UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

BRUNO OLIVEIRA CATTELAN

Deterministic Network Calculus for Multicasting: A Numerical Comparison Between Explicit Intermediate Bounds and Multicast Feed-Forward Analysis

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Alberto Egon Schaeffer-Filho
Coadvisor: Dr. Steffen Bondorf

Kaiserslautern
July 2018

ABSTRACT

Nowadays networked systems have become widely spread. Computer networks are not only important for businesses and entertainment, but also for specific safety-critical applications. One important example of a special-purpose network for such safety-critical applications are the Avionics Full-Duplex Switched Ethernet (AFDX) networks, which have been patented by Airbus. These networks require guarantees about their performance, and Deterministic Network Calculus (DNC) has been used to certificate them. An important characteristic is that their data flows are defined as Virtual Links (VL), which can be multicast. However, for a long time DNC was not able to properly analyse such flows. This limitation was previously circumvented by making overly pessimistic assumptions about the demands of the flows in the network. Although valid, this kind of analysis leads to over-provisioning of the network, that in turn means unnecessary increases in cost. In this work, we discuss the two most performant DNC multicast analysis methods presented in the literature. These are the Explicit Intermediate Bounds (EIB) and the Multicast Feed Forward Analysis (MFF). We compare both algorithms to the Unicast Transformation (UT) technique to analyse multicast flows regarding the quality of the bounded delay. We also compare both algorithms to each other, and offer an comparative analysis of their performances.

Keywords: Deterministic Network Calculus. AFDX. Explicit Intermediate Bounds. Multicast Feed Forward. Unicast Transformation.

Cálculo Determinístico de Redes para Multicasting: Uma Comparação Numérica entre as análises Explicit Intermediate Bounds e Multicast Feed Forward

RESUMO

Hoje em dia, redes de sistemas estão amplamente presentes. Redes de computadores não são importante apenas para negócios ou entretenimento, mas também para aplicações críticas. Um exemplo importante de uma rede de propósito específico para tais aplicações críticas são as chamadas redes Avionics Full-Duplex Switched Ethernet (AFDX), que foram patenteadas pela Airbus. Essas redes precisam de garantias sobre sua performance, e o Cálculo Determinístico de redes (DNC) já foi usado para certificá-las. Uma importante característica delas é que seus fluxos de dados são definidos como Links Virtuais (VL), que podem ser multicast. Apesar disso, por muito tempo o DNC não era capaz de analisar com precisão esse tipo de fluxo de dados. Essa limitação foi anteriormente contornada assumindo suposições demasiadamente pessimistas sobre as demandas dos fluxos de dados na rede. Apesar de válido, esse tipo de analise leva para um suprimento exagerado de recursos da rede, o que leva a aumentos desnecessários do custo da rede. Nesse trabalho, nós discutimos as duas melhores análises multicast oferecidas por DNC na literatura. Essas são a Explicit Intermediate Bounds (EIB) e a Multicast Feed Forward Analysis (MFF). Nós comparamos ambos os algoritmos ao Unicast Transformation (UT) para analisar fluxos de dados multicast quanto à qualidade da latência limite computada. Nós também comparamos ambos algoritmos entre si, e oferecemos uma análise comparativa de sua performance.

Palavras-chave: Cálculo Determinístico de Redes. AFDX. Explicit Intermediate Bounds. Multicast Feed Forward. Unicast Transformation.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AFDX	Avionics Full-Duplex Switched Ethernet
ARB	Arbitrary
BAG	Bandwidth Allocation Gap
DNC	Deterministic Network Calculus
EIB	Explicit Intermediate Bounds
ES	End-System
FA	Forward End-To-End Approach
FIFO	First In First Out
Foi	Flow of Interest
HA	Holistic Approach
l.o.	leftover
MFF	Multicast Feed Forward Analysis
PBOO	Pay Burst Only Once
PMOO	Pay Multiplexing Only Once
SFA	Separate Flow Analysis
TA	Trajectory Approach
TFA	Total Flow Analysis
UT	Unicast Transformation
VL	Virtual Link

LIST OF SYMBOLS

$\otimes$	Convolution
$\oslash$	Deconvolution
$\ominus$	Non-decreasing Upper Closure
α	Arrival Curve
β	Service Curve
h	Horizontal Distance

CONTENTS

1 INTRODUCTION

Some network systems operate in life-critical situations. Such systems require certification that some performance bounds are met. Otherwise in an emergency the system could fail, resulting in a loss of property or life. Although problems can occur in well designed networks, failures such as packet loss due to increased traffic can be prevented by a worst-case analysis of the system, and a corresponding provisioning as to fulfil the bounded demands. Over the years, Deterministic Network Calculus(DNC) (Le Boudec; THIRAN, 2001) (CHANG, 2000) has established itself as the main tool for such analysis. It is a worst-case framework, that allows designers to bound pessimistic delays for flows. These are upper bounds, which means that the real delay value will never be larger than them. An example of such a system are the Avionics Full-Duplex Switched Ethernet (AFDX) networks (TECHNICAL REPORT AERONAUTICAL RADIO INC., 2002-2005). This is a patented topology from Airbus, which requires certification. An interesting characteristic of these networks is that they can have multicast flows.

1.1 Motivation

The pessimistic nature of DNC makes it suitable for the certification of real-time systems. However, an overly pessimistic analysis can result in over provisioning the system. This translates to an unnecessary increase in cost, and underutilization of the network. For this reason, different methods have been proposed over the years to improve the bounds given by DNC. Even so, for a long time DNC did not provide an efficient method for bounding the delay of multicast flows.

The literature, on the other hand, offers many different methods for multicast flow analysis. Such methods include the Trajectory Approach (TA) (MIGGE, 1999), which is an adaptation of the Holistic Approach (HA) (TINDELL; CLARK, 1994), and the Forward End-to-End Approach (FA) (KEMAYO et al., 2014). TA has been used for the study of multicast systems, including AFDX networks as shown in (BAUER; SCHARBARG; FRABOUL, 2009). FA as well, being shown to outperform some of the tradicional DNC multicast analyses in (KEMAYO et al., 2014). Both analyses are later compared to the DNC state-of-the-art multicast analysis algorithms presented in this work, in (BONDORF; GEYER, 2016). In this same work, the modern DNC analyses are shown to perform equally or better than TA and FA, which is why such new analyses are the

focus of our work.

With so many different approaches, it can be difficult to decide which one to use. Therefore, it is important to compare different analyses, and point their weaknesses and strengths so that we can more efficiently apply them in the right scenarios.

1.2 Objectives

This work primarily intends to compare the state-of-the-art algorithms for multicast DNC, not only between themselves but also with the traditional analyses available in the literature. In order to do that, we also had as a goal the implementation of such algorithms, which as of the time of writing had no known implementations. With this, we aim to help future network designers and researchers to select the best analyses for their needs. Not only that but, these implementations will be freely available in a open source tool. As a side objective, we would also like to offer a more approachable introduction to DNC. With this, the authors hope that more researchers find interest in this promising area.

1.3 Organization

In Chapter 2 we show the necessary background on DNC. We start by describing the model used for describing topologies, then the model used for the devices and the data flows. With these models in mind, we proceed to show the DNC theorems, and use them to describe the three most used unicast DNC analyses. We show in Chapter 3 some other comparative works available in the literature. We also describe two traditional multicast analyses, and give an initial comparison of them. The next chapter describes the state-of-the-art algorithms for multicast flow analysis, implemented during the development of this work. Moreover, we also show which traditional algorithms were used as a baseline for comparing them. In Chapter 5 we compare the analyses shown in the previous chapter. We start by a theoretical comparison, showing how one analysis can outperform the other. This is followed by an experimental evaluation, which is divided into two sections. First, the algorithms are executed in an AFDX environment. The second part uses a topology which is more similar to the Internet environment. Chapter 6 concludes the work, and presents future research directions.

2 NETWORK CALCULUS BACKGROUND

This work describes in detail some of the most advanced Deterministic Network Calculus analyses in the literature. For this reason, we present in this chapter the theory behind DNC's unicast analyses. Since a more in-depth look is out of the scope of this work, we suggest the interested reader to look at (Le Boudec; THIRAN, 2001) and (CHANG, 2000). These analyses bound both the delay and the backlog of given flows in a network. These bounds are deterministic, and represent the worst-case for a given scenario. For this reason, DNC is commonly used to the study of real-time systems.

We start the study of the DNC theory by describing how the networks are modelled in the context of a DNC analysis. This includes the translation of a topological description to a more detailed one, as well as how to circumvent some limitations inherent to the theory. Later, we describe in detail how the data flows and the servers are modelled by using simple curves. In order to introduce the analyses, we must describe the Min-Plus algebra. It greatly simplifies the description of DNC theorems. These theorems are necessary to finally describe the DNC analyses. A brief description of AFDX networks is also given at the end of this chapter, when we discuss the applications of the theory.

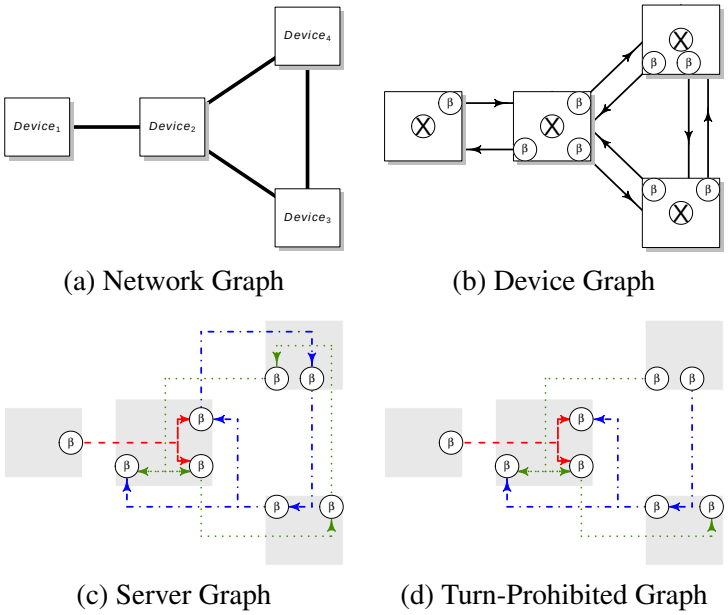
2.1 Network Modeling

DNC can be used to model and analyze many types of systems such as networks, concurrent programs or digital circuits. This work focuses on the study of network systems. In this section we describe how to model a network according to DNC's theory, given a topological description of a network. This is done by using incrementally more complex models; the Network Graph, the Device Graph, and the Server Graph. These steps are also described in (CATTELAN; BONDORF, 2017).

2.1.1 Network Graph

It is natural when describing a network to use a topological description. This model is also called a network graph. It can be defined as a graph G with N nodes and M edges. Here devices are represented as nodes and connections between devices are edges. Because we assume Ethernet connections, all edges are bidirectional. This will

Figure 2.1: Network Modeling Example



(a) Network Graph (b) Device Graph

(c) Server Graph (d) Turn-Prohibited Graph

Source: Author

be important in later sections. An example of such a graph can be seen in Figure 2.1a. However, this model does not describe the network in as much detail as is necessary for the application of DNC. For this reason a Device Graph is necessary.

2.1.2 Device Graph

This model also takes into consideration the output ports of the links. Since DNC works on the queues of the outgoing links, it is not necessary to model the input ports. Each bidirectional edge is now represented by two unidirectional links. Figure 2.1b shows the refined graph for the previous example. The switching fabric of the devices is also visible in the image. It is the symbol in the center of each device. For their modelling it is necessary to define a switching policy, which describes how the device serves incoming packets. As we make no assumption regarding this policy, it is safe to model the switching fabric with arbitrary multiplexing (ARB). By considering this policy we make the model more general, but in exchange we increase its pessimism. This can be seen in (SCHMITT; ZDARSKY; FIDLER, 2008). An ARB policy means that we use the worst-case for a given flow. That is, all other flows must be served before it.

Usually when drawing the refined graph, arrows are drawn in and out of the switching fabric. However, this often leads to cluttered representations. For this reason they were omitted in Figure 2.1b. An example of a complete drawing can be seen

in (CATTELAN; BONDORF, 2017).

2.1.3 Server Graph

In the last modelling step we consider not the devices, but only their output ports. Those ports are now called *servers* [1]. Each bidirectional link in the network graph will have exactly two servers in the server graph, one for each direction. As shown in Figure 2.1c the devices are no longer modelled. This loss of information is resolved by connecting the incoming links of a device to all of its servers. With this any packet routed from one device to another in the device graph is guaranteed to still have a valid path. The exception being the server that represents the port back to the source device. This prevents cycles from one device to another device connected directly to it.

It is important to note that data flows defined in the device graph need to be translated to the server graph. As described in this subsection, the devices used to describe the original path are no longer modelled; only their output ports. Therefore, paths from the device graph are translated to a sequence of output ports, following the original route. As no input port is modelled, the sink of such a server graph path will in fact be the output port which would lead to the final device. This ignored link is referred to as the *last-hop*. From this point on, we will only refer to server graphs, unless otherwise stated.

Another point worth mentioning is the effect of modelling only the output ports. One could argue that this reduces the realism of the analysis, since we cannot model delays in the input ports too. However, by modifying how the server handles data, one could model the delay imposed by the input ports.

2.1.4 Feed-Forward Networks

Modelling a server graph is not enough to guarantee a successful analysis. Some topologies have cycles, which allow flows to have cyclic dependencies. These kind of dependencies are not handled by the DNC analyses. Therefore, it is necessary to modify the topology of the modelled network in order to enforce the feed-forward property. This property means that the network is cyclic-free. Many solutions were presented in the literature to address this issue.

[1] Since DNC can be applied in contexts different than networks, a general name is used for such constructs when modelling a system.

The simplest method for handling networks of arbitrary topologies is to compute the Spanning Tree of the graph G. This is done by removing all links except the ones necessary for maintaining the connectivity of the graph. As expected, this adds larges amounts of pessimism to the network with regards to the usage of resources. Many links will not be considered while others will be overloaded, especially the link connected to the root.

A more sophisticated approach is the so called Turn-Prohibition algorithm, described below. The version presented here is simplified. This simplified version does not enforce connectivity, which makes it not suitable for some cases. However, it is much more intuitive. For the complete version, we point the reader to the work of (STAROBINSKI; KARPOVSKY; ZAKREVSKI, 2003).

