A simple and efficient class of functions to model arrival curve of packetised flows* †

Marc Boyer  
ONERA – The French Aerospace Lab  
F 31055, Toulouse, France  
marc.boyer@onera.fr

Jörn Migge  
RealTime-at-Work (RTaW)  
54600 Villers-lès-Nancy, France  
jorn.migge@realtimeatwork.com

Nicolas Navet  
INRIA / RTaW  
54600 Villers-lès-Nancy, France  
nicolas.navet@inria.fr

ABSTRACT

Network Calculus is a generic theory conceived to compute upper bounds on network traversal times (WCTT – Worst Case Traversal Time). This theory models traffic constraints and service contracts with arrival curves and service curves. As usual in modelling, the more realistic the model is, the more accurate the results are, however a detailed model implies large running times which may not be the best option at each stage of the design cycle. Sometimes, a trade-off must be found between result accuracy and computation time. This paper proposes a combined use of two simple classes of curves in order to produce accurate results with a low computational complexity. Experiments are then conducted on a realistic AFDX case-study to benchmark the proposal against two existing approaches.

Categories and Subject Descriptors
C.4 [PERFORMANCE OF SYSTEMS]; Performance attributes; G.3 [Mathematics of Computing]; Miscellaneous—Queueing theory; C.3 [SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS]; Real-time and embedded systems

1. INTRODUCTION

Context of the problem. Aircrafts today embed hundreds of sensors, actuators and computers communicating through networks. AFDX (Avionics Full-Duplex Switched Ethernet) is a widely used network technology that equips most recent large civil airplanes. Thousands of data flows, called Virtual Links (VL) in AFDX terminology, are exchanged among the nodes. To ensure the correct temporal behaviour of the applications, that are often subject to stringent timing constraints, the time between the emission and reception of each VL packet, also called the Worst Case Traversal Time, must be upper bounded.

Definition of the problem. An AFDX network is an Ethernet-based technology, using full duplex links. So, there are no collisions between frames, and the only indeterminism (and main cause of delays) comes from the waiting time in the switch queues. Due to the size of the system (number of VLs, number of nodes and switches, redundant networks), computationally efficient algorithms must be used, especially early in the design cycle, when design space exploration may be used for optimizing topologies, streams routing and functions allocation. But to obtain computationally efficient algorithms, some conservative (i.e. pessimistic) approximations have to be done in the modelling and/or the bound computation.

Existing work. If network calculus has successfully been used to compute the WCTT of the AFDX network and certify the A380, the computed bounds are most often larger than the real worst-cases, and significant improvements are probably not out of reach (see, for instance for recent works). To compute accurate bounds, a key mechanism that should be modelled is serialisation, also called grouping or shaping depending on the authors: when two flows share the same link from one systems to another, they can not reach the destination at the same time. Modelling (or ignoring) this mechanism can modify the bounds up to 40% in a realistic AFDX configuration.  

Shortcomings of existing solutions. A source of pessimism in network calculus, as it has been most often used until yet, is the fluid modelling of flows. Indeed, because they have a lot of good properties ($O(n)$ complexity for the main algorithms, easy implementation, etc), concave piecewise linear function (CPL) are often used. But in such models, the packet view is lost, and it leads to pessimistic approximation. Staircase functions can model the packet aspect, but not the shaping. It exists a generic class, UPP, which generalises both CPL and staircases, but it leads to computation times that are usually more than one order of magnitude larger than with CPL and staircase functions (see  

Contribution and organisation of the paper. In this study the proposal is to make a combined use of two simple classes (staircases and CPL), and to juggle with both in the
same algorithms, in order to not resort to more computationally involved class of functions. The paper, after a recap on network calculus (Section 2), discusses (Section 3) the problem of the flow modelling in network calculus, and the running time performance versus accuracy trade-off. Then, a common algorithms (with its proof) designed to compute bounds in FIFO algorithm is presented in Section 4. The main contribution of the paper, the adaptation of the algorithm is presented in Section 4.2. Section 6 presents an experiment on a realistic AFDX configuration. Section 6 concludes.

