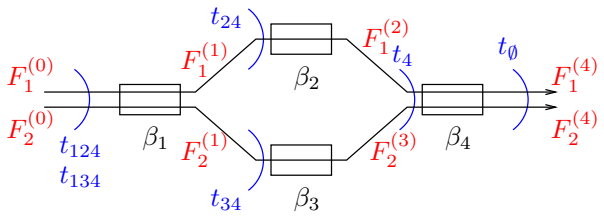

Figure 1.20 Example of a feed-forward network.

program that computes the exact worst-case performance efficiently, even for tandem networks. For example, an attempt to find a linear program encoding Static Priorities can be found in [11]. In these cases, a good strategy is to mix Algorithm 1.1 with the linear programming approach used for blind multiplexing. This can be done as follows:

1. Generate the LP assuming blind multiplexing;
2. Using Algorithm 1.1, compute the intermediate arrival curves for each flow at each node it traverses;
3. Add linear constraints corresponding to these intermediate arrival curves to the LP;
4. Compute the optimal solution of the LP thus modified.

1.5.2 FIFO multiplexing

Under FIFO multiplexing it is also possible to compute tight bounds using mathematical programming. The fundamental modeling difference with respect to the blind multiplexing case are:

- exploit the *FIFO hypothesis* to infer input-output relationships;
- consider simple service curves instead of strict service curves.

Consider the trivial example of a single server traversed by two flows. The above two elements induce the following linear constraints: if t_1 is the departure date of the bit of interest,

- *(FIFO hypothesis)* there exists $t_2 \in [t_3, t_1]$ such that $F_1^{(0)}(t_2) = F_1^{(1)}(t_1)$ and $F_2^{(0)}(t_2) = F_2^{(1)}(t_1)$, and
- *(simple service curve)* there exists $t_3 \leq t_1$ such that we have $F_1^{(1)}(t_1)+F_2^{(1)}(t_1) \geq (F_1^{(0)} + F_2^{(0)}) \otimes \beta(t_1) = F_1^{(0)}(t_3) + F_2^{(0)}(t_3) + \beta^{(1)}(t_1 - t_3)$,

Furthermore, the monotonicity, causality and arrival curves constraints are still valid. Note that for one date (t_1), *two* new dates have been introduced, t_2 and t_3.

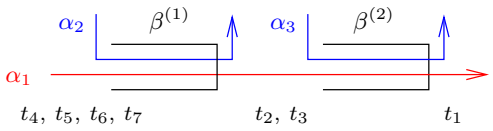

Figure 1.21 Example of a network with two servers and three flows.

This hints at the fact that the number of dates will double at each server, and we will also face the problem of ordering those variables.

Consider the example of two servers and three flows in Figure 1.21. Define t_1, t_2 and t_3 as in the previous example. We write $t_2 = FIFO(t_1)$ and $t_3 = SC(t_1)$ for notational convenience. For server 1, we can define $t_4 = FIFO(t_2)$, $t_5 = SC(t_2)$, $t_6 = FIFO(t_3)$ and $t_7 = SC(t_3)$. We know that $t_3 \leq t_2 \leq t_1$, that $t_5 \leq t_4 \leq t_2$, that $t_7 \leq t_6 \leq t_3$ and that $t_7 \leq t_5$ and $t_6 \leq t_4$. However, nothing can be deduced for the order of t_5 and t_6. This order is necessary to get tight bounds, because we need to ensure the monotonicity of the functions: if $t_5 \geq t_6$, we must also enforce $F_i^{(1)}(t_5) \geq F_i^{(1)}(t_6)$, $i \in \{1, 2\}$. This order can be set by the linear program using binary variables: consider a sufficiently large positive constant Q (in our case, this constant can easily be computed and some LP solvers can also compute it themselves), and let b be a binary variable. Consider the following constraints:

$$\begin{aligned} t_5 + bQ &\geq t_6 \\ t_6 + (1-b)Q &\geq t_5. \end{aligned}$$

If b = 1, then we have $t_5 + Q \geq t_6$, which is always verified for Q large enough, and $t_6 \leq t_5$; if $b = 0$, we have $t_5 \leq t_6$ and $t_6 + Q \geq t_5$, which is always true. This way, variable b determines the order of the two dates. Variable b can then be used to enforce the same order for the functional variables at these two dates, using similar constraints, so that monotonicity is preserved. Note that, due to the presence of binary variables, the model becomes a Mixed Integer Linear Program (MILP).