Algorithm 1: The simplified Turn-Prohibition algorithm pseudo-code

```
   Data: A bi-directed graph
   Result: A list of prohibited turns
 1 orderNodes(list_of_nodes);
 2 while list_of_nodes not empty do
 3 |   node = getFirst(list_of_nodes);
 4 |   prohibitTurnsAround(node);
 5 |   allowTurnsFrom(node);
 6 |   removeNode(node);
 7 end
 8 Function orderNodes (list_of_nodes)
 9 |   order nodes by degree (low to high);
10 Function prohibitTurnsAround (node)
11 |   adds the turns around the node to the prohibited list;
12 Function allowTurnsFrom (node)
13 |   adds the turns starting from the node to the allowed list;
14 Function removeNode (node)
15 |   remove the node from the graph;
16 |   remove all incident links;
17 |   orderNodes(list_of_nodes);
```

A *turn* here is defined as a pair of input-output links around a given node. The idea is to remove only the turns that cause such cycles, instead of whole links. As seen in Figure 2.1d, the links incoming to the servers describing Device 3 were removed by the algorithm. Although it seems like no path is possible from Device 2 to Device 3 in this new graph, this is not true. Because the *last-hop* is ignored, as discussed in the previous subsection, a flow ending in the rightmost servers of Device 2 can still reach Device 3 or Device 4, and vice-versa. Hence, all Devices are in fact still reachable, yet

no cycle is possible. Moreover, it has been shown in (STAROBINSKI; KARPOVSKY; ZAKREVSKI, 2003) that the algorithm removes at most one third of all turns in a given graph.

Algorithm 1 is executed over the device graph, but the turns are removed from the server graph. The translation, however, is straightforward. Since each (unidirectional) link in the network graph translates to exactly one server in the server graph, the turns can in fact be read as links in the server graph.

2.2 Flow and Server Modelling

2.2.1 Input and Output Functions

The total amount of packets that have arrived from a flow in a given time t can be described as a cumulative wide-sense increasing function $R(t)$, which means that $R(s) \leq R(t)$ for all $s \leq t$. For this work we will focus on the fluid model, where time and function are both continuous. More formally, $t \in \mathbb{R}^+ = [0, +\infty]$ and R is a continuous function.

This means that $R(t)$ gives the amount of packets that have arrived into a system S up to time t for a given flow f. It is always assumed that $R(0) = 0$. Because data arrives into the system, we call it an input function. An output function $R^*(t)$, on the other hand, is the output of a system after serving an input function.

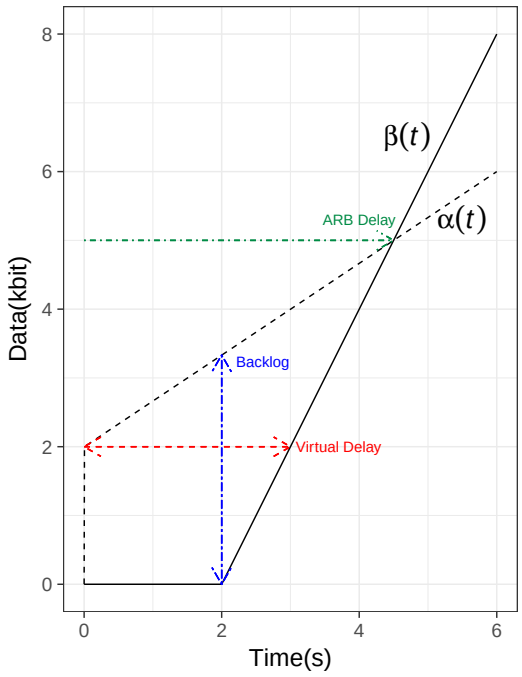
2.2.2 Backlog and Delay

The backlog is the vertical distance between the input and the output function. It gives the amount of data currently on the system. In a simple server with one input, it is the amount of data in the buffer.

The virtual delay is the largest horizontal distance between both curves. This is the time it takes for a packet that arrived at a given time to be served and read at the server's output port. In a First-in-First-out (FIFO) policy, the virtual delay is equal to the real delay. This means that in such a policy this distance is a valid delay.

However, when working with Arbitrary Multiplexing (ARB), the virtual delay is not the real delay. Since flows can interfere with each other, it is necessary to take the time in which both curves intersect as the delay. At this point, all data that was queued

Figure 2.2: Delay and Backlog Example Using $\alpha_{2/3,2}$ and $\beta_{2,2}$



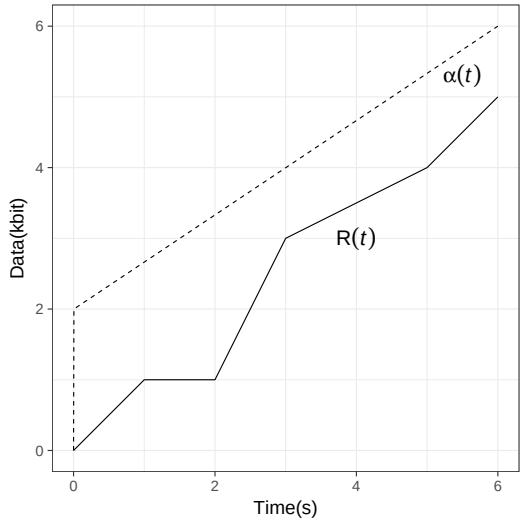
Source: Author

in the server's buffer finishes being served. This is the worst-case, since it assumes that the flow of interest was the last one to the served. After this point, all data that comes into a server is immediately served. In other words, there is no queue, and different incoming flows do not interfere anymore. It is important to note that this overly pessimistic approach is only necessary in the case where multiple flows arrive together at a server's input port. In the case of a single flow arriving alone at a server, the FIFO policy can be used even though the server uses ARB. This differentiation is addressed again in later sections, when describing the DNC unicast analyses. Figure 2.2 shows the backlog as well as the difference between virtual delay and the ARB delay. As mentioned before, the virtual delay is clearly much smaller than the ARB delay.

2.2.3 Arrival Curve

Instead of considering the individual arrival of each packet, it is possible to create an upper bound to the input function. For that, a concave cumulative wide-sense increasing function called arrival curve is used. Even though it provides an increase in the pessimism of the model, this abstraction drastically reduces its complexity. For this reason, arrival curves are widely used in DNC.

Figure 2.3: Arrival Curve Example with $r = 2/3$ and $b = 2$



Source: Author

Definition 2.2.1 (Arrival Curve)**.** A given input function R is said to be constrained by an arrival curve α if for all $s \leq t$

$$R(t) - R(s) \leq \alpha(t-s),$$

2.2.3.1 Token Bucket

Arrival curve constraints were inspired in algorithms such as token bucket and cell rate. The cell rate algorithm is used to describe stair functions, whereas the token bucket is used to describe affine functions. Since we focus on continuous curves, only the token bucket algorithm will be studied here. Such algorithms are used to check if a data flow is conformant to certain limits.

Definition 2.2.2 (Token Bucket)**.** Imagine we have a bucket with capacity b. Every $1/r$ seconds a token is added to the bucket. If the bucket is full, the token is discarded. When data comes to be tested for conformity, the respective number of tokens is removed (e.g. the number of bytes transmitted). If there are not enough tokens, it is said to be non conformant.

In other words, this algorithm allows for flows with rate r and bursts up to b. Because an arrival curve is an upper bound on the arrival of data, we assume the largest data flow that is conformant. This function is called $\gamma_{r,b}$ where r is the rate and b is the burst. For this work we will call α an arrival curve defined by a token bucket. An example of an arrival curve α for a given input function $R(t)$ can be seen in Figure 2.3. In

this case, the input function $R(t)$ has as an arrival curve α, with burst 2 kbits and rate $2/3$ kbits/second. The increase in pessimism can be seen by the vertical distance between the two curves.

With these constructs, a more formal definition of a data flow can be made. We define here unicast and multicast flows. This chapter, however, will only address unicast analyses. The multicast analyses are studied in detail in the next chapters.

Definition 2.2.3 (Unicast Flow)**.** A unicast flow f is composed of an arrival curve α_f and a unicast path P_f. This unicast path P_f is a list of servers S from a *source* to a *sink*, where the source is the first node and the sink is the last node that routes the data.

Definition 2.2.4 (Multicast Flow)**.** A multicast flow f is composed of an arrival curve α_f and a multicast path P_f. This path is a list of n unicast paths $P_{f,j}$ where $n \geq 1$ and $1 \leq j \leq n$. Each unicast path $P_{f,j}$ is a list of servers S_i from a *source* to a $sink_j$. All unicast paths share the same *source*, but have different *sinks*. For this reason, each one can be uniquely identified by the identifier j.

2.2.4 Minimum Service Curve

Similar to the arrival curves, minimum service curves are worst-case bounds. However, they bound the service offered by a server instead of an arrival of data. They are lower bounds to the rate in which a given system serves incoming data. Moreover, these are convex cumulative wide-sense increasing curves.

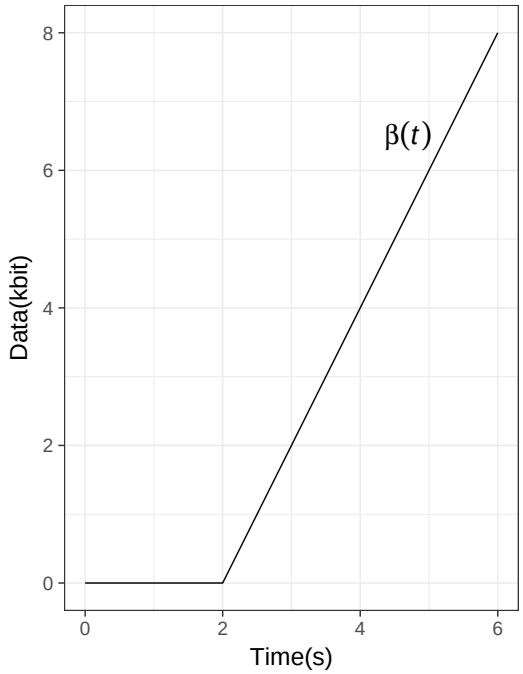
Definition 2.2.5 (Service Curve)**.** A system S is said to offer a service curve β to a flow f iff for all t we can find an s that satisfies

$$R^*(t) - R(s) \geq \beta(t-s),$$

2.2.4.1 Rate-Latency

A common service curve example is a simple rate-latency curve. It can be seen as a linear function starting on the origin with rate R shifted by a latency T. Figure 2.4 shows an example of such a service curve. In this case, the service curve β has a latency T of 2 seconds and rate R of 2 kbits/second. The reader can assume that all service curves β used in this work are defined as rate-latency curves.

Figure 2.4: Service Curve Example with $R = 2$ and $T = 2$



Source: Author

Definition 2.2.6 (Rate-Latency)**.** A rate-latency service curve can be defined as the following:

$$\beta_{R,T}(t) = \begin{cases} R(t-T) & \text{if } t > T \\ 0 & \text{if } t \leq T. \end{cases}$$

The T gives a horizontal translation to the linear function $R * t$. For all values before that, the function is zero. This models the fixed latency of a server, independent of the amount of incoming data.

2.3 Min-plus Algebra

This section first introduces the min-plus algebra. Later, the three most important theorems for Network Calculus are presented. Such theorems are defined in the min-plus algebra, which greatly simplifies their description.

The standard algebra is defined on the triple $(\mathbb{R}, +, \cdot)$ whereas the min-plus algebra is defined on $(\mathbb{R} \cup \{\infty\}, \min, +)$. This means that for the min-plus algebra the addition becomes minimum and the multiplication becomes addition. To better illustrate this we define here the convolution and deconvolution operations in min-plus algebra.

Definition 2.3.1 (Convolution in Standard Algebra)**.** In the standard algebra the convolu-

tion of two functions f and g can be seen as the integral of the point-wise multiplication between them:

$$(f * g)(t) \coloneqq \int_0^t f(t-x)g(x)\mathrm{d}x.$$

In min-plus algebra, however, addition becomes minimum and the multiplication becomes addition.

Definition 2.3.2 (Convolution and Deconvolution in Min-plus Algebra)**.** [2] If f and g are both non-negative wide-sense increasing functions, then their convolution in min-plus algebra is defined as

$$(f \otimes g)(t) \coloneqq \inf_{0 \leq x \leq t}\{f(t-x) + g(x)\},$$

which is the minimum of the point-wise addition of f and g. Similarly, the deconvolution in min-plus algebra is defined as

$$(f \oslash g)(t) = \sup_{u \geq 0}\{f(t+u) - g(u)\}.$$

Using the convolution previously defined, it is possible to concatenate a series of servers into one unique server. This is done by convolving all their service curves into one total service curve. In other words, a series of servers are substituted by one unique server, which has a total service curve.

Theorem 1 (Concatenation of Servers)**.** *Assume a single flow f with an arrival curve α traverses a sequence of two servers s_1 and s_2 with respective service curves β_1 and β_2. Then we can consider the total service β_{total} as being the concatenation by convolution of both servers*

$$\beta_{\text{total}} = \beta_1 \otimes \beta_2.$$

More generally speaking, this can be extended to a sequence of an arbitrarily large number of servers

$$\beta_{\text{total}} = \beta_1 \otimes \beta_2 \otimes \ldots \otimes \beta_n,$$

which, since the convolution operation is associative, can be expressed as

$$\beta_{\text{total}} = \bigotimes_{i=1}^{n} \beta_i.$$

Imagine now that several flows interfere with a flow f_1 in a given server s. It is necessary to lower bound the service curve left offered by s for f_1. Since it is the worst-case, this assumes that all flows are served before f_1.

Theorem 2 (Left-over Service Curve)**.** *Consider now the case of a single server s_1 serving two flows f_1 and f_2 with α_1 and α_2 as their respective arrival curves. Assuming we are*

[2] Using these constructs, we can now define arrival curves as $R \oslash R \leq \alpha$ and service curves as $R^* \geq R \otimes \beta$.

interested in finding the residual worst-case service offered by s_1 to f_1 in an arbitrary multiplexing policy, then it holds that

$$\beta^{\text{l.o.}f_1} = \beta \ominus \alpha_2,$$

where $(\beta \ominus \alpha)(d) := \sup\{(\beta - \alpha)(u) \mid 0 \leq u \leq d\}$ is the non-decreasing upper closure of $(\beta - \alpha)(d)$.

Finally, this last theorem shows how to compute performance bounds using the studied curves.