2. NETWORK CALCULUS

Network calculus is a theory to obtain deterministic upper bounds in networks. It is mathematically based on the $(\land, +)$ dioid. This section provides a short introduction to network calculus. The reader should refer to [8, 9] for the first works, and [10, 11] for comprehensive presentations.

Basic notations. $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{R}$ the set of real numbers. The minimum (resp. maximum) operator is denoted $\land$ (resp. $\lor$). $[x]^+ \overset{df}{=} x \lor 0$, and $[x]$ is the ceiling function ($\{x \in \mathbb{N}, x \leq [x] < x+1\}$.

Network calculus mainly handles non decreasing functions, null before 0: $\mathcal{F}$.

$$\mathcal{F} = \{ f : \mathbb{R} \rightarrow \mathbb{R} \mid x < y \implies f(x) \leq f(y) \}$$

Figure 1: Common curves and illustration of the delay

There are six common curves, parametrised by real non negative values $d, R, T, r, b, \tau$. These curves, shown in Figure 1, are: latency $\delta_d$, rate $\lambda_R$, rate-latency $\beta_{R,T}$, token bucket $\gamma_{r,b}$, stair-case $\nu_{T,\tau}$, test $\delta_{(>)}$ and they are defined as:

$$\delta_d(t) = \begin{cases} 0 & \text{if } t \leq d \\ \infty & \text{otherwise} \end{cases}$$
$$\gamma_{r,b}(t) = (rt + b)\delta_{(>)}(t)$$
$$\beta_{R,T}(t) = R[t - T]^+$$
$$\nu_{T,\tau}(t) = \left[\frac{t + \tau}{T}\right] \land \delta_0$$

$\delta_d$ should be mentioned that, even if all the algorithm principles have been presented in the literature [11], this is the first time, to the best of our knowledge, that a complete detailed algorithm is presented and proven.

Three basic operators on $\mathcal{F}$ are of major interest; the convolution $\ast$, the deconvolution $\otimes$ and the sub-additive closure $\delta^*$:

$$f \ast g)(t) = \inf_{0 \leq s \leq t} (f(t - u) + g(u))$$
$$f \otimes g)(t) = \sup_{0 \leq u} (f(t + u) - g(u))$$
$$f^* = \delta_d \land f \land (f \ast f) \land \cdots$$

There are several well-known properties of these operators and the common curves. For instance, the convolution is associative and commutative, $f \ast (g \ast h) = (f \ast g) \ast h$.

Arrival and service curves. A flow is represented by its cumulative function $R \in \mathcal{F}$, where $R(t)$ is the total number of bits sent by this flow up to time $t$. A flow $R$ is said to have an arrival function $\alpha$ as arrival curve (denoted $R \prec \alpha$) if $\forall t, s \geq 0 : R(t + s) - R(t) \leq \alpha(s)$. It means that, for any instant $t$, the flow $R$ will produce at most $\alpha(s)$ new data in $s$ time units. An equivalent condition, expressed in the $(\land, +)$ dioid is $R \leq R \land \alpha$. This min-plus definition of arrival curves leads to three interesting results: first, if $\alpha$ is an arrival curve for $R$, also is $\alpha^*$; second, if $\alpha$ is an arrival curve for $R$, any $\alpha^* \geq \alpha$ also is; third, if $\alpha$ and $\alpha'$ are arrival curve for $R$, also is $\alpha \land \alpha'$.