1.5.2.1 The mixed integer-linear program

Variables The variables of our problem are the following:

- *time variables:* $t_1, \ldots, t_{2^{n+1}-1}$, where t_{2k} and t_{2k+1} correspond to the FIFO hypothesis and the service curve constraints with regards to t_k, respectively: $t_{2k} = FIFO(t_k)$ and $t_{2k+1} = SC(t_k)$;
- *functional variables:* $F_i^{(h)}(t_k)$ for $h \in [fst(i) - 1, lst(i)]$ and for $k \in [2^{n+1-h}, 2^{n+2-h} - 1]$.

Linear constraints As explained before, the number of dates (hence of variables) grows exponentially with the tandem length, since it doubles at each node as we

go backwards. Moreover, in a multi-node scenario, these dates are only partially ordered. For $k < 2^n$, we have (*) $t_{2k+1} \leq t_{2k} \leq t_k$. Moreover, if $2^h \leq k, k' < 2^{h+1}$ and (**) if $t_k \leq t_{k'}$, then $t_{2k} \leq t_{2k'}$ (cumulative functions are non-decreasing) and $t_{2k+1} \leq t_{2k'+1}$ (same as above, plus β_j is convex). These relations and the transitivity only lead to a partial order of $t_{2^h}, \ldots, t_{2^{h+1}-1}$.

We must order the dates t_k, t'_k if there exists h such that $2^h \leq k, k' < 2^{h+1}$ by introducing binary variables. We do this recursively backwards as follows:

- For server n, we have $t_2 \geq t_3$:
- Suppose that dates are ordered for all $k \in [2^{h-1}, 2^h - 1]$ and now consider dates t_k, $k \in [2^h, 2^{h+1} - 1]$.
  - First generate the partial *known* order using (*) and (**);
  - For $k = 2\ell$ and $k' = 2\ell'$ or $k = 2\ell + 1$ and $k' = 2\ell' + 1$, use the same binary variable used to order t_ℓ and $t_{\ell'}$;
  - For any other pair, introduce a new binary variable.

If variables x and y are ordered by the variable b, we note $x \leq_b y$ to represent the constraints

$$\begin{aligned} x + bQ &\geq y \\ y + (1-b)Q &\geq x. \end{aligned}$$

Moreover, if no binary variable is needed, we write $x \leq_\emptyset y$ to represent the constraint $x \leq y$.

We can now write the linear constraints:

- *time and monotonicity constraints:* for any $t_k \leq_b t_{k'}$, $t_k \leq_b t_{k'}$, $F_i^{(h)}(t_k) \leq_b F_i^{(h)}(t_{k'})$ are constraints;
- *FIFO hypothesis:* if $fst(i) < h$, $F_i^{(h)}(t_k) = F_i^{(h-1)}(t_{2k})$ is a constraint;
- *service constraints:* if $k \in [2^{n-h}, 2^{n+1-h} - 1]$, $D^{(h)}(t_k) \geq A^{(h-1)}(t_{2k+1}) + \beta_h(t_k - t_{2k-1})$ is a constraint;
- *arrival constraints:* if $h = fst(i)$ and $t_k \leq_\emptyset t_{k'}$, $F_i^{(h)}(t'_k) - F_i^{(h)}(t_k) \leq \alpha_i(t_{k'} - t_k)$ is a constraint; if $h = fst(i)$ and $\exists b \neq \emptyset, t_k \leq_b t_{k'}$ then $F_i^{(h)}(t_{k'}) - F_i^{(h)}(t_k) \leq \alpha_i(t_{k'} - t_k) + (1-b) \cdot Q$ and $F_i^{(h)}(t_k) - F_i^{(h)}(t_{k'}) \leq \alpha_i(t_k - t_{k'}) + b \cdot Q$ are constraints.