Theorem 3 (Performance Bounds)**.** *As stated in Subsection 2.2.2, the Backlog is the vertical distance between two curves. This gives us the following bounds for the case of a flow f with arrival curve α being served by a server s with service curve β:*

Backlog*: $Q(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\} = (\alpha \oslash \beta)(0)$*

On the other hand, the virtual delay is:

Delay*: $D(t) \leq \inf\{d \geq 0 \mid (\alpha \oslash \beta)(-d) \leq 0\}$*

The output curve is:

Output Curve*: $\alpha^*(d) = (\alpha \oslash \beta)(d)$.*

In Figure 2.2 the different distances are shown for the arrival and service curves used in the previous example. The backlog is the vertical distance between the two curves. In this case, backlog$= 10/3 - 0$. The value $10/3$ comes from the arrival curve at time 2, and the 0 comes from the service curve at the same point in time. At this point, we have the largest vertical distance between the two curves.

This figure shows also the difference between FIFO and ARB delay bounds. Whereas FIFO takes the largest horizontal distance between the curves virtual delay$= 3 - 0$, the ARB delay takes the point of intersection of both curves, which is $\text{delay}_{\text{ARB}} = 4.5$. Clearly the FIFO delay is much smaller, but it can not always be applied. For instance, the switching policy might not be known, and using ARB guarantees that all results are valid. The next section presents the unicast analyses that use all the properties studied so far. This section also shows in each circumstances each of the delays can be used.

2.4 Unicast Analyses

In this section we present the three most common unicast network analyses available in the literature. They are the Total Flow Analysis (Le Boudec; THIRAN, 2001), the Separate Flow Analysis (Le Boudec; THIRAN, 2001), and the Pay Multiplexing Only

Once (SCHMITT; ZDARSKY; MARTINOVIC, 2008). Such analyses work on server graphs as described in Section 2.1. They allow us to compute worst-case delay and backlog bounds. This work, however, focus mainly on the delay bound. At the end of this section, a small example shows the different bounds computed by each algorithm for the same network.

Before we proceed, it is necessary to explain some of the nomenclature used. Each one of the following analyses makes use of a so called flow of interest (foi). This flow is the data flow currently being analysed. Another point worth noting is that the algorithms shown here are simplifications of the algorithms shown in (SCHMITT; ZDARSKY, 2006). The original algorithms make use of the maximum service curve optimization. This however escapes the scope of this work, and this bound optimization will not be used. This optimization limits the amount of data that can flow from the servers, limiting it to respect the physical capacity of the medium. That is, the output bounds can never be larger than the link capacity. Since we are discussing worst-case analysis algorithms, all results of this work remain valid.

2.4.1 Total Flow Analysis (TFA)

The main idea of this simple algorithm is to compute per server delay bounds and then to sum these individual delays, according to the flow of interest's path. These delay bounds per server are independent of the foi; the flow of interest in this analysis simply gives the trajectory to be analyzed. This analysis' name comes from all flows that share a server with the foi being analyzed as a whole. Differently from the other analyses studied in this section, TFA makes no use of the Concatenation Theorem (Theorem 1). This introduces a considerable amount of pessimism to the algorithm, meaning that the end-to-end delay bound computed by TFA is usually not tight. At the end of this section we present a comparison between the three algorithms illustrating this.

In Algorithm 2 we show a high level description of the algorithm. First, it computes the output bounds reaching the sink of the flow of interest. From the sink it recursively computes the output bounds of the servers leading to the sink until it reaches the source of the foi. In line 10 we see that the algorithm first adds the arrival curves of all flows that accompany the foi until its sink (α_i^{pred}). In the next line we add the arrival curves from the flows that joined the foi's path at some point but do not accompany the foi until the end (α_{excl}). Later in line 13 we subtract these α_{excl} from the node i's service

curve using Theorem 2. Finally, the algorithm computes the output bound from server i in line 14 using α_i^{pred} and β_i^{eff} according to Theorem 3.

The effective service curves from the nodes are stored in the variable β_i^{eff} whereas the total amount of data flowing towards the sink per node is stored in α_i^{pred}. With both the output bounds reaching each server and their respective service curves, the algorithm computes the end-to-end delay by adding the per server delays. Since α_i^{pred} is an aggregation of many flows, to compute the foi's delay it is necessary to use the ARB Delay shown in Figure 2.2 for an ARB multiplexing scenario. That translates to the worst-case, since each server must wait until all incoming data is served. This can be seen in line 5.

Algorithm 2: Pseudo-code for Total Flow Analysis (SCHMITT; ZDARSKY, 2006)

```
1  Function ComputeDelayBound (flow of interest f)
2      ComputeOutputBound(sink(f), all flows to sink(f))
3      delay_total = 0.0
4      forall nodes i on the path from source to sink(f) do
5          delay_total = delay_total + h(α_i^pred, β_i^eff)
6      end
7      return delay_total
8  Function ComputeOutputBound(from node i, flows 𝔽)
9      forall pred(i) do
10         α_i^pred = α_i^pred + ComputeOutputBound(pred(i), {flows to node i from
             pred(i)}∩𝔽)
11         α_excl = α_excl + ComputeOutputBound(pred(i), {flows to node i from
             pred(i)}\𝔽)
12     end
13     β_i^eff = [β_i − α_excl]^+
14     return α_i^pred ⊘ β_i^eff
```

2.4.2 Separate Flow Analysis (SFA)

This is the first algorithm shown in this section to make use of server concatenation (Theorem 1). That theorem allows this analysis to take into account the burst of the flow of interest only once. This property is known as Pay Burst Only Once (PBOO). Because of that property, SFA is known to be superior to TFA. The explanation for this can be seen in (SCHMITT; ZDARSKY; FIDLER, 2008). One could think of it like the following: When computing the horizontal distance between an arrival curve and a service curve, the arrival's curve burst term must be taken into account. TFA makes this

computation per server, which means that this burst term will appear once per delay computation. On the other hand, by concatenating the servers, SFA only has one service curve to compute the horizontal distance. This means that the burst term will appear only once in the computation, hence the Pay Burst Only Once property.

SFA's high level description can be found in Algorithm 3. In this analysis the traffic from the foi is separated from the traffic from other flows as shown in line 4, which is why this analysis is called Separate Flow Analysis. The computation of the output bounds for each server is similar to the one found in TFA. The difference resides in how it works with them. As mentioned above, the SFA works in the end with only one total service curve. In order to do that, the algorithm must first compute the output bounds per server as is done in the TFA. Using them, the algorithm computes the total traffic bound on a server and then subtracts it from the effective service curve of such server. This can be seen from lines 3 to 6. After the final service curves are computed for each server they are concatenated using Theorem 1 in line 8. Finally, the horizontal distance (h) is computed in the next line using both the total service curve from the foi's path as well as its arrival curve.

It is important to note that this algorithm uses the horizontal distance, which is the virtual delay shown in Figure 2.2. Even in an ARB multiplexing scenario, the way in which SFA computes the bounds allows it to use this smaller distance. This is valid because, when bounding the delay, only one flow is considered. This can be seen in the delay computation in line 9; differently from the TFA just presented that used the total incoming data in a given server, SFA uses only the data coming from the flow of interest.

2.4.3 Pay Multiplexing Only Once (PMOO)

This is the most recent unicast analysis available in the literature. A more in-depth explanation of this analysis can be found in (SCHMITT; ZDARSKY; MARTINOVIC, 2008), where it was initially presented. Similar to SFA, PMOO also benefits from the Concatenation Theorem (Theorem 1). However, instead of first computing the leftover service curve and then concatenating the servers, this analysis does it in the reverse order. PMOO first convolves the servers and then subtracts the cross-traffic. Using this order, the burst term of the cross-traffic appears only once when compared to SFA.

Due to its nature, the PMOO analysis presented here is not suitable for FIFO multiplexing networks. For this reason, this algorithm can only be used in ARB multi-

Algorithm 3: Pseudo-code for Separated Flow Analysis (SCHMITT; ZDARSKY, 2006)

```
1  Function ComputeDelayBound (flow of interest f)
2      forall nodes i ∈ path(f) starting at sink(f) do
3          forall pred(i) do
4              α_pred += ComputeOutputBound(pred(i), {flows to node i from
                pred(i)}\{f})
5          end
6          β_i^eff = [β_i − α_pred]^+
7      end
8      β^SF = ⊗_{i=1}^{n} β_i^eff
9      return h(α_f, β^SF)
10 Function ComputeOutputBound(from node i, flows 𝔽)
11     forall pred(i) do
12         α_i^pred += ComputeOutputBound(pred(i), {flows to node i from
            pred(i)}∩𝔽)
13         α_excl += ComputeOutputBound(pred(i), {flows to node i from
            pred(i)}\𝔽)
14     end
15     β_i^eff = [β_i − α_excl]^+
16     return α_pred ⊘ β_i^eff
```

plexing scenarios. This limitation does not affect this work, since we always assume ARB multiplexing. Although PMOO most often outperforms SFA, depending on the configuration of the analyzed network, this may not hold true. This was shown in (SCHMITT; ZDARSKY; FIDLER, 2008).

The high level description of PMOO can be seen in Algorithm 4. When computing the end-to-end delay bound for a given flow of interest, the first step is to compute a total service curve for its path as seen in line 2. Looking at the function *ComputePMOOServiceCurve*, line 5 means that all rejoining flows must be removed. When a flow that shares a path with the foi later diverges and rejoins the foi's path is identified, the analysis substitutes each rejoin as a new flow. Each one of these new flows must have the arrival curve of the original flow at that server. This transformation does not affect the network bounds. In the next line we merge all parallel flows into one flowset $\mathbb{F}$. Parallel here means that all flows have the same ingress and egress servers. This flowset is treated from this point on as one unique flow.

The merging process was initially thought as a runtime optimization; fewer flows help later stages of the algorithm to be more efficient. However, later in (BONDORF; SCHMITT, 2016) this idea was generalized to the computation of output bounds. This

paper also shows that the generalized process actually gives tighter bounds.

The next step is to compute the arrival curves of the flowsets $\mathbb{F}_i$ from line 7 to 11. This is achieved by computing the output bound of the given flows reaching the input of the flowset, i.e. the first common server of the flows. The output bound computation for this algorithm is, however, different from TFA and SFA. First it finds the shared path p between all flows in the flowset $\mathbb{F}$ as stated in line 15 from the server i. These flows are known to share at least some of their paths due to the creation of $\mathbb{F}$ in line 6. In line 16 it saves the split point of the flowset in the variable s. Next it recursively computes the output bounds of the cross-traffic of $\mathbb{F}$ for all the paths leading to s, adding them into a total arrival curve. After computing the PMOO service curve in line 20 it computes the output arrival curve using both the PMOO service curve and the total arrival curve. This is seen in line 21.

With each output bound computed for each flowset, the algorithm now must calculate the PMOO service curve as seen in line 12. This is the highest cost operation of this analysis. For each combination of token bucket curve (2.2.2) taken from the arrival curves of the cross-traffic and the rate-latency (2.2.6) curves from the servers service curves, it creates a new rate-latency function. This function has as rate the minimal residual rate of the servers after deducting the respective rates of the token buckets. The latency on the other hand is the sum of the latencies from the servers plus the burst terms of the cross-traffic. An interesting point is that each one of these burst terms will be paid only once at the server with the lowest residual rate. After creating said functions the algorithm takes the point-wise maximum of them, resulting in the end-to-end PMOO service curve. Here (SCHMITT; ZDARSKY; MARTINOVIC, 2008) is offered a small yet detailed example of such a computation, as well as a more detailed explanation. For the same reason as SFA, PMOO uses the virtual delay when computing the delay bound.

2.4.4 Small Example

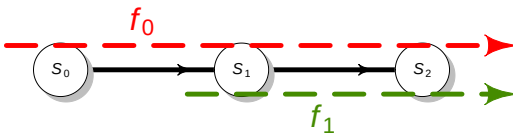
In this subsection an example of a network analysis is presented. This small example, although simple, already shows the differences between the algorithms. The topology is as follows: Three servers with $\beta_{1.0E8,0.02}$ as service curves connected in series, with two flows having $\alpha_{1.0E5,1.0E4}$ as arrival curves. The server graph just described can be seen in Figure 2.5. Each algorithm analyzes flow $f0$, and the results can be found in Table 2.1.

Even in this small configuration the differences between the algorithms are clearly

Algorithm 4: Pseudo-code for Pay Multiplexing Only Once Analysis (SCHMITT; ZDARSKY, 2006)

```
1  Function ComputeDelayBound (flow of interest f)
2      β^PMOO = ComputePMOOServiceCurve(path(f), {f})
3      return h(α_f, β^PMOO)
4  Function ComputePMOOServiceCurve(path p, flowset 𝔽)
5      eliminate rejoining interfering flows
6      merge parallel interfering flows into flowsets 𝔽_i
7      forall interfering flowsets 𝔽_i do
8          forall pred(n_{𝔽_i}) of the ingress node n_{𝔽_i} of 𝔽_i do
9              α_{𝔽_i} += ComputeOutputBound(pred(n_{𝔽_i}), 𝔽_i)
10         end
11     end
12     calculate β^PMOO
13     return β^PMOO
14 Function ComputeOutputBound(from node i, flowset 𝔽)
15     find shared path p of flows in 𝔽 starting at node i
16     call s the last node on path p (⟶ split point)
17     forall pred(s) do
18         α_split += ComputeOutputBound(pred(s), {flows to node s from
             pred(s)}∩𝔽)
19     end
20     β_p^PMOO = ComputePMOOServiceCurve(p, 𝔽)
21     return α_split ⊘ β_p^PMOO
```

Figure 2.5: A Small Server Graph



Source: Author

visible. As suggested by Subsection 2.4.1, the TFA analysis yields the worst delay bound. Similarly, as suggested in Subsection 2.4.3, the PMOO analysis yields the tightest bound. Finally, it is worth mentioning that although TFA has a large difference to the other analyses, SFA and PMOO have quite similar bounds for this example. For more comparisons between these unicast analyses, we point the reader to the work (SCHMITT; ZDARSKY; FIDLER, 2008), where all algorithms presented here were tested in topologies of different sizes and with different utilizations.