A server $S$ is a relation between input flows and some output flows, denoted $R \xrightarrow{S} R'$. A server offers a service curve $\beta$ (denoted $S \succeq \beta$) if for all arrival flows, the relation between the input flow $R$ and the output flow $R'$, one has $R' \geq R \ast \beta$. In this case, $\alpha = \alpha \land \beta$ is an arrival curve for $R'$. The delay experimented by the flow $R$ can be bounded by the maximal horizontal difference between curves $\alpha$ and $\beta$, formally defined by $h(\alpha, \beta)$ (a graphical interpretation of $h$ is shown in Figure 1).

$$h(\alpha, \beta) = \sup_{s \geq 0} (\inf \{ t \geq 0 \mid \alpha(s) \leq \beta(s + \tau) \})$$

A server can also have a shaping curve $\sigma \in \mathcal{F}$, meaning that the output $R'$ of the server always respects the shaping curve, i.e. $\forall t, \Delta \geq 0 : R'(t + \Delta) - R'(t) \leq \sigma(\Delta)$. It can also be said that each output has $\sigma$ as arrival curve.

Sequence of servers. When a flow goes through a sequence of server $S, S'$, then, the sequence itself can be seen as a server $S; S'$ defined by

$$R \overset{S;S'}{\rightarrow} R' \iff \exists R': R \overset{S}{\rightarrow} R' \text{ and } R \overset{S'}{\rightarrow} R'$$

These first results allow to handle linear topologies, like the one of Figure 2. Given the arrival curve $\alpha$ of flow $R$, and the services curves $\beta, \beta'$ of network elements $S, S'$, we are able to get a bound on the delay in $S$: $h(\alpha, \beta)$, and another for the delay in $S'$: $h(\alpha \land \beta, \beta')$.

A famous result of the network calculus is known as “pay burst only once” (PBOO). It states that if a flow goes trough artion.
bound of the delay, $\beta_d$ is a commonly used residual delay for each $R_i$.

$$\beta_d = \delta_h(\beta, \alpha_1 + \alpha_2)$$

(11)

A last result of importance for this paper concerns packetizers. A packetizer $P$ is an element that inputs some “fluid” flow $R$ and that outputs packetized flow $R'$. In a more formal way, each flow $R$ is assumed to carry a sequence of packets. Let $s_1, s_2, \ldots, s_\infty$ be the size of the packets. Let $P: \mathbb{R} \rightarrow N$ defined by $P(x) = \max \left\{ j \sum_{i=1}^{j} s_i < x \right\}$, denotes the index of the last full packet. Then, $R'(t) = \sum_{i=1}^{P(R(t))} s_i$.

An important feature of the packetizer is that, repacking does not increase the delay. That is to say, in case of a sequence $R \xrightarrow{\delta} R' \xrightarrow{\delta} R''$, assuming that $R$ is packetized (i.e. $R(t) = \sum_{i=1}^{P(R(t))} s_i$), then, the worst delay of the combined system $S; P$ is the same as for the fluid system [12, Th. 1.7.1].

$$d(R, S; P) = d(R, S)$$

(12)

It does not mean that the packetizer has no effect on the flow, but when considering the worst delay, since all bits of a packet enter the combined system at the same date, since the worst delay is experience in the fluid system by the last bit, and since the last bit of a packet is not delayed by a packetizer, then, the packetizer does not add any delay.

It should be noted that, if $S$ is a FIFO server, then $S; P$ is also a FIFO server, and eq. (11) can be used. Also, if a flow with arrival curve $\alpha$ crosses a packetizer, the output will have arrival curve $\alpha + l_{\text{max}}$, with $l_{\text{max}}$ the maximal packet size of the flow.

$$R \xrightarrow{\delta} R', R < \alpha \implies R' < \alpha + l_{\text{max}}$$

(13)

Last, it must be remind that a packetized flow can have a fluid arrival curve: it is a loss of information, and lead to pessimistic over-approximation, as will be presented in next section, but it does not gives optimistic bounds.

### 3. AFDX FLOW MODELLING ISSUES

In AFDX, all data flows are Virtual Links. A Virtual Link uses a static multicast path through the network and consists of a sporadic stream: frames of the stream are of bounded size, $s_{\text{max}}$, and there is a minimal duration between two successive frame transmissions (known as the “Bandwidth Allocation Gap”, or BAG).