Objective function If the objective is to compute the WCD, then the objective of the MILP is $\max t_1 - t_{2^n}$.

As for the blind multiplexing case, we can exploit the same model to compute the maximum backlog at server n too. In order to do this, one has to introduce new variables $F_i^{(fst(i)-1}(t_1)$ for $i \in n$ and the constraints

$$F_i^{(fst(i)-1}(t_1) - F_i^{(fst(i)-1}(t_{2^{n-fst(i)+1}}) \leq \alpha_i(t_1 - t_{2^{n-fst(i)+1}});$$

then the objective of the MILP is $\max \sum_{i \in n} F_i^{(fst(i)-1)}(t_1) - \sum_{i \in n} F_i^{(n)}(t_1)$.

Let us denote with Λ the MILP defined above and d_Λ (resp. b_Λ) its optimal solution if the objective is the WCD (resp. the maximum backlog at server n). The following theorem holds.

Theorem 1.4 *For a tandem network with n servers and p flows. The MILP instance Λ is such that the optimum is the WCD for flow 1 is d_Λ (resp. the maximum backlog at server n is b_Λ).*

This can be generalized with no difficulty to any flow. The proof of this theorem can be found in [12]. Here, we illustrate the concept using the example of Figure 1.20.

1.5.2.2 Example The MILP for computing the WCD of the tandem of of Figure 1.20 is:

Maximize $t_1 - t_4$ such that:

- *time constraints*:
  - $t_k \geq t_{2k} \geq t_{2k+1}, k \in \{1, 2, 3\}$;
  - $t_4 \geq t_6$;
  - $t_5 + (1 - b) \cdot Q \geq t_6$ and $t_6 + b \cdot Q \geq t_5$;
- *service constraints:*
  - $F_1^{(2)}(t_1) + F_3^{(2)}(t_1) \geq F_1^{(1)}(t_3) + F_3^{(1)}(t_3) + \beta^2(t_1 - t_3)$;
  - $F_1^{(1)}(t_k) + F_2^{(1)}(t_k) \geq F_1^{(0)}(t_\ell) + F_2^{(0)}(t_\ell) + \beta^1(t_k - t_\ell), (k, \ell) \in \{(2, 5), (3, 7)\}$;
- *FIFO constraints:*
  - $F_i^{(2)}(t_1) = F_i^{(1)}(t_2), i \in \{1, 3\}$;
  - $F_i^{(1)}(t_k) = F_i^{(0)}(t_{2k}), i \in \{1, 2\},\ k \in \{2, 3\}$;
- *monotonicity constraints:*
  - $F_i^{(1)}(t_3) \leq F_i^{(1)}(t_2), i \in \{1, 3\}$;
  - $F_i^{(0)}(t_k) \geq F_i^{(0)}(t_\ell), i \in \{1, 2\},\ (k, \ell) \in \{(4, 5), (4, 6), (4, 7), (5, 7), (6, 7)\}$;
  - $F_i^{(0)}(t_5) + (1 - b) \cdot Q \geq F_i^{(0)}(t_6), i \in \{1, 2\}$;
  - $F_i^{(0)}(t_5) \leq b \cdot Q + F_i^{(0)}(t_6), p \in \{1, 2\}$;

- *arrival constraints:*
  - $F_3^{(1)}(t_2) - F_3^{(1)}(t_3) \leq \alpha_3(t_2 - t_3)$;
  - $F_i^{(0)}(t_k) - F_i^{(0)}(t_\ell) \leq \alpha_i(t_k - t_\ell), i \in \{1,2\},\ (k,\ell) \in \{(4,5),(4,6),(4,7),(5,7),(6,7)\}$;
  - $F_i^{(0)}(t_5) - F_i^{(0)}(t_6) \leq \alpha_i(t_5 - t_6) + (1-b)\cdot Q, i \in \{1,2\}$;
  - $F_i^{(0)}(t_6) - F_i^{(0)}(t_5) \leq \alpha_i(t_6 - t_5) + b \cdot Q, i \in \{1,2\}$.