Table 2.1: Results from the Small Example Analysis for $f0$

Algorithm	*Delay(s)*
TFA	0.060661
SFA	0.060360
PMOO	0.060240

Source: Author

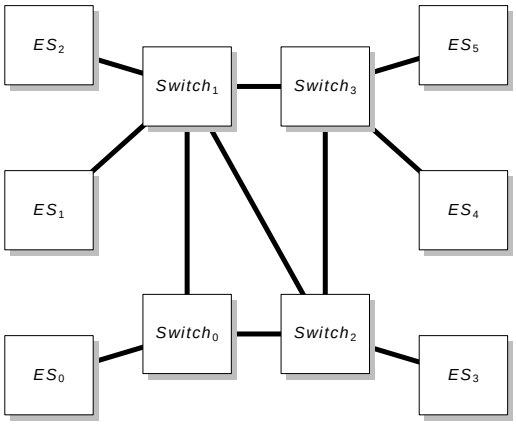
2.5 DNC Real World Application

Since Avionics Full-Duplex Networks play an important role in DNC's development, in this section we will briefly explain how they are organized. AFDX networks are a patented topology created by airbus. The specification can be seen in (TECHNICAL REPORT AERONAUTICAL RADIO INC., 2002-2005). They are composed of a dense core of switches, with dozens of End-Systems (ES) connected to them. Each one of these ESs connects to exactly one switch. The switches, however, can connect to multiple ESs. A small example of such topologies can be seen in Figure 2.6.

Due to the safety-critical applications in which they are used, these systems require performance bounds certification. The airbus A380 backbone network was one of the first industry applications of DNC.

AFDX networks have some specific characteristics besides their topology. Since they are based on the Ethernet technology, when modelling a server graph all servers have

Figure 2.6: Small AFDX Topology Example



Source: Author

the same service curve. Moreover, their End-Systems generate and receive Virtual Links (VL). These VLs are data flow abstractions, which are multicast. These type of flows differ from the unicast flows from Definition 2.2.3 only in their paths. In the next chapter we will formally define them. Their arrival curve is described using a *Maximum Frame Length* (MaxFrame) and a *Bandwidth Allocation Gap* (BAG).

- *MaxFrame*: It is the maximum size of packets generated by an ES.
- *BAG*: It limits the rate in which data can be sent by the ES. More specifically, it is the minimum interval in which two Ethernet frames can be sent.

The token bucket for the VL's arrival curve would have the *MaxFrame* as its burst term and $MaxFrame \div BAG$ as rate.

3 RELATED WORK

3.1 Study and Comparison of Delay Bounding Approaches

The literature offers many approaches for bounding flow delays in networks. This prompts the question of which analysis is more appropriated in which situation. Therefore, it is important to study and compare each approach, in order to use them in the right cases.

In (SCHMITT; ZDARSKY; FIDLER, 2008), the authors focus on DNC unicast, comparing all the analyses studied in 2.4. They use an optimization solution as a baseline, which gives an exact result. By using it as a lower bound, the analyses can be compared by how much they deviate from the exact result. Moreover, in this work they also address how PMOO can in fact underperform in some situations.

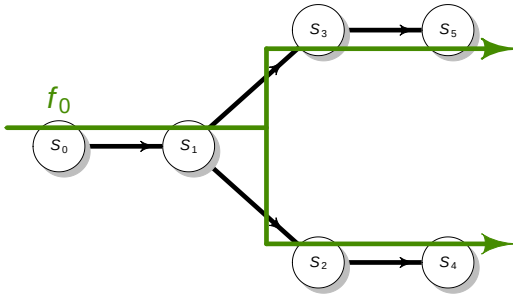
For instance, in (CHARARA et al., 2006), DNC is compared to simulations in the context of AFDX networks. The problem with DNC analyses is that they are pessimistic, so they cause over provisioning. However, because they give conservative bounds they are well suited for worst-case analysis of real time systems. Simulations on the other hand give estimates, with a certain degree of confidence. This makes them suitable for average behaviour estimation. In the same work they also explore model checking, based in timed automata. It explores all possible states of the system, and thus give an exact worst-case end to end delay. However, this solution does not scale.

In (BONDORF; GEYER, 2016), traditional DNC multicast analyses are compared to TA (BAUER; SCHARBARG; FRABOUL, 2009), FA (KEMAYO et al., 2014) and two new analyses. In this work they show that modern DNC can outperform other approaches available in the literature. However, this is a limited comparison, with only two small networks explored.

3.2 Traditional Multicast Analysis using DNC

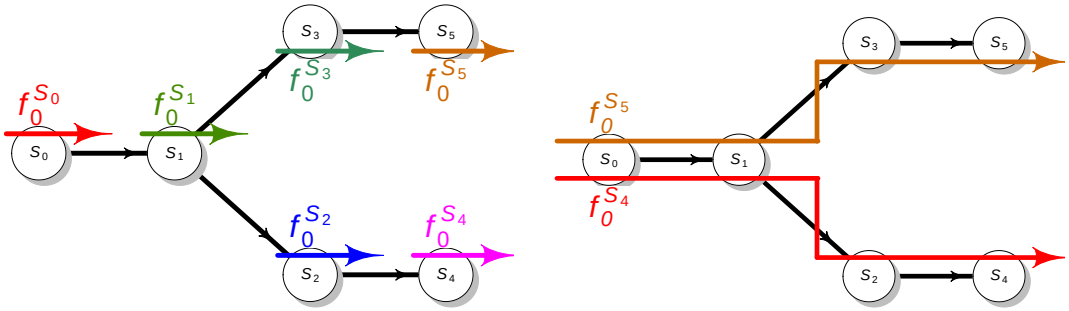
Before the algorithms introduced in (BONDORF; GEYER, 2016), DNC only offered the Multicast Total Flow Analysis and the Unicast Transformation algorithms to handle multicast flows. These two algorithms are known to be too pessimistic, yielding loose delay bounds. In order to better illustrate both algorithms, they are applied to the example topology shown in Figure 3.1.

Figure 3.1: Small Topology for Multicast Example



Source: Author

Figure 3.2: Multicast Analyses Example (Multicast TFA and UT)



(a) Multicast TFA (b) Unicast Transformation

Source: Author

The main characteristic of the algorithms here described is that the computation of the delay bounds is divided into two major steps. Before computing the bounds, it is necessary to modify the multicast flows in the network in order to better handle them. In both algorithms, the multicast flows are converted into unicast flows. Only after this conversion the delay bounds can be computed, since both methods rely heavily on the algorithms studied in Section 2.4.

3.2.1 Multicast Total Flow Analysis (Multicast TFA)

The multicast Total Flow Analysis was first introduced in (GRIEU, 2004). It is a procedure which allows the TFA algorithm presented in Subsection 2.4.1 to be applied in multicast networks. Its description can be found in Algorithm 5.

As stated before, the analysis is divided into two steps. First, a transformation of the network is made in order to apply the TFA algorithm presented in Subsection 2.4.1. This transformation breaks the multicast path of the multicast flow f into a series of unicast flows. This new flow creation can be seen from line 5 to 12.

In order to create the subflows correctly, all shared subpaths must first be identified. This is seen in line 6. Not only the subpaths must be saved, but also their relation. In other words, each subpath must also have a set representing the next subpaths after a fork and a pointer to the previous subpath from where it came from. With this, later steps can still follow the original unicast path's trajectory when necessary.

Having the shared trajectories, the algorithm continues to create a single flow for each server in each of the subpaths. Since each shared subpath appears only once, each one of the servers will have exactly one flow created from f. It is important to keep in mind that a subpath from a flow f can cross another subpath from the same flow f. This would create an interference pattern in which a flow interferes with itself; but this is left out of this high level description. The transformation described so far is illustrated in Figure 3.2a, applied to the topology in Figure 3.1. It is explicit in this figure the worst-case per server approach, which is intrinsic to TFA.

Once the unicast flows have been created, their arrival curves must be updated. This is seen from line 13 until 19. Care must be taken in order to select valid flows to update. We name such flows *independent*. Later in the next chapter this problem will be discussed in more detail. The method used for computing each output bound is the same as the one used in Algorithm 2. Starting by the root flows, the algorithm computes the flow output bound and substitutes the arrival curves of the next flows in the trajectory using this curve.

After this transformation is finished, the delay bounds can finally be computed. Since each original multicast flow has multiple sinks, a delay bound must be computed for each one of them. Such results are saved in a map from the identifier of the sink to the respective delay. The initialization of the map entry can be found in line 22. Following the original unicast path from the source server to the sink j, the algorithm sums up the individual delays of the unicast flows f^i.

3.2.2 Unicast Transformation (UT)

This simple algorithm directly converts the multicast paths of flows into individual unicast flows. Algorithm 6 shows its high level description. Similar to the Multicast TFA, the analysis of the network is divided into the same two steps. The network transformation, however, is much simpler.

For each flow f in the network, the algorithm iterates through all unicast paths in

Algorithm 5: Pseudo-code for Multicast TFA

```
1  Function PerformAnalysis(flow of interest f)
2      TransformFlows()
3      ComputeDelayBounds(f)
4  Function TransformFlows()
5      forall f ∈ all network flows do
6          identify shared subpaths of unicast flows from P_f
7          forall server s_i in shared subpath do
8              create new flow f^i with P = s_i and α = α_f
9              add f^i to the network
10         end
11         remove f from the network
12     end
13     add all root flows to flowset updatedFlows
14     while updatedFlows ≠ all flows in network do
15         find f_independent ∈ updatedFlows
16         α_new = ComputeOutputBound(Sink of f_independent, f_independent)
             according to Algorithm 2
17         update the arrival curve of flows in flowset next(f_independent) using
             α_new
18         add next(f_independent) to updatedFlows
19     end
20 Function ComputeDelayBounds(flow of interest f)
21     forall unicast paths P_{f,j} do
22         delay[j] = 0
23         forall server s_i in P_{f,j} do
24             delay[j] += ComputeDelayBound(f^i) using Algorithm 2
25         end
26     end
```

the multicast path P_f. A new unicast flow is then created for each unicast path. In Figure 3.2b this transformation is applied to the running example from Figure 3.1. These new flows have the same arrival curve as the original flow f without the necessity of updating them as in the previous algorithm. This can be seen in the Function $TransformFlows()$ in line 4.

After transforming the network, the delay bounding becomes a series of unicast ones. For each unicast path from the original flow a unicast analysis is performed using any of the algorithms presented in Section 2.4. This means that this method can fully benefit from the PBOO and PMOO property, and using the PMOO algorithm can yield tight bounds.

Algorithm 6: Pseudo-code for Unicast Transformation

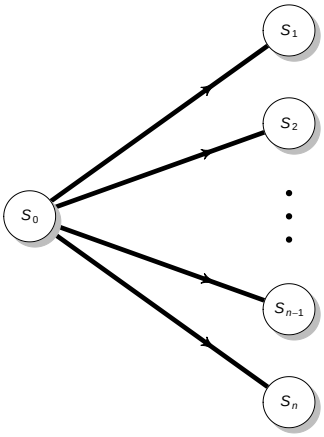
```
1  Function PerformAnalysis(flow of interest f)
2      TransformFlows()
3      ComputeDelayBounds(f)
4  Function TransformFlows()
5      forall f ∈ all network flows do
6          forall path P_{f,j} in multicast path P_f do
7              create new flow f^j with P = unicast path P_{f,j} and α = α_f
8              add flow f^j to the network
9          end
10         remove f from the network
11     end
12 Function ComputeDelayBounds(flow of interest f)
13     forall unicast paths P_{f,j} do
14         delay[j] = ComputeDelayBound(f^j) using any of the Algorithms
             presented in Section 2.4
15     end
```

3.2.2.1 UT Worst-Case

Even though it can outperform the Multicast TFA algorithm, depending on the topology in which it is executed this method can result in very loose bounds. This happens due to its overly pessimistic assumption of resources demand. Since many unicast paths $P_{f,j}$ share subpaths with each other, this individual conversion to unicast flows can overload such shared trajectories. As an example, we present the topology in Figure 3.3. However, this behavior can already be seen when comparing both transformations in Figure 3.2 from server S_0 to S_1.

Imagine a multicast flow with source S_0 and sinks from S_1 to S_n. When applying

Figure 3.3: Unicast Transformation Problem



Source: Author

the network transformation, S_0 will have n flows going through it. This shows that the pessimism added to the analysis in this case increases proportionally with the number of sinks. On the other hand, when applying the Multicast TFA, each flow appears only once in any shared subpath.

3.2.3 UT vs Multicast TFA

The topology used in this comparison is the same as in Figure 3.1. It is composed of six servers with $\beta_{1.0E8,0.02}$ as service curves and one multicast flow f_0 which has $\alpha_{1.0E5,1.0E4}$ as an arrival curve. This flow has server S_0 as a source, and forks at S_1. Servers S_4 and S_5 are the flow's sinks.

This network was analyzed using both the Multicast TFA and the Unicast Transformation. The results can be found in Table 3.1. Only the analyses for f_{0,S_4} are displayed, since they are equal to f_{0,S_5} due to the system's symmetry. The Unicast Transformation yielded both the largest and the smallest delay. This is attributed to its flexibility: it allows the use of any of the previously discussed unicast analyses. That allows it to use the PMOO analysis, which is known to be superior to TFA in most cases. However, when UT also uses TFA, the issue showed in Figure 3.3 can be seen. The network transformation applied by both analyses can be seen in Figure 3.2.

This shows that both analyses are not strictly superior to one another. For some topologies one might yield tighter bounds than the other. This further supports the results shown in (BONDORF; GEYER, 2016) for UT and Multicast TFA. We conclude that

both methods are overly pessimistic in different ways, which prompted the study of other approaches using the DNC framework.

Table 3.1: Results from the Small Multicast Example Analysis for f_{0,S_4}

Algorithm	*Delay(s)*
Multicast TFA	0.080520
UT - TFA	0.080822
UT - PMOO	0.080240

Source: Author

4 DESCRIPTION OF THE COMPARED MULTICAST ALGORITHMS

This chapter presents the two DNC multicast algorithms implemented during the development of this work. Initial results were shown in (PULIAFITO; TRIVEDI, 2018). As mentioned in that work, we decided to implement such analyses on the DiscoDNC Tool (BONDORF; SCHMITT, 2014). It is an open source tool, which implements all the DNC constructs presented in Chapter 2. It also implements some optimizations, which were not studied in this work. Such optimizations include the maximum service curve cited in Section 2.4. For a higher portability, it was developed in Java.

Both Explicit Intermediate Bounds and Multicast Feed-Forward were first presented in (BONDORF; GEYER, 2016). In the same work, the Unicast Feed-Forward Transformation is also briefly explained. Due to its simplicity, this algorithm is used as a baseline for evaluating the performance of Explicit Intermediate Bounds and Multicast Feed-Forward Analysis methods.