One can choose different arrival curves to model such a sporadic flow with maximal frame size $s_{\text{max}}$ and inter-frame gap $BAG$. A first idea is to use a simple staircase function $\alpha(\Delta) = s_{\text{max}} \times \nu_{\text{BAG}}(\Delta) = s_{\text{max}} \left[ \frac{\Delta}{\text{BAG}} \right]$ (as presented in Figure 3). But one may prefer a more simple linear function $\alpha'(\Delta) = \frac{\text{max}}{\text{BAG}} s_{\text{max}}(\Delta) = s_{\text{max}}(1 + \frac{\Delta}{\text{BAG}}) \wedge \delta_0$. 

![Figure 2: A flow going through two network elements in sequence](image)

![Figure 3: A single shared network element](image)

![Figure 4: Illustration of the packetizer](image)
4. AN ALGORITHM TO COMPUTE BOUNDS ON AFDX NETWORKS

We present in this section an algorithm to compute WCTT bounds on an AFDX network. In fact, it can be used for any FIFO store and forward networks, assuming that the basic operations on network calculus can actually be computed. This algorithm is parametrised by the arrival and service curves. We will then adapt this algorithm to make a combined use of two simple classes of functions, and provides more accurate results with a low computation time (Section 4.2), as it will be shown in the experiments of Section 5.

4.1 An algorithm for FIFO store and forward network

Let us consider a store and forward network, with FIFO policy, and without cyclic dependency. The network is made of switches interconnected by full duplex links. The modelling of a switch, illustrated in Figure 8, is as follows. Input ports are modelled as packetizers, the switching fabric consists in a single delay $\delta$, with an important property: it keeps the flow packetized. Lastly, the output port is a server with a minimal curve and a shaping curve. A $n \times n$ switch is modelled in network calculus by $2n+1$ servers: $n$ packetizers, one delay, and $n$ servers with minimal service and shaping curve. In the example of Figure 8, assuming a maximal size of $1$ Ko (i.e. 8Kb), a fabric delay of $16\mu$s and output port running at $10$ Mb/s, servers $S_1$, $S_2$ are packetizers with $l_{max} = 8 \times 10^3$, $S_3$ has a service curve $\delta_{10}$ and servers $S_4$, $S_5$ have service curve $\lambda_{10}$ and also shaping curve $\lambda_{10}$.

![Figure 8: Switch model (2x2)](image)

The algorithm applies on a set of server, $S_1, \ldots, S_n$, crossed by a set of end-to-end flows $F_1, \ldots, F_m$. Each end-to-end flow is itself a sequence of (local) flows (i.e. cumulative functions) and servers: $F_i = F_{i_1}^{S_{i_1}} \circ F_{i_2}^{S_{i_2}} \circ \ldots \circ F_{i_n}^{S_{i_n}}$ (the length of the end-to-end flow is denoted $|F_i| = p$). One needs to encode the topology with two kinds of elements: flows and servers. We decide to use notations introduced for another kind of bipartite graph: Petri nets. The input (resp. output) flows of a server $S_j$ is denoted $\bullet S_j$ (resp. $S_j \bullet$). Similarly, the source of a flow $F_k$ is denoted $\bullet F_k$ and its destination $F_k \bullet$. By induction, $\bullet S_j$ is the set of nodes who have one output, that is an input of $S_j$, and $S_j \bullet \cap \bullet S_k$...
is the (possibly empty) set of flows going from $S_j$ to $S_i$.  

The principle of the algorithm is the following. In each server, the global delay of the server is computed, and propagated on each flow using eq. (11). To compute this global delay, the arrival curve of the global input flow must be computed: it is not computed flow per flow, but grouping the flow per source. This is related to shaping: if 4 flows goes into a network element $S$ (i.e. $S = \{R_i\}_{i \in \mathbb{N}}$), but they are in fact produced by only two nodes ($\{R, R_2\} = S'$, $\{R_2, R_4\} = S''$), considering that each flow $R_i$ has the individual arrival curve $\alpha_i$, and that the servers $S'$, $S''$ have the shaping curves $\sigma'$, $\sigma''$, the arrival curve for the global flow $R_1 + R_2 + R_3 + R_4$ is $(\alpha_1 + \alpha_2) \wedge \sigma' + (\alpha_3 + \alpha_4) \wedge \sigma''$.