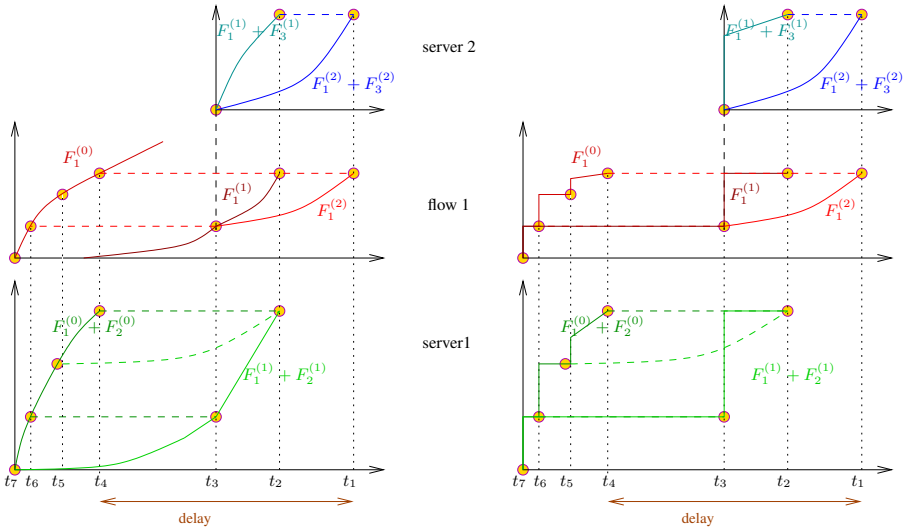

Figure 1.22 From the trajectory to the MILP and back to the trajectories for the network of Figure 1.20. Left: from the trajectory to the MILP: the circles represent the values associates to the variables of the form $F_i^{(h)}(t_k)$. Right: from those variables, it is possible to draw a new trajectory satisfying the constraints where the arrival are maximized according to their constraints, staircase shaped for the intermediate flows, and follows the service curve for the last server.

1.6 Related Work

The study of tandem networks under blind multiplexing has already been addressed in [34]. The authors compute tight end-to-end delay bounds for some tandem networks, pointing out the difficulties we mentioned earlier in this chapter. They detail their computations for a network with three servers and three flows and a more general approach is suggested in the corresponding technical report [33]. The main difference between this approach and the linear programming one is that they directly

compute equivalent service curves, with some free parameters to optimize. Moreover, this approach is basically intended for leaky-bucket/rate-latency arrival/service curves and can be generalized only in a very inefficient way.

Another approach has been developed in [7, 8] and can deal with concave/convex arrival/service curves, but loses the tightness of the bounds. The idea is to define a multi-dimensional convolution to directly compute an equivalent service curve for the flow of interest. This would correspond to the computation of $\tilde{\beta}_b$ in Section 1.4.2. Besides the fact that tight bounds may not be computed, this convolution might be algorithmically complex to compute.

As far as FIFO networks are concerned, the first paper to determine some bounds on the delay has been [20], which shows that, for a *generic* FIFO network (i.e., not necessarily feed-forward), upper bounds on the WCD can only be computed for small utilization factors. A *critical utilization factor* ν is defined, which is inversely proportional to the maximum path length. The paper shows two fundamental limits:

- For a utilization $u \leq \nu$ the bound is proportional to $1/(\nu - u)$, hence approaches infinity as the utilization approaches ν.
- For any utilization $u > \nu$ and finite delay d, it is possible to construct a (non feed-forward) network where some traffic exhibits a delay larger than d.