4.1 Explicit Intermediate Bounds (EIB)

Similar to the Unicast Feed-Forward Transformation (3.2.2) and the Multicast TFA (3.2.1), the Explicit Intermediate Bounds also requires an initial step before bounding the delay. This first step is to break all the multicast flows into smaller unicast ones. EIB's high level description can be seen in Algorithm 7. This step is shown from lines 5 to 12. Each multicast flow is substituted by a set of unicast flows, one for each common path. Figure 4.1 shows the EIB network transformation applied to the same example of Figure 3.1. The flow f_0 in this example has two unicast paths, one from server S_0 to server S_4 and one from server S_0 to server S_5. Both unicast paths share the subpath from S_0 to S_1, therefore in the transformation this is represented as one flow. The remaining subpaths have no other relation, and so they become each one unicast flow.

It is important to note, that at this moment the algorithm must somehow store the relation between the unicast flows regarding their origin in the multicast path. This could be done by creating a tree for each flow, where the root flow has pointers to the children flows and so on. The children here refer to the next flows in the network, respecting the original direction of the multicast flow. Another possibility is to create a map using a unique flow id as the key, and having pointers to the children. From this point on, the next flows in the original multicast path will be addressed as children. This information

is necessary for later steps of the algorithm.

At this point, the network has no longer multicast flows. However, the unicast flows created from the multicast flow must be differentiated from the other unicast flows originally present in the network, since they need special treatment. This can be seen in line 13. It is necessary to update their arrival curves, because each child's data flow depends on the previous path, i.e. their parent's arrival curve. At first, only the root flows have a valid arrival curve. Starting from them, the output bounds of their sinks must be computed using any of the unicast algorithms previously discussed. Care must be taken to ensure that only independent flows are used to update their children. Independent flows are defined in Definition 4.1.1. This is necessary to ensure that valid arrival curves are used to compute the output bounds.

Once the arrival curves of the new unicast flows have been updated, the network transformation is complete. The next step is to bound the delay of the flows. The delay bound of a unicast path of a multicast flow is simply the addition of the delay bounds of the unicast flows that compose such path. This can be seen in Function $ComputeDelayBounds()$ in line 21. Each multicast flow will have a delay bound for each of its sinks. This can be seen in line 23. Similar to the update of the arrival curves in the previous step, any of the discussed unicast algorithms can be used.

Definition 4.1.1 (Independent Flow)**.** A flow is said to be independent if

- it has no cross flows,
- it suffers interference from already updated flows, or
- it suffers interference from independent flows.

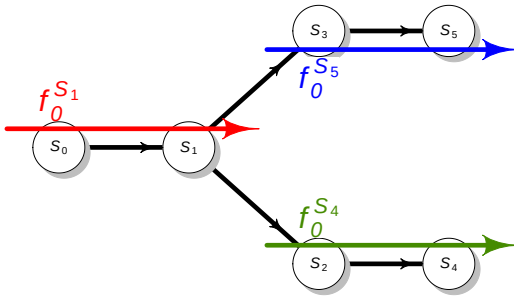
It is important to note that this is a recursive problem. In order to know if a given flow is independent, all interfering flows must be tested.

4.2 Multicast Feed-Forward (MFF)

As EIB was the generalization of the Multicast TFA analysis, the Multicast Feed-Forward approach is a generalization of the unicast analyses. Instead of creating a new analysis, in (BONDORF; GEYER, 2016) the authors generalize the steps used in the unicast algorithms. For this reason, this approach is compatible with TFA, SFA and PMOO.

The main idea is to identify the subpaths that are shared by the multicast flows

Figure 4.1: Explicit Intermediate Bounds Example



Source: Author

Algorithm 7: Pseudo-code for Explicit Intermediate Bounds

```
1  Function PerformAnalysis(flow of interest f)
2      TransformFlows()
3      ComputeDelayBounds(f)
4  Function TransformFlows()
5      forall f ∈ all network flows do
6          identify shared subpaths of unicast paths from P_f
7          forall shared subpath sub_i do
8              create new flow f^i with P = sub_i and α = α_f
9              add f^i to the network
10         end
11         remove f from the network
12     end
13     add all root flows to flowset multicastFlows
14     while multicastFlows ≠ empty do
15         find f_independent ∈ multicastFlows
16         α_new = ComputeOutputBound(Sink of f_independent, f_independent)
             according to any of the unicast algorithms
17         update the arrival curve of flows in flowset next(f_independent) using
             α_new
18         add next(f_independent) to multicastFlows
19         remove f_independent from multicastFlows
20     end
21 Function ComputeDelayBounds(flow of interest f)
22     forall unicast paths P_{f,j} do
23         delay[j] = 0
24         forall server s_i in P_{f,j} do
25             delay[j] += ComputeDelayBound(f^i) using any of the unicast
               algorithms
26         end
27     end
```

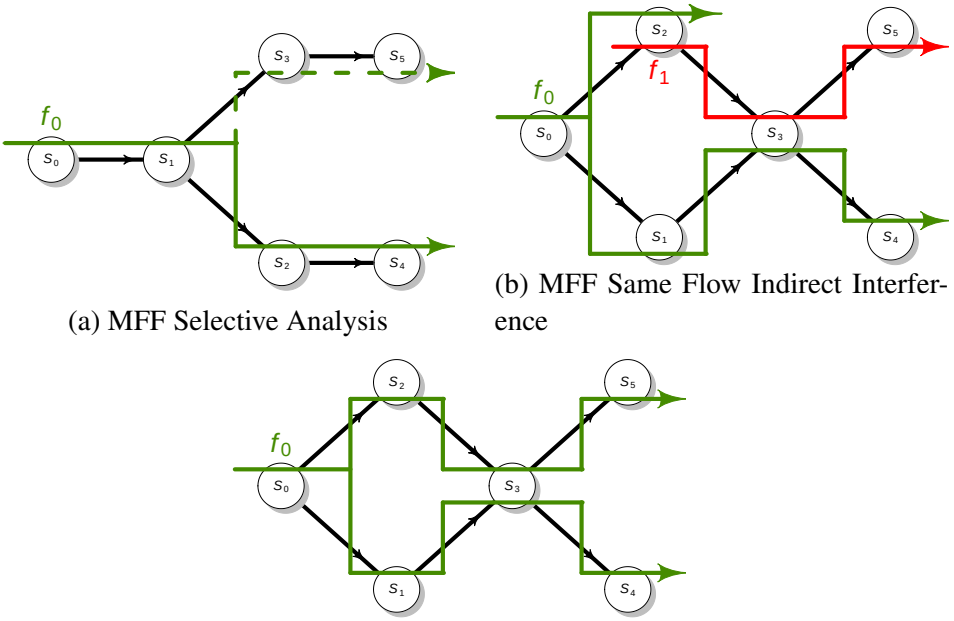
when backtracking, and count them only once. This allows for unprecedented tight bounds in multicast analyses, since they can now fully benefit from the PBOO and PMOO properties. Because SFA is known to outperform TFA, the latter will not be discussed in this work.

Since it is a generalization of the unicast analyses, the reader can refer to the Algorithms 3 and 4 in order to apply the concepts described in this section. The idea of this method is to take into account only the parts of the flows that are necessary for the analysis. A simple example of this can be seen in Figure 4.2a. From the figure, it is easy to see that the analysis in this case is simply a unicast analysis of the flow f_0 from S_0 to S_4. This is, however, a trivial example.

For more complex interference patterns, care must be taken. When backtracking, a flow can encounter itself interfering with another flow. This case can be seen in Figure 4.2b. In this example, f_0 is the flow of interest. Imagine we want to bound its delay from server S_0 to S_4. In order to do that, we must compute the output bound of flow f_1 at server s_3, in order to know how f_1 affects f_0 at that server. This requires us to backtrack f_1's path until we reach a new point of interference, this time at server S_2. However, at this point f_1 is actually suffering the interference of another part of flow f_0. Since no more interferences exist, this now can be computed as in Figure 4.2b. By considering only the subpath from S_0 to S_2 for f_0, any of the previous algorithms can be used to compute it's output bound.

In another case, a flow can fork at one point and rejoin itself at another. This would mean that a flow suffers interference from itself. An example can be seen in Figure 4.2c. Flow f_0 is affected by itself at server S_3, and in order to bound any of its delays this distinction must be made. Therefore, MFF must somehow identify this situations, and consider each subpath as a different flow until the point where the first fork occurs. A more complex example can be seen in (BONDORF; GEYER, 2016), which is also evaluated manually.

Figure 4.2: Network Modeling Example



(a) MFF Selective Analysis

(b) MFF Same Flow Indirect Interference

(c) MFF Same Flow Direct Interference

Source: Author

5 ALGORITHMS EVALUATION

In this chapter we compare the UT, EIB and MFF analyses. Because UT is one of the traditional DNC multicast analyses, it is used as a baseline for comparing the other analyses. Moreover, although it is a relatively simple method, it can still outperform other more complex ones, as shown in (BONDORF; GEYER, 2016).

This chapter is divided as follows. In the first section we have a brief discussion, in which we show how one analysis is not strictly superior to the other. This discussion focuses on the PMOO version of the analyses, since it, in general, yields the tightest bounds. The following section shows the numerical experiments made using specific topologies. The experiments were divided in two different topologies, and for each we present a small example and a more realistic one.

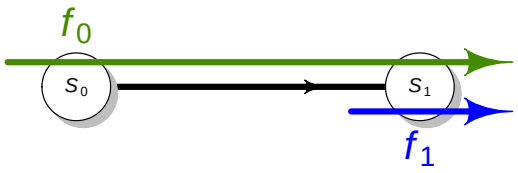
5.1 Preliminary Discussion

Although the PMOO analysis in general outperforms the SFA analysis, it is known to underperform in certain situations. In (SCHMITT; ZDARSKY; FIDLER, 2008), it is shown that, for a sink tree topology with a heterogeneous network, PMOO can have an exponential increase in its delay bounds. Heterogeneous here refers to networks in which the servers have different service curves. However, such behaviour can also be seen in much simpler networks, such as the one in Figure 5.1. In this example, assume the server S_0 to have a much smaller service curve than S_1, and that f_0 is the flow of interest.

Similarly, in a homogeneous network the same behaviour can be seen. However, as shown in Figure 5.2, a second interfering flow is necessary. In this case, the new flow f_1. The idea is that this second flow occupies the first server, creating a similar situation to the previous example. That is, an imbalance in the left over service curves of the servers in the foi's path. The foi in this example is also f_0. This type of interference pattern can be seen in the homogeneous networks used in the rest of this work.

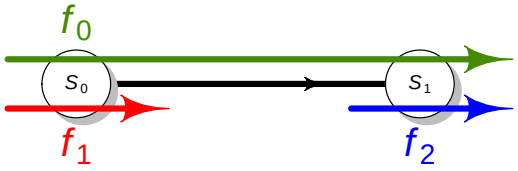
Both worst-case scenarios are also applicable to the MFF analysis, since it is a generalization of both PMOO and SFA to multicast flows. It is also true that because the EIB and the UT analyses use the unicast analyses, this is also applicable to them. However, the MFF and the UT analyses have the same PMOO worst-case scenario, whereas the EIB has a slightly different one. This means that even though the MFF analysis can have the PMOO property working on larger sequences of servers, there are specific sce-

Figure 5.1: Worst-Case Topology for PMOO - Heterogeneous Network



Source: Author

Figure 5.2: Worst-Case Topology for PMOO - Homogeneous Network



Source: Author

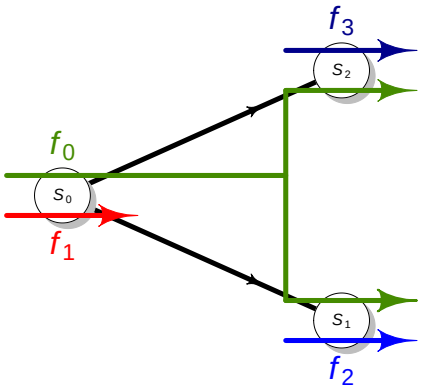
narios in which the EIB can in fact outperform it. Examples of such scenarios are shown in Figure 5.3. In all cases f_0 is the flow of interest.

In 5.3a, we have a three server system. For both MFF and UT, this case translates to the same behaviour as described above when discussing Figure 5.2. When bounding the delay from S_0 to S_2, MFF simply ignores the other subpath from S_0 to S_1. Therefore, this becomes the exact system shown before. Similarly, UT would break f_0 into two unicast flows. When bounding the delay from S_0 to S_1, the unbalance would remain, and the same behaviour would be observed.

However, this scenario does not affect EIB. Because it would break f_0 at servers S_0, S_1, and S_2, the PMOO analysis would not be executed over the unbalanced path. EIB would compute the delay at S_0, and sum it to the delay computed at S_2 individually. In Figure 5.3b, on the other hand, we show a four server system in which EIB suffers from the same behavior. In this case, f_0 would be broken into three unicast flows, one with path S_0, one from S_2 to S_4, and one from S_1 to S_3. When computing the delay from S_0 to S_4, EIB would bound the delay at S_0, and sum it to the bounded delay of the following subflow. However, this other subflow would have exactly the scenario described in Figure 5.2. From this point on, this type of scenario will be addressed as PMOO worst-case.

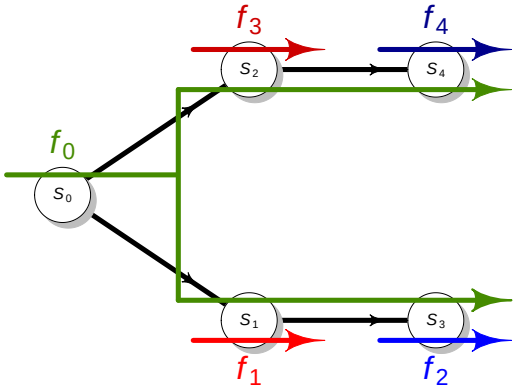
Figure 5.3: Multicast PMOO Worst-Cases

(a) Worst-Case Topology for MFF and UT



Source: Author

(b) Worst-Case Topology for EIB



Source: Author

Source: Author

5.2 Experimental Evaluation

The experiments described in this section were executed in a Supermicro X7DVL server, with a Intel Xeon E5420 CPU amd 12GB RAM. For these experiments we used two topology generators.

The first, is an AFDX generator, which creates topologies similar to the one presented in 2.5. This generator was already implemented in the DiscoDNC Tool. However, it was a limited tool. We extended it in order to create more realistic networks, and also implemented multicast support.

For the second experiment we used the aSHIIP generator (TOMASIK; WEISSER, 2010), which was ported to the DiscoDNC Tool in (HAMSCHER, 2018). Since it was implemented with only unicast flows in mind, we also generalized it to support multicast flows. This generator offers a vast series of network models, of which we decided to use the General Linear Preference(GLP) model. The GLP model was presented in (BU; TOWSLEY, 2002), and aims to model an Internet-like behaviour.