![Figure 9: Grouping/shaping principle](image)

Without loss of generality, to simplify the description algorithm, we assume that there is a specific source node $S_0$ such that all end-to-end flows are emitted by this source (i.e., for all flow $F_i$, $\forall (F_i^0) = S_0$). We also assume that each packetizer has only one source.

The algorithm requires some ordering for the computations: to compute the input of server $S_j$, one first needs to compute the output of all servers in $\bullet \bullet S_j$. To avoid some topological sorting, the algorithm is written in a recursive way. This leads to Algorithm 1 as an illustration, a partial computation is presented in Appendix A and its proof is presented in Appendix A.

### 4.2 Juggling with two simple classes of service

![Figure 10: CPL overapproximation of a stair-case function](image)

We consider simple linear shaping functions (i.e. $\sigma_{P_i} = \lambda_{R_i}$) and rate-latency services (i.e. $\beta_i = \beta_{R_i, T_i}$). If the arrival curves are linear $\gamma_{x,y}$ functions, i.e. a simple subset of CPL functions, all operations preserve the CPL shape of $\alpha^k_i$ functions.

---

<sup>3</sup>An example is provided in Appendix A

<sup>4</sup>with some test to avoid to redo twice the same computations

---

**Algorithm 1:** Acyclic FIFO net – per server delay propagation and group shaping

**Input:** Acyclic network description: servers $\{S_1, \ldots, S_n\}$, paths of flows $\{(F_1, \ldots, F_m)\}$

**Input:** Arrival curves of initial flows $(F_i^0 < \alpha_i)$; Curves of minimal service $(S_j \geq \beta_j)$; Shaping curves $\sigma_j$ of each $S_j$.

**Data:** Computed arrival curves of flows: $\alpha^k_i$; Total input for server $S_i$: $\alpha^S_i$.

**Result:** Maximum delay for each server: $d_j$; Maximum delay for each flow: $d^F_i$.

1. **Function** GroupAC($S_j, F$):
2.   ```
3.     if $S_j = S_i$ then
4.         return $\sum P_i \in F \sigma^k_i$;  // $F \subset S_i$ $\implies$ $k = 0$
5.       else if $S_j$ is an agent server then
6.         if $d_j = \text{null}$ then
7.             $\alpha^S_j \leftarrow \sum P_i \in \bullet \bullet S_j \text{ GroupAC}(S_p, S_p \cap \bullet \bullet S_j)$;
8.             $d_j \leftarrow \text{b}(\alpha^S_j, \beta_j)$;
9.             foreach $F_i \in \bullet \bullet S_j$ do
10.                $\alpha^S_{i+1} \leftarrow (\alpha^S_i \odot d_j)$;
11.       return $(\sigma_j \land \sum P_i \in F \alpha^k_i)^*$
12.  else if $S_j$ is a P-packetizer with per flow maximal packet size $l_{\text{max}}$ then
13.     Assert($\bullet \bullet S_j = 1$);
14.     if $d_j = \text{null}$ then
15.         $d_j \leftarrow 0$;
16.         GroupAC($\bullet \bullet S_j, \emptyset$);
17.         foreach $F_i \in \bullet \bullet S_j$ do
18.             $\alpha^S_{i+1} \leftarrow \alpha^S_i$
19.             $l^\text{max}_i = \sqrt{l^\text{max}_i}$;
20.             $S_p \leftarrow \bullet \bullet S_j$
21.         return $(\sigma_p + l^\text{max}_i \land \sum P_i \in F \alpha^k_i)^*$
22.     // Main
23. begin
24.     // Initialisations
25.     for $j = 1$ to $n$ do
26.         $d_j \leftarrow \text{null}$;
27.     for $i = 1$ to $m$ do
28.         $\alpha^S_i \leftarrow \alpha^S_i$;
29.     // Forcing evaluation for each server
30.     for $j = n$ downto 1 do
31.         GroupAC($S_j, \emptyset$);
32.     // Per flow delay
33.     for $i = 1$ to $m$ do
34.         $d^F_i \leftarrow \sum P_i \odot d^F_i$
35.         // with $F_i = F_i^0 \xrightarrow{S_i} \ldots \xrightarrow{S_m} F_i^n$
Table 1: Comparison of the 3 methods on the case-study