The above finding can be interpreted as implying that better results can only be obtained by adding some more hypothesis. For instance, the *feed-forward* hypothesis is relevant in practice and hardly constraining at all. A tandem network is in fact a *feed-forward* network, and the latter are known to be stable for any utilization up to 100%. This means that we can find better bounds under that hypothesis than those computed using the method in [20]. Several papers dealing with computing delay bounds in FIFO tandem networks have appeared recently [29, 27, 30, 2, 3, 4, 13, 24]. All these papers rely on *equivalent service curves*, i.e., those computed by using Theorem 1.2. A method known as Least Upper Delay Bound (LUDB) is described in those papers. The method is based on removing the cross flows by iteratively applying Theorem 1.2. Depending on the paths of the cross flows, two situations may arise: in a so-called *"nested"* tandem, i.e., one where either any two flow paths are disjoint or one includes the other, it is possible to compute an equivalent end-to-end service curve for the tagged flow [30]. Otherwise, no end-to-end service curve can be computed: first, the tandem has to be *cut* into (possibly many) nested sub-tandems; then, bounds on the WCD of each sub-tandem must be computed and summed up to obtain a bound on the end-to-end WCD. In both cases, the delay bound (whether for the whole tandem or for sub-tandems) is computed by solving a piecewise-linear programming (P-LP) problem. A tool called DEBORAH has been devised to solve the problem [3]. It transforms the P-LP problem into a number of LPs, each one of which produces an upper bound on the WCD, solves all the LPs and takes the minimum solution (i.e., the least upper bound). The LUDB method, in general, does not compute the WCD. It does in sink-tree networks ([27]), where, besides, a faster algorithm not relying on LP can be used, and in some more special cases ([4]), but this is not always the case, even in simple nested tandems, as proved

in [2]. Moreover, authors of[4] argue that, in non-nested tandems, the LUDB may be grossly overrated, due to the fact that cutting the tandem entails assuming separate, non compatible worst-case scenarios at each sub-tandem, much as summing per-node WCDs would do in a per-flow scheduling tandem. The upside of the LUDB method is that it is relatively efficient from a computational standpoint. Computing the LUDB is still an exponential problem, but an optimized implementation that exploits some structure in the problem, described in [4], allows one to analyze up to 30-node tandems in minutes on off-the shelf hardware. For shorter tandems (e.g., up to 10 nodes), the LUDB can indeed be computed online, in split-second times. One of the downsides, instead, is that it can only work with single leaky-bucket arrival curves and rate-latency curves. Works [13, 24] discuss the advantages of including peak constraints (i.e., double-leaky-bucket arrival curves) in the model, but limit themselves to the tagged flow. Authors in [13] show that a double leaky bucket cannot be assumed for cross flows, since this makes Theorem 1.2 yield curves which are not always wide-sense increasing, hence cannot be assumed to be equivalent service curves.

With respect to these works, the linear-programming approach described in this chapter is:

- more general: it allows arbitrary piecewise-concave arrival curves, and arbitrary piecewise-convex service curves;
- capable of computing the WCD, whereas the other is not.

Obviously enough, the above advantages are paid for by a higher complexity: LP-based WCD computation is difficult to achieve in practice for more than 10 nodes in the current implementation of the solver. However, the latter is not optimized for speed: more specifically, there has been – as of today – no effort to obtain an optimized ILP solution strategy, relying instead on the prowess of a general-purpose solver such as CPLEX. We expect that the current speed figures can be improved considerably by both reformulating the model in a more efficient way and integrating an optimized solution strategy in our solver.

1.7 Numerical Results

In this section, we illustrate the effectiveness of the linear programming approach by comparing it to the classical approaches on a tandem scenario. Consider the scenario given in Figure 1.23. Every cross-flow traverses two servers (except those at the extremities, i.e., servers 1 and n, which only traverse one). The tagged flow traverses every server. Every server has the same characteristics: a latency of 0.1 s and a service rate of 10 Mbps. Flows have a maximum burst σ of 1 Mb and an arrival rate ρ of 0.67 Mbps.