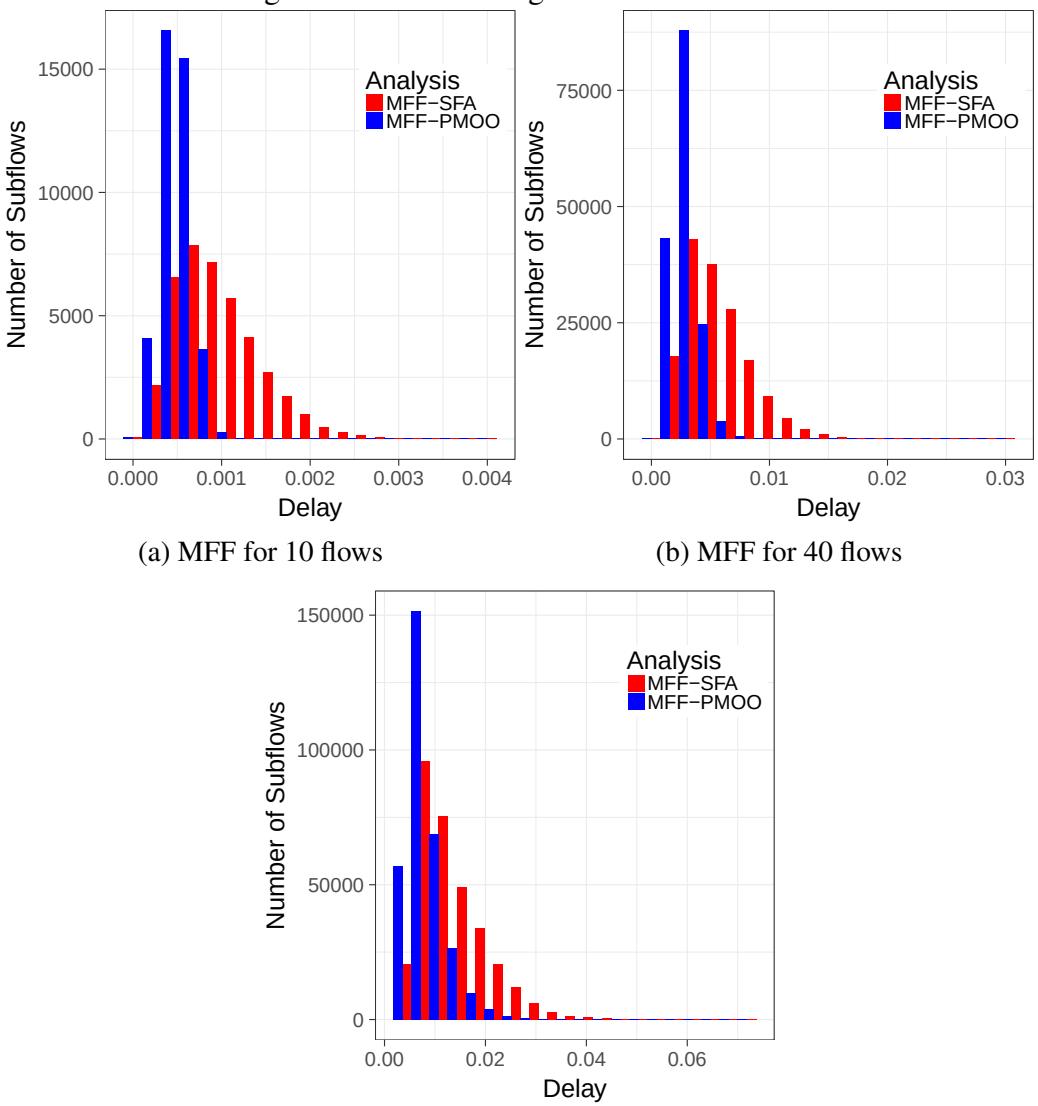
Since the networks are random, multiple networks were used in order to get a more statistically relevant idea of the behaviour of each analysis. In order to better study the results, we refer from here on to subflows as being the unicast part of a multicast flow, from the source to one of the sinks. All the delay bounds presented in this section are given in seconds.

5.2.1 AFDX Topology

The first AFDX experiment used a small topology, with 10 End-Systems and 6 switches. Each switch could connect to 1 or 4 switches, and to 1 or 3 ESs. Since AFDX networks use Ethernet technology, we modeled all service curves as rate latency curves, with rate $100e^6$ and 0 latency. Each VL had 4 sinks, with BAG values randomly selected varying from 1 to 128 milliseconds (in powers of 2), and the max frame varying from 500 to 1200 bytes. We varied the number of VLs in order to verify how the algorithms behaved to differently occupied networks. The sizes chosen were 10, 40 and 80 VLs. For each size we created 1000 networks. The VL paths were chosen randomly. This can cause some instances to be invalid, i.e., the utilization is so high that the bounded delay is infinite. For this experiment we created a 1000 networks, which were valid to all analyses.

In Figure 5.4, we show the results for the MFF analysis for the small AFDX topology. These histograms consider all subflows from the 1000 generated networks. They aim to give a general idea of the average distribution of delays bounded by each analysis. This figure is further divided into three histograms, one for each network size. Figure 5.4a shows the distribution for the networks with only 10 flows, Figure 5.4b shows the distribution for 40 flows, and Figure 5.4c shows the distribution for 80 flows. The first important point to note in these histograms is the difference between the PMOO version (in blue) and the SFA version (in red). For all three histograms, we have the PMOO concentrated in the left portion of the histograms. This shows that the PMOO version of the analysis is yielding tighter bounds than SFA, which occupies bins ranging from the lesser values of the histograms to the larger ones. This is true for all three histograms of Figure 5.4.

Figure 5.4: MFF Histograms - Small AFDX



(a) MFF for 10 flows

(b) MFF for 40 flows

(c) MFF for 80 flows

Source: Author

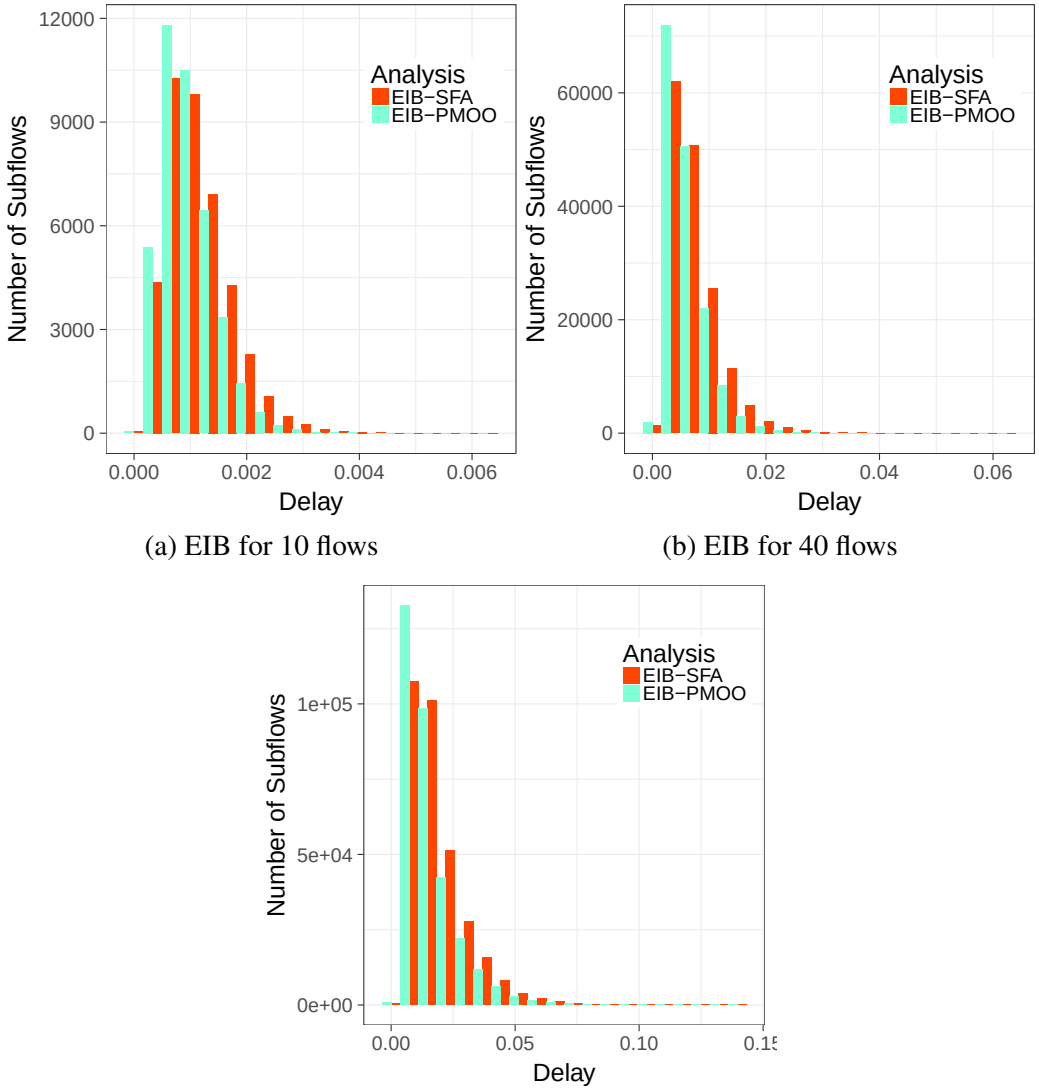
Figure 5.5 shows the results for the EIB analysis for the same small AFDX topologies. As with the previous results, Figure 5.5a gives the delay distribution bounded by the analysis for all networks with 10 flows, Figure 5.5b the delays bounded for all networks with 40 flows, and Figure 5.5c the delays for the networks with 80 flows. As with the MFF results, we point the reader to the left portion of the histograms. The PMOO portion (in light blue) has much larger bins in the left portion of the histograms than SFA (in light red), meaning that the delays bounded by it are tighter.

However, by looking at the maximal and minimal values of the histogram bins, we can see a large difference between MFF and EIB. The maximal value shown by the MFF histogram of Figure 5.4a is 0.004 seconds, whereas in EIB Figure 5.5a is 0.006 seconds.

This means that the EIB analysis for the same networks as MFF has yield larger delays, which means that this analysis is adding more pessimism to the results.

When looking only at the PMOO values, a similar behaviour can be seen. For the MFF Figure 5.4a, the larger bins of PMOO are around 0.001 seconds. On the other hand, for the EIB Figure 5.5a, the larger bins of PMOO are over 0.003 seconds. This is also seen in the histograms of the networks with 40 flows and 80 flows.

Figure 5.5: EIB Histograms - Small AFDX



(a) EIB for 10 flows

(b) EIB for 40 flows

(c) EIB for 80 flows

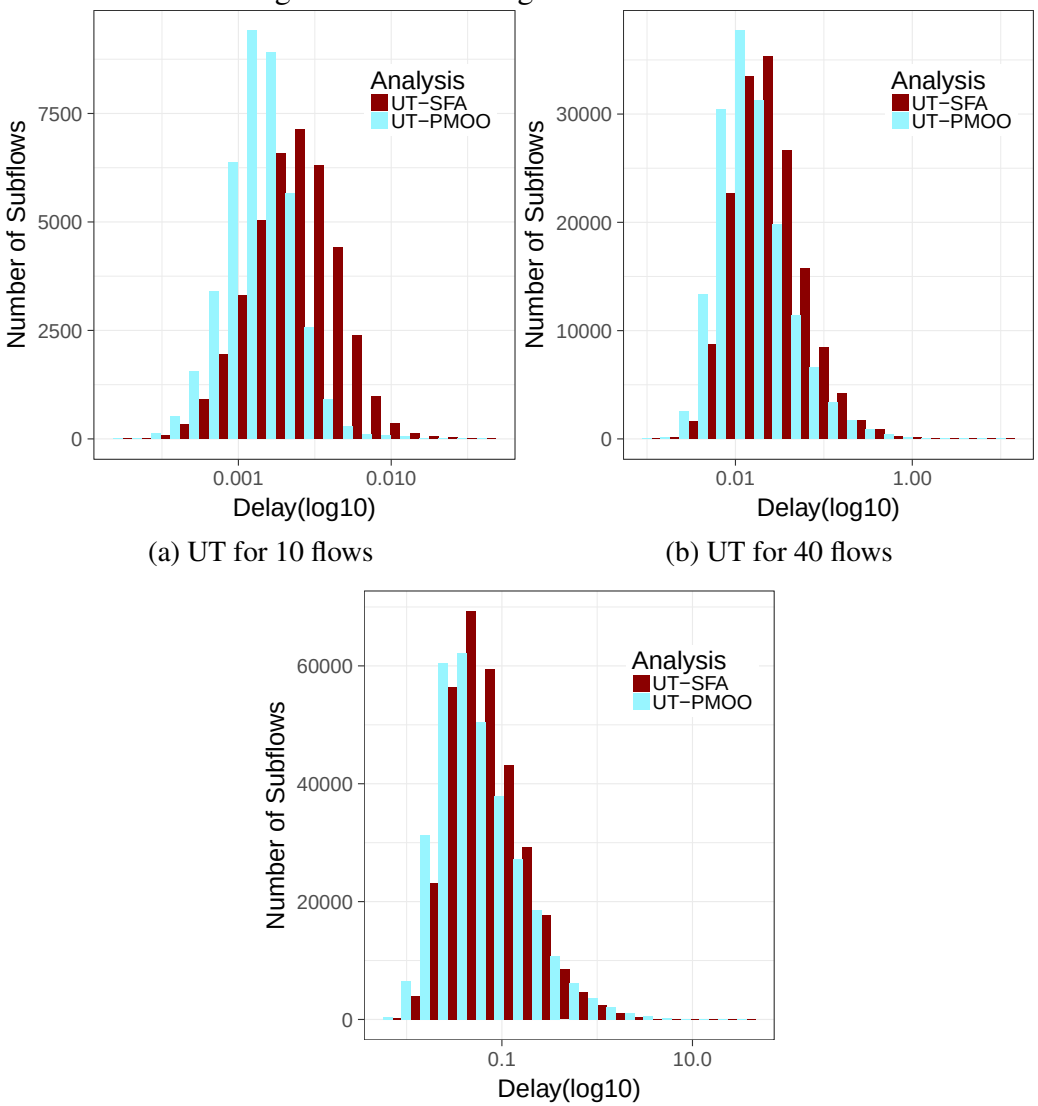
Source: Author

Figure 5.6 shows the results for the UT analysis, one of the traditional multicast analyses for DNC. Because this analysis had such large maximum bin values, we display the delay values using a log10 scale. These results are shown in the same fashion as the previous, with Figure 5.6a showing the results for the networks with 10 flows, Figure 5.6b

the results for 40 flows, and Figure 5.6c the results for 80 flows. This analysis yield the largest delays bounds of the analyses. This is evidenced by the large variation in the bin values, as mentioned before.