<table>
<thead>
<tr>
<th>Method</th>
<th>CPL (float)</th>
<th>CPL/b νT,τ (float)</th>
<th>UPP (rat.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation time</td>
<td>0.9 s</td>
<td>1.1 s</td>
<td>7.2 s</td>
</tr>
<tr>
<td>Min gain</td>
<td>-</td>
<td>0%</td>
<td>0.15%</td>
</tr>
<tr>
<td>Max gain</td>
<td>-</td>
<td>7.8%</td>
<td>15.2%</td>
</tr>
<tr>
<td>Av. gain</td>
<td>-</td>
<td>2.49%</td>
<td>5.92%</td>
</tr>
<tr>
<td>Min gain on 1000 biggest</td>
<td>-</td>
<td>0.8%</td>
<td>3.7%</td>
</tr>
<tr>
<td>Max gain on 1000 biggest</td>
<td>-</td>
<td>4.4%</td>
<td>11.9%</td>
</tr>
<tr>
<td>Av. gain on 1000 biggest</td>
<td>-</td>
<td>2.9%</td>
<td>8.3%</td>
</tr>
</tbody>
</table>

The experiments are conducted on a realistic industrial configuration provided by Thales Avionics, with 104 nodes connected to an AFDX network having 8 switches. The number of virtual links amounts to 974 leading to 6501 WCTT latency constraints. This case-study has been first introduced in [5].

The configuration is analysed with 3 methods implemented in the RTaW-Pegase WCTT evaluation tool [5]. The first method, denoted by CPL in Table 11 uses Algorithm 1 with the fluid modelling of arrival curve where each VL \(i\) has an arrival curve \(a_i^\tau = \gamma_i \max_{\nu \in \mathcal{A}} \nu_i \max\). Then, the computation is done within the CPL function class, using floating point numbers. The second method, denoted UPP in Table 1 also uses Algorithm 1 but with stair-case arrival curves: \(a_i = g_i \max \times \nu_i \mathcal{A}_i \). Then, the delay propagation requires the general UPP function class, which implies rational computation. The third approach , denoted by CPL/ν in Table 1 also implements the algorithm adaptation presented in Section 4.2 also considering stair-case arrival curves, but the computation remains in the realm of floating point numbers.

The results of the experiments are presented in Table 1 and Figures 11 and Figures 12, the latter being a zoom on the results for the 1000 paths with the largest delays. When considering that the reference method is the fluid modelling, the new algorithm provides a 2.9% improvement, at the expense of an increase of 22% of the computation time, whereas the UPP modelling provides an 5.9% improvement, at the expense of an increase of 700% of the computation time.

These results suggest that the approach developed in this paper offers a reasonable trade-off between the two other methods. The computation time is 22% larger than the simple CPL method, and 84% smaller than the UPP one. The accuracy of the results also is between both other methods, as shown in Figures 11 and Figures 12. When the UPP gain is on average equal to 5.92%, CPL/ν leads to an average gain of 2.9%.

6. CONCLUSION

Network calculus has been used for more than 15 years to compute guaranteed upper bounds in networks. However, in large scale systems such as avionics ones, computation times can be an issue. To make network calculus more practical and suited to a wider range of use-cases, algorithms corresponding to different trade-offs between computation time and result accuracy should be developed.