Fig. 1.24 shows the delay obtained with the separated flow analysis approach (SFA) and the tight LP method assuming *blind* multiplexing. Unsurprisingly, the two methods give the same result when there is only one server. For a network

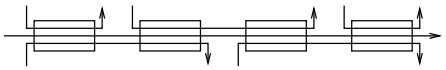

Figure 1.23 Tandem scenario with 4 servers.

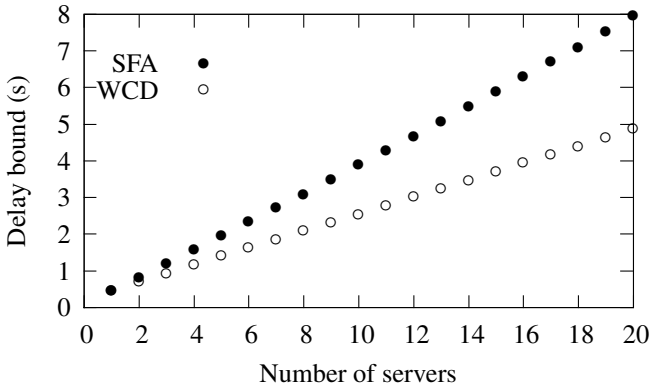

Figure 1.24 Upper bounds for the delay of the scenario of Fig. 1.23.

with 20 servers, the LP method reduces the SFA bound by a factor $8/5$, for a link utilization of 20%.

Fig. 1.25 depicts the distance between SFA and LP methods when the utilization of the servers varies, with 20 servers. The arrival rate of the flows varies so as to obtain the utilization shown in the horizontal axis. As the utilization grows, the gain becomes huge (note that the vertical scale is logarithmic).

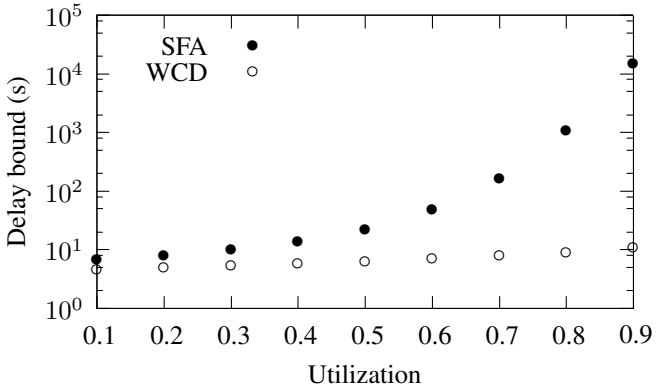

Figure 1.25 Upper bounds for the delay of the scenario of Fig. 1.23 for 20 servers and when the utilization varies.

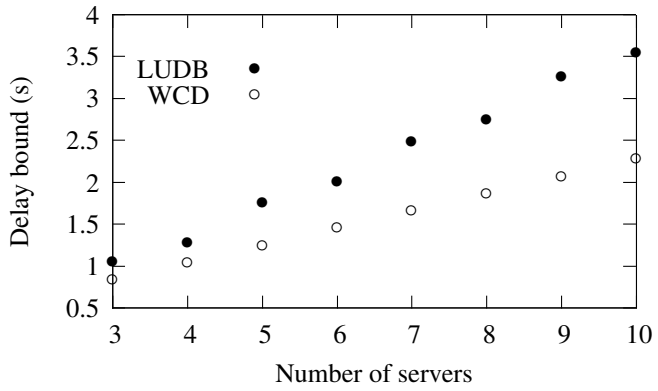

Figure 1.26 Upper bounds for the delay of the scenario of Fig. 1.23 (with FIFO multiplexing).

Assume now the same scenario in Figure 1.23, this time with *FIFO* multiplexing. We compare the linear-programming approach with the LUDB one ([4]).