This behaviour can be explained by the UT worst-case shown in Section 3.2.2.1, where the over pessimistic assumption of resource demand of UT is described. This AFDX networks were small, which led to possible multiple flows sharing the same servers. This affects all analyses, as seen by the larger histogram intervals of Figures 5.4c, 5.5c when compared to the interval of the smaller networks, in Figures 5.4a, 5.5a. Even so, UT suffers much more from this increase in the number of flows, because each new multicast flow added to the network translates to an addition of as many unicast flows as the number of sinks of that multicast flow in the UT transformed network. For instance, since these topologies used multicast flows with 4 sinks, each new flow added to the network would mean 4 new flows in one server for the UT analysis.

Figure 5.6: UT Histograms - Small AFDX



(a) UT for 10 flows

(b) UT for 40 flows

(c) UT for 80 flows

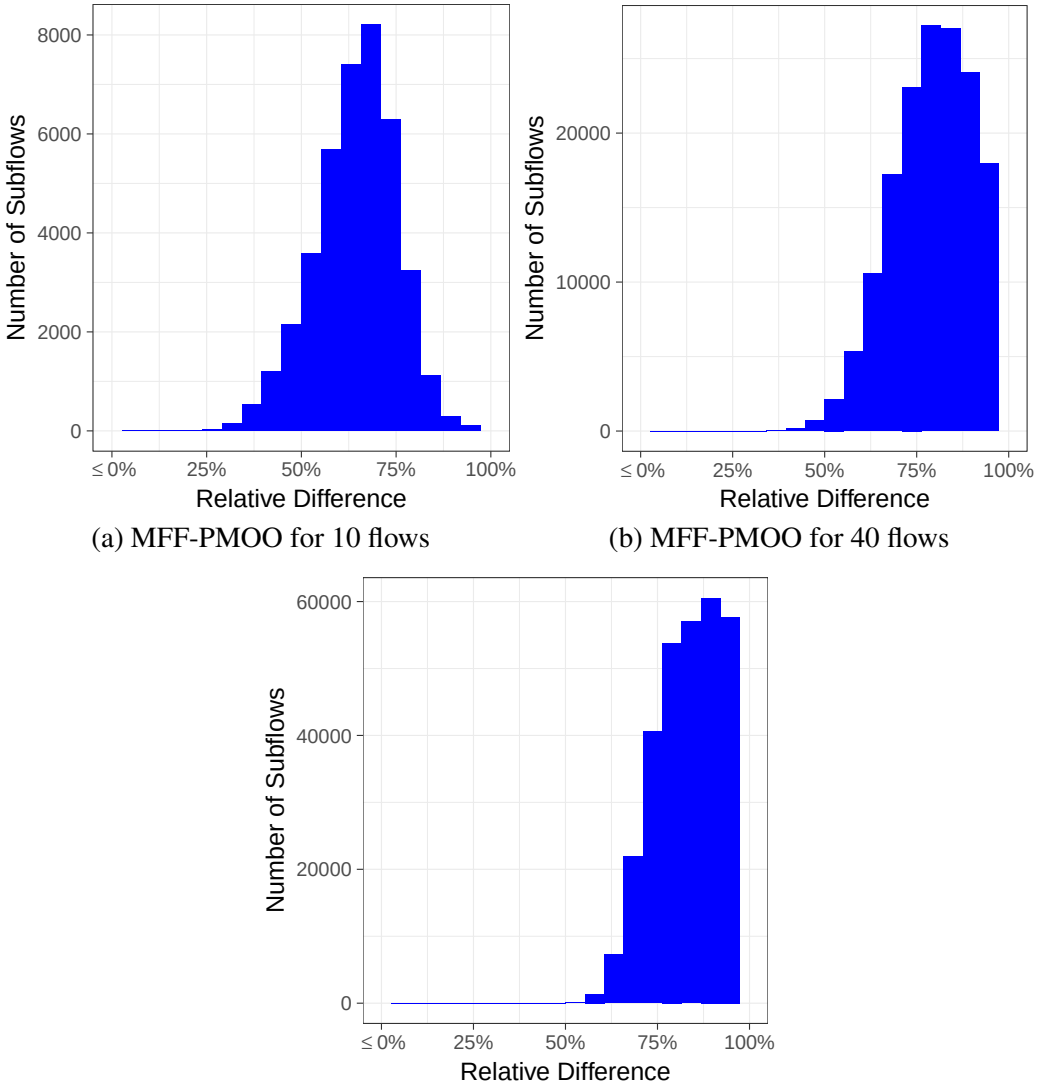
Source: Author

With this overly pessimistic behaviour in mind, we used the UT results of each subflow and computed the relative difference from results of MFF and EIB. Since for MFF and EIB the PMOO versions yield the tightest bounds, we compute this difference using the values bounded by their version of the analysis. This relative difference gives us how much the new analyses improved in respect to this traditional analysis. The relative difference of MFF to UT is seen in Figures 5.7, whereas the relative difference of EIB to UT is seen in Figures 5.8. These figures are divided as the previous results, with one histogram for each network size. Results where MFF and EIB outperform UT will be represented by values in the rightmost portion of the relative difference histograms. However, it is possible that UT can equally perform or even outperform the analyses, as

shown in the preliminary discussion at the beginning of this chapter (Section 5.1). Such values would be represented by the one leftmost bin in the histograms.

In Figures 5.7 we have the relative difference of MFF with respect to UT. It is worth noting that for all network sizes, the leftmost bin, which represents results that were equal or in which UT outperformed MFF, is empty. This means that for the experiments executed at this moment, MFF always outperformed UT. Moreover, in Figure 5.7a the relative difference is ranging around 65%. As the number of flows increases, the distribution of bins tends more and more to the right, up to the point in Figure 5.7c where they all range from over 50% to 100%. This extreme value of 100% can be attributed to the extreme pessimism shown by some bounds given by UT. In the histogram of Figure 5.6c, the largest values range over 10 seconds. In fact, when looking at the data, the largest delay for UT for those networks was of 30 seconds. This is an extremely pessimistic bound. The MFF however, for the same flow yield a delay bound of only 0.01 seconds. Because of such a large difference between the results, we see this 100% bin being so large for the networks of size 80 flows.

Figure 5.7: Relative Difference Histograms for MFF-PMOO - Small AFDX



(a) MFF-PMOO for 10 flows

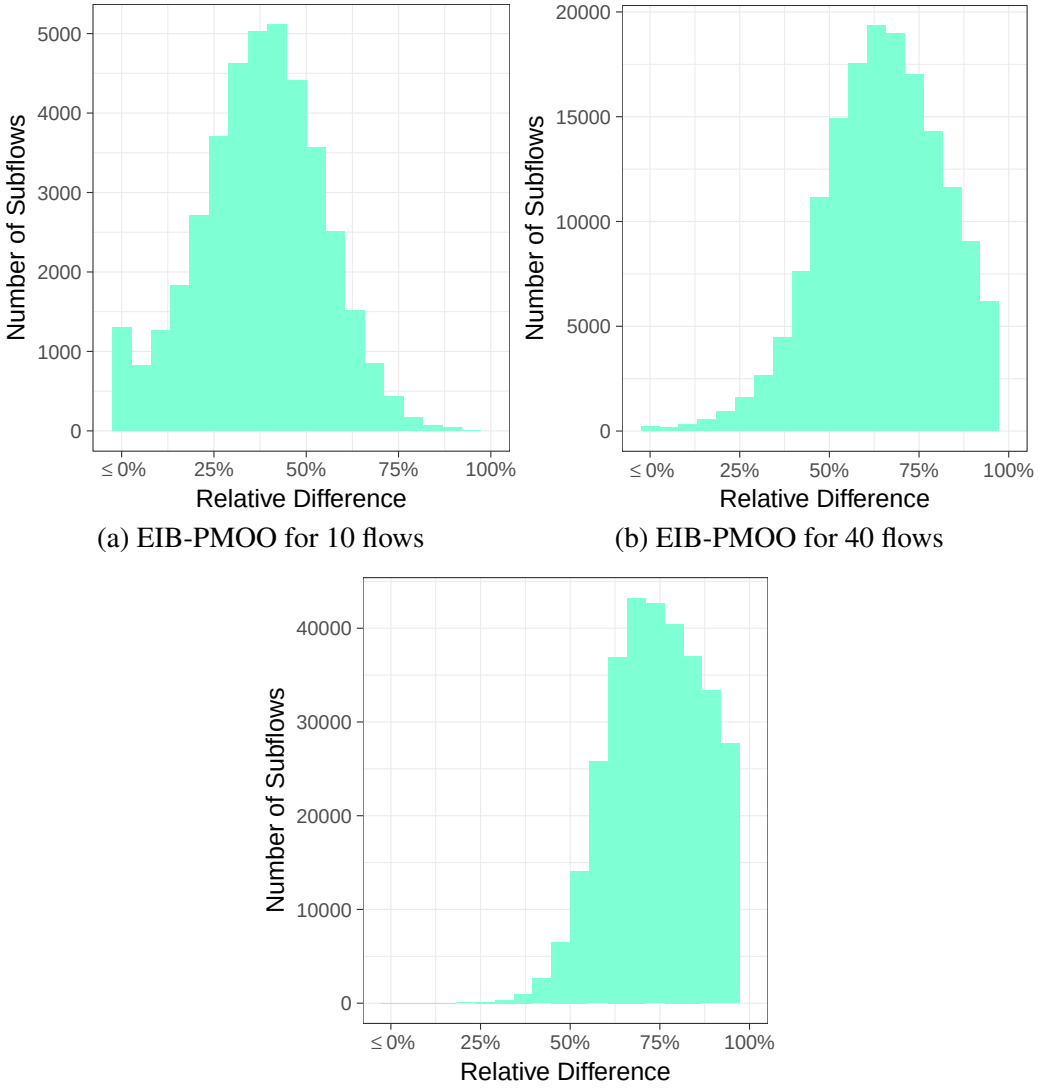
(b) MFF-PMOO for 40 flows

(c) MFF-PMOO for 80 flows

Source: Author

The relative difference histograms from EIB to UT can be found in Figure 5.8. When looking at these histograms, we see a similar behaviour to the MFF histograms. As the number of flows in the network increase, the distribution of flows tend more to the rightmost bins, showing that EIB also scales better than UT. However, at the histogram of Figure 5.8a the first bin has over 1000 subflows. This means that, for that network size, UT outperformed EIB in these cases. Since that is the network with least flows, UT's pessimistic resource demand does not affect it as much. Moreover, UT can fully benefit from the PMOO property, wheres EIB is forced to work with less servers due to its inherent break of multicast flows into smaller unicast ones. Even so, for 40 flows in Figure 5.8b this first bin is much less pronounced, and in Figure 5.8c it is barely visible.

Figure 5.8: Relative Difference Histograms for EIB-PMOO - Small AFDX



(a) EIB-PMOO for 10 flows

(b) EIB-PMOO for 40 flows

(c) EIB-PMOO for 80 flows

Source: Author

In Table 5.1 we present the percentage of subflows for each topology and each analysis in which the delay bound given by SFA is lower than by PMOO. These are cases in which the PMOO worst-cases shown in the preliminary discussion are most likely to be happening. Looking at the table, we see that as the number of flows are increased, the number of subflows affected by this behaviour increases for UT and EIB. On the other hand, for MFF the percentage ranges from 5% to 2%, showing no relation to the utilization of the network. Although UT and MFF have the same PMOO worst-case, UT also suffers from its pessimistic server utilization, which is very relevant in this small topology, as shown by all data collected so far. As a result, we see that after a given number of flows, analyses such as UT can have a large number of subflows in which SFA outperforms

PMOO. In such networks, it might be more useful to use a SFA version of these analyses. This also has implications to unicast analyses in general. If we ignore the semantic of the multicast flows in a UT transformed network, it is in fact a unicast network. Therefore, as we increase the number of flows in this type of highly utilized network, it seems that SFA can in fact be more suitable than PMOO.

Table 5.1: Percentage of Subflows where SFA Outperforms PMOO - Small AFDX

Algorithm	*10 Flows*	*40 Flows*	*80 Flows*
UT	8.7%	28.1%	46.8%
EIB	15.1%	23.4%	25.9%
MFF	5.2%	2.3%	3.9%

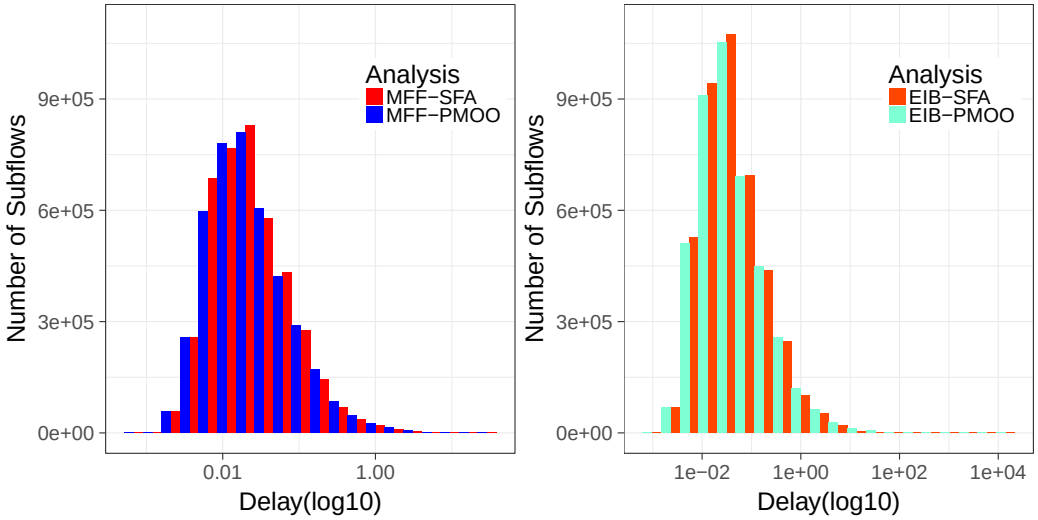
Source: Author

The second experiment for AFDX networks used an A350-like topology. We created a network similar to the one described in (HOTESCU et al., 2017), which is much more realistic than the one in the previous experiment. However, due to the complexity of the analyses we used a reduced number of VLs. For this experiment, we used only 650 VLs per network, with BAG values ranging from 2 to 128 milliseconds (in powers of 2), and max frames values from 300 to 500 bytes. Each VL also had from 1 to 12 sinks. Because UT was clearly inferior to the other analyses, in this experiment we executed only EIB and MFF. This allowed for the creation and study of networks with much higher server utilizations, since they were not limited by the over pessimism of UT.