This paper considers the case of a FIFO network, and makes some adaptations to a common algorithm in order to obtain a meaningful trade-off between existing solutions. This proposal can be of particular interest for design space exploration. Experiments conducted on a realistic AFDFX configuration confirm the initial intuition: when considering that the reference method is the fluid modelling, the new algorithm provides a 2.9% improvement, at the expense of an increase of 22% of the computation time, whereas the most detailed modelling provides a 5.9% improvement at the expense of an increase of 700% of the computation time.

Another contribution of the paper is that it provides, to the best of our knowledge, the first detailed presentation and proof of the delay computation algorithm in the FIFO case.

7. REFERENCES
Figure 11: Path delay bounds with the 3 methods (VLs are sorted by increasing delay computed with the second method)

Figure 12: Zoom on the last part of Figure 11


where three flows are produced by two end-systems, crossing 

Network Calculus


elling and also are the sub-additive closures in the algorithm. 


APPENDIX

A. EXAMPLE FOR ALGORITHM 1

The Algorithm \[1\] is illustrated with a simple example where three flows are produced by two end-systems, crossing one 2x2 switch (Figures 13). The system is modelled with three flows \( F_1, F_2, F_3 \) and six servers \( S_1, \ldots, S_6 \). The servers \( S_1, S_2 \) are packetizers, and the others are aggregate servers.

To give an overview of notations: \( S_2^* = \{F_1, F_1\}, S_2 = \{F_2, F_2\}, S_3 = (S_1, S_1), S_4^* \cap S_5 = \{F_2, F_2\} \).

First of all, the arrival curves are initialised \((\alpha^S_0 = \alpha^*_{\sigma})\).

The algorithm first computes GroupAC\((S_1, \emptyset)\). Since this is the first time this servers is crossed \((d_0 = null)\), \(\alpha^S_0\) is computed as GroupAC\((S_1, F_1^0)\). An anthropomorphic anal-

ogy would be: “Hey, \(S_1\), give me an arrival curve for \(F_1^0\).

Now, control is passed to \(S_1\), that requires the computation of GroupAC\((S_2, \emptyset)\), which leads to the computation of \(\alpha^S_2 = \text{GroupAC}(S_0, \{F_2, F_2\})\).

The control has reached the origin \(S_0\), and the result is simply the sum of the arrival curves: GroupAC\((S_0, \{F_2, F_3\}) = \alpha^S_2 + \alpha^S_3 = \alpha^S_2 + \alpha^S_3\) (see eq. 5).

With this information, \(S_2\) can computes its own delay, \(d_2\), and propagate it to \(F_1^1\) (eq. 12): \(\alpha^S_1 = \alpha^S_1 \otimes d_2\) (eq. 11).

The packetizer \(S_1\) now, first computes the arrival curve of its output \((i.e., \alpha^S_2 = \alpha^S_1)\) by application of eq. 11 and eq. 12, and secondly can give surface GroupAC\((S_1, F_2^0)\) as the minimum of \(F_2^0\) and its own shaping curve \((\sigma_2 + l^{\text{max}})\).

At last, \(S_0\) is able to computes its own delay \(d_0 = h((\sigma_2 + l^{\text{max}}) \land \sigma_4, 0)\). And the end-to-end delay of \(F_3\) is \(d_3 = d_2 + d_0\).

The same can be done from \(S_1\) for flows \(F_1\) and \(F_2\).

B. PROOF OF ALGORITHM 1

The main function is GroupAC\((S_j, F)\). The semantics of this function is to compute an arrival curve for the group of flows \(F\) (i.e. \(F \prec \text{GroupAC}(S_j, F)\)), assumed that all flows in \(F\) have the same source \(S_j\) (the parameter \(S_j\) could have been deduced from \(F\) but it is given to improve readability).

To keep it simple, the switch fabric is omitted in the mod-

ing and also are the sub-additive closures in the algorithm.