Fig. 1.26 compares the LUDB and the linear-programming exact WCD as the number of servers increases. Note that computing the exact WCD takes longer with FIFO, due to the presence of binary variables, hence we stick to smaller-size tandems. The figure shows that the overrating of the LUDB method increases with the number of nodes. Note that the WCD increases linearly with the number of nodes.

Fig. 1.27 compares the LUDB and the linear-programming exact WCD then the number of servers is equal to eight. It is evident that the LUDB overrating increases with the utilization.

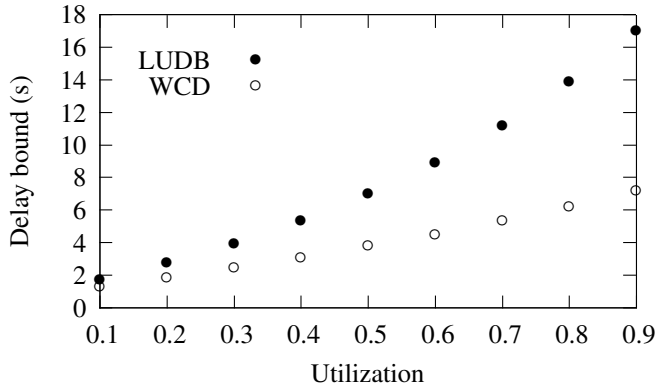

Figure 1.27 Upper bounds for the delay of the scenario of Fig. 1.23 for eight servers and when the utilization varies (with FIFO multiplexing).

1.8 Conclusions

In this chapter we have shown how Network Calculus can be used to compute worst-case delay bounds in tandem networks. Computations are easy when per-flow scheduling is in place: in that case, given a good model of the arrivals of a flow and the service at each node, we quickly obtain *tight* bounds on the delay by exploiting the *Pay Burst Only Once* principle, which clearly outperforms per-node analysis. When, instead, per-aggregate multiplexing is in place, computations get considerably more involved. We have shown that methods based on equivalent service curves do not lead to tight delay bounds, which can instead be obtained through a *linear-programming* approach. We have exemplified such an approach on two well-known paradigms: *blind* multiplexing, where the queueing policy is assumed to be arbitrary, and *FIFO* multiplexing. Our results show that, in both cases, the linear-programming approach outperforms the others.

REFERENCES

1. H. Bauer, J.-L. Scharbarg, and C. Fraboul. Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach. *IEEE Transactions on Industrial Informatics*, 6(4):521–523, 2010.
2. L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea. Estimating the worst-case delay in FIFO tandems using network calculus. In *proceedings of Valuetools'08*, 2008.
3. L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea. Deborah: A tool for worst-case analysis of FIFO tandems. In *proceedings of ISoLA'10, Special Track on Worst-case Traversal Time*, 2010.
4. L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea. Numerical analysis of worst-case end-to-end delay bounds in FIFO tandem networks. *Springer Real-Time Systems Journal*, 2012.
5. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services*. IETF, 1998.
6. A. Bouillard. Composition of service curves in network calculus. In *Proceedings of WCTT 2011*, pages 25–42, 2011.
7. A. Bouillard, B. Gaujal, S. Lagrange, and E. Thierry. Optimal routing for end-to-end guarantees using network calculus. *Performance Evaluation*, 65(11-12):883–906, 2008.
8. A. Bouillard, L. Jouhet, and E. Thierry. Computation of a (min,+) multi-dimensional convolution for end-to-end performance analysis. In *proceedings of Valuetools'08*, 2008.
9. A. Bouillard, L. Jouhet, and E. Thierry. Service curves in Network Calculus: dos and don'ts. Research Report RR-7094, INRIA, 2009.

10. A. Bouillard, L. Jouhet, and E. Thierry. Tight performance bounds in the worst-case analysis of feed-forward networks. In *proceedings of INFOCOM'10*, 2010.

11. A. Bouillard and A. Junier. Worst-case delay bounds with fixed priorities using network calculus. In *proceedings of Valuetools'11*, 2011.

12. A. Bouillard and G. Stea. Exact worst-case delay for FIFO-multiplexing tandems. In *proceedings of VALUETOOLS*, pages 158–167, 2012.

13. M. Boyer. Half-modeling of shaping in FIFO net with network calculus. In *proceedings of RTNS'12*, 2012.

14. M. Boyer, J. Migge, and N. Navet. An efficient and simple class of functions to model arrival curve of packetised flows. In *Proceedings of WCTT 2011*, pages 43–50, 2011.

15. R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview*. IETF, 1994.

16. P. Cappanera, L. Lenzini, A. Lori, G. Stea, and G. Vaglini. Efficient link scheduling for online admission control of real-time traffic in wireless mesh networks. *Elsevier Computer Communications*, 34(8):922–934, 2011.

17. P. Cappanera, L. Lenzini, A. Lori, G. Stea, and G. Vaglini. Optimal joint routing and link scheduling for real-time traffic in tdma wireless mesh networks. *Elsevier Computer Networks*, 57(11):2301–2312, 2013.

18. S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, 2003.

19. C. S. Chang. *Performance Guarantees in Communication Networks*. TNCS, Springer-Verlag, 2000.

20. A. Charny and J.-Y. L. Boudec. Delay bounds in a network with aggregate scheduling. In *proceedings of QoFIS*, pages 1–13, 2000.

21. R. L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

22. R. L. Cruz. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

23. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

24. F. Jafari, A. Jantsch, and Z. Lu. Worst-case delay analysis of variable bit-rate flows in network-on-chip with aggregate scheduling. In *proceedings of DATE*, pages 538–541, 2012.

25. A. Koubaa, M. Alves, and E. Tovar. Modeling and worst-case dimensioning of cluster-tree wireless sensor networks. In *proceedings of IEEE RTSSS'06*, pages 412–421, 2006.

26. J.-Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume LNCS 2050. Springer-Verlag, revised version 4, may 10, 2004 edition, 2001.

27. L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea. Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. *Performance Evaluation*, 63(9-10):956–987, 2006.

28. L. Lenzini, E. Mingozzi, and G. Stea. Tradeoffs between low complexity, low latency and fairness with deficit round robin schedulers. *IEEE/ACM Transactions on Networking*, 12(4):681–693, August 2004.

29. L. Lenzini, E. Mingozzi, and G. Stea. Delay bounds for FIFO aggregates: a case study. *Computer Communications*, 28(3):287–299, 2005.

30. L. Lenzini, E. Mingozzi, and G. Stea. A methodology for computing end-to-end delay bounds in FIFO-multiplexing tandems. *Performance Evaluation*, 65(11-12):922–943, 2008.

31. A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking (TON)*, 1, 1993.

32. J. B. Schmitt and U. Roedig. Sensor network calculus: A framework for worst case analysis. In *proceedings of 1st International Conference on Distributed Computing in Sensor Systems*, pages 141–154, 2005.

33. J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay bounds under arbitrary multiplexing. Technical report, University of Kaiserslautern, 2007.

34. J. B. Schmitt, F. A. Zdarsky, and M. Fidler. Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch ... In *proceedings of INFOCOM'08*, 2008.

35. J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving performance bounds in feed-forward networks by paying multiplexing only once. In *proceedings of MMB'08*, 2008.

36. M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *IEEE/ACM Transactions on Networking*, 4:375–385, June 1996.

37. T. Skeie, S. Johannessen, and O. Holmeide. Timeliness of real-time IP communication in switched industrial ethernet networks. *IEEE Transactions on Industrial Informatics*, 2:25–39, 2006.

38. D. Stiliadis and A. Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Transactions on Networking*, 6(5):611–624, October 1998.

39. Y. Xu, F. Ren, T. He, C. Lin, C. Chen, and S. Das. Real-time routing in wireless sensor networks: A potential field approach. *ACM Transactions on Sensor Networks*, 9(3), 2013.