The histograms of Figure 5.9 show that for this topology the MFF again outperforms EIB. The difference, however, is much more pronounced in this case, with EIB having extremely pessimistic bounds. In the histogram of Figure 5.9b, the maximum value of the largest bins are over 1000 seconds, whereas the values in the MFF histogram of Figure 5.9a are much less pessimistic. Because there was a large variation of delay bounds, specially for EIB, the delays are shown in a log10 scale. This is a similar behaviour to the one seen in the UT analysis. We believe that this happens because these networks have a much higher server utilization than the previous experiments. Because EIB is more pessimistic than MFF, it creates unrealistic results when analysing these extreme cases.

A difference from the previous experiment is that, in this case, the SFA versions of both analyses outperformed the PMOO versions in the majority of subflows, as shown in Table 5.2. This can again be attributed to the higher utilization of the network servers. This shows that for these extreme cases, the SFA version might in fact be better suitable. Moreover, in this case the percentage of subflows suffering from this PMOO worst-

Figure 5.9: Histograms for MFF and EIB - Large AFDX



(a) MFF for A350 (b) EIB for A350

Source: Author

case was higher for MFF. This could be explained by the MFF analysis sharing the same PMOO worst-case than UT. Because UT was so affected by its worst-case, it is possible that the networks generated and considered valid were the ones where this case was not as pronounced as in this one.

Table 5.2: Percentage of Subflows where SFA Outperforms PMOO - Large AFDX

Algorithm	*Percentage of Flows*
UT	—
EIB	63.3%
MFF	78.6%

Source: Author

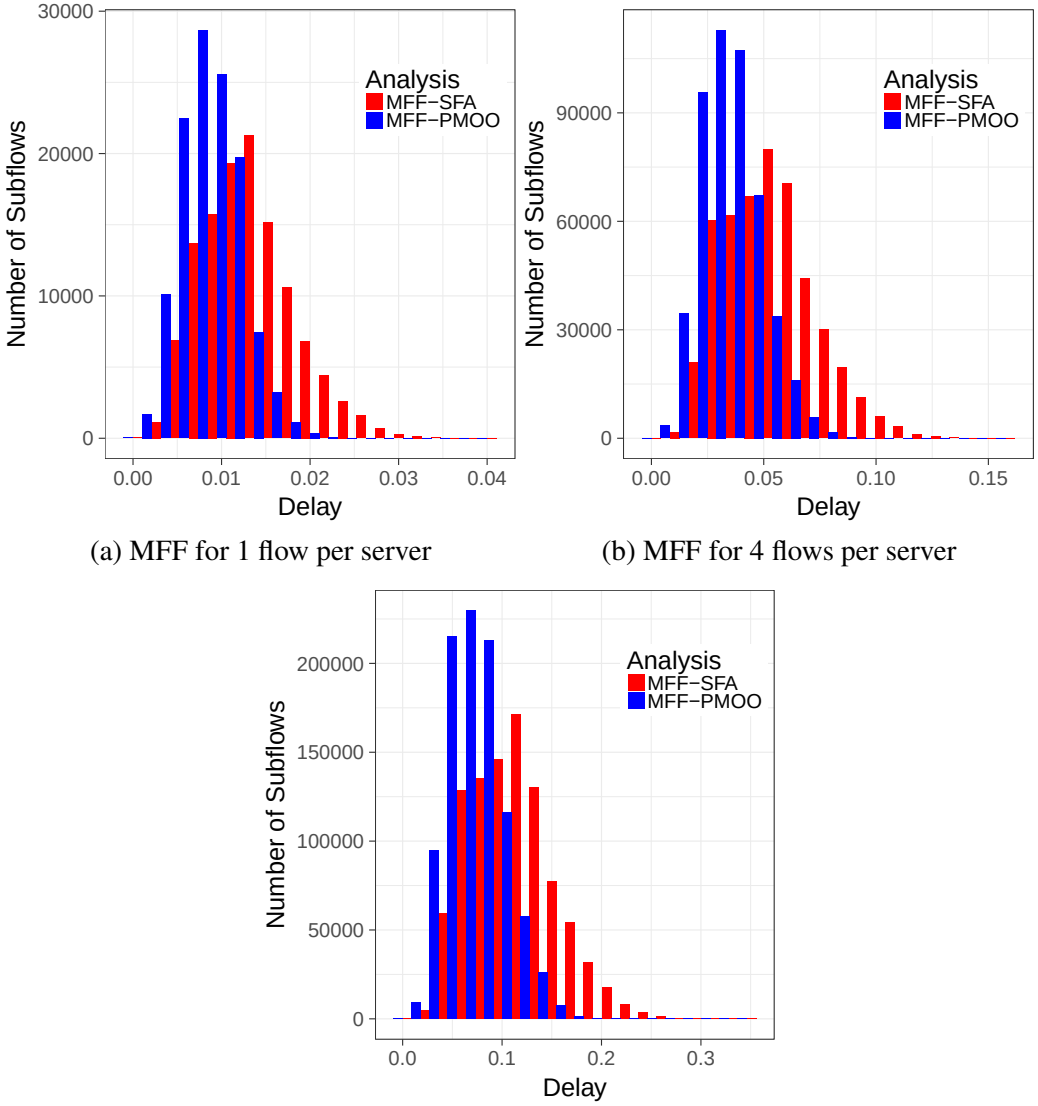
5.2.2 GLP Topology

Similar to the previous experiment, we created two classes of topologies. One smaller, in which we vary the number of flows, and a larger one, which tends to be more realistic. We followed a similar configuration as in (TOMASIK; WEISSER, 2010). This, according to the same work, creates networks with an Internet-like topology. It is important to note that in this experiment we define the sizes of the networks not as the number of flows, but the number of flows per server. All flows generated are still completely random, with no control over their source and sinks. However, this scales the number of flows depending on the number of servers in the server graph.

When compared to the previous experiment, we see that these topologies are quite different. AFDX networks are limited, in the sense that the interference pattern is limited to its switch core. In GLP networks, this interference pattern can happen in any part of the network, with a random distribution of flows. Another important distinction to the previous experiment are the service curves of the servers. In this topology the servers have a rate of $1e^9$ and 0 latency. Moreover, the VLs have fixed arrival curves with rate $5e^6$ and burst $5e^6$.

In Figure 5.10, we show the results for the MFF analysis in the first GLP experiment. In a similar fashion as the first AFDX experiment, we show one histogram for each size of network. Figure 5.10a shows the histogram of all networks with one flow per server, Figure 5.10b the histogram for 4 flows per server, and Figure 5.10c shows the histogram for 8 flows per server. When looking at the histograms, one can see that the PMOO version of the analysis (in blue) again outperforms the SFA version (in red), since it occupies mostly the lower bins in the left portion of the histograms. The SFA on the other hand, has its values more spread over the whole histogram. Although this is a similar result to what was shown in the AFDX histograms of Figure 5.4, in that experiment the PMOO bins where much larger than in this one. This means that for the GLP experiment the difference between PMOO and SFA for this size of networks is not as pronounced as in AFDX.

Figure 5.10: MFF Histograms - Small GLP



(a) MFF for 1 flow per server

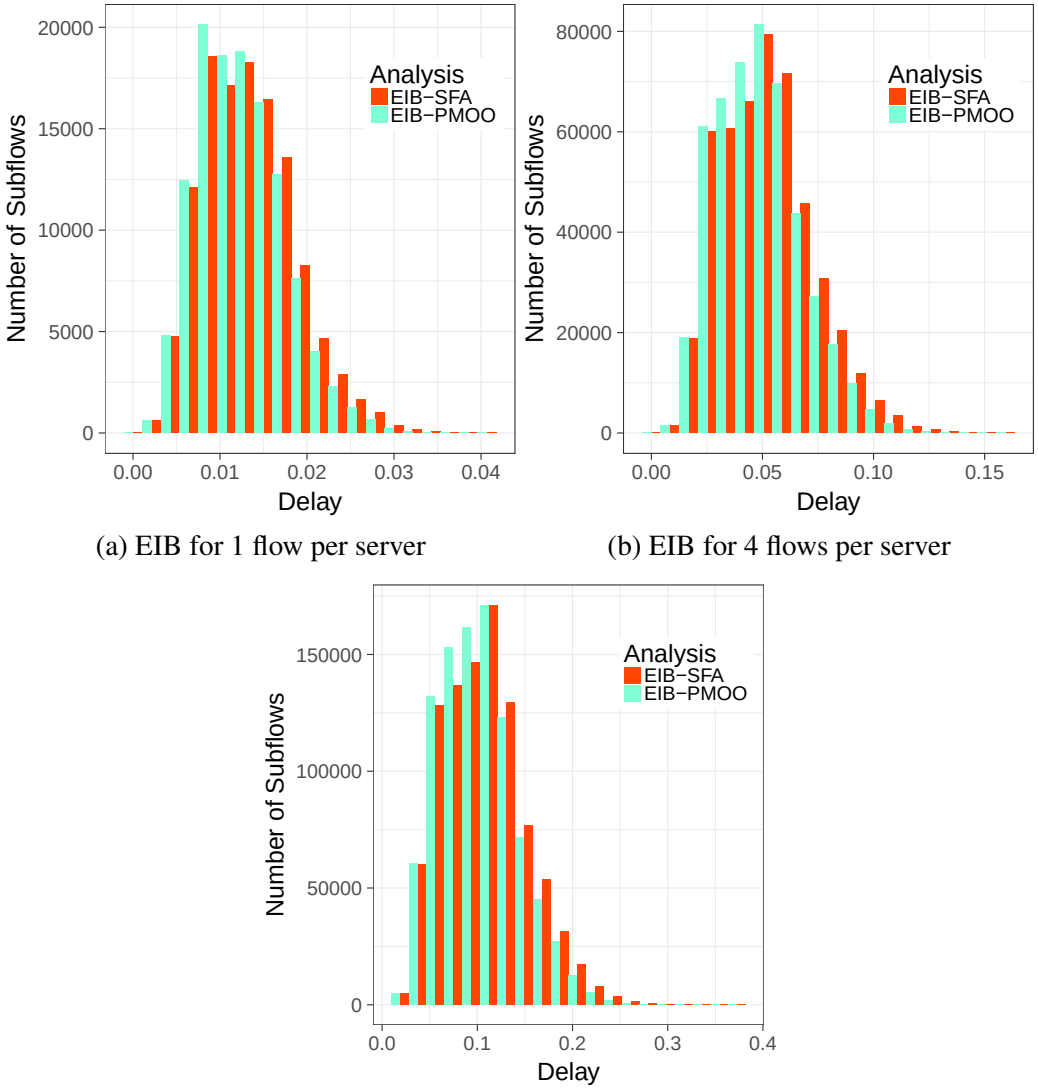
(b) MFF for 4 flows per server

(c) MFF for 8 flows per server

Source: Author

Figure 5.11 shows the histograms for EIB. Again, we show one histogram for each network size. We see that the PMOO version in the histograms of Figure 5.11 (in light blue) of the analysis follows a similar pattern to the SFA (light red). Although PMOO still occupies more bins in the left portion of the histograms, as was with the MFF for GLP the difference between the analysis versions is smaller than in the AFDX experiment. Moreover, in the histograms of Figure 5.10 this difference between PMOO and SFA was more pronounced than in the EIB histograms of Figure 5.11

Figure 5.11: EIB Histograms - Small GLP



(a) EIB for 1 flow per server

(b) EIB for 4 flows per server

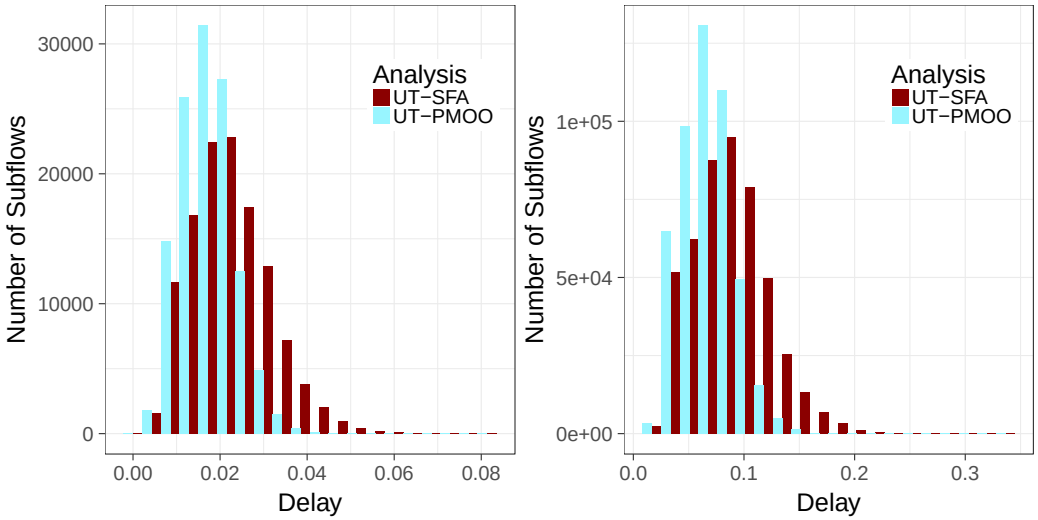
(c) EIB for 8 flows per server

Source: Author

In Figure 5.12, we show the results for the UT analysis for the same networks. When comparing these histograms to the ones of MFF and EIB, we see that for this experiment the difference between the analyses was not as pronounced as in the AFDX experiment. While the MFF histogram in Figure 5.10a shows as a maximal bin value a delay bound of 0.04 seconds, for the same networks the EIB shows in Figure 5.11a the same delay bound, and UT a much larger one in Figure 5.12a with 0.08 seconds. Although larger than the others, this is not the extreme pessimism shown in the small AFDX experiments. This can be attributed to the much higher service rate of the servers in the GLP networks. This mitigates the UT worst-case mentioned in the preliminary discussion of the analyses. Even so, UT shows overall delay bounds that are more pessimistic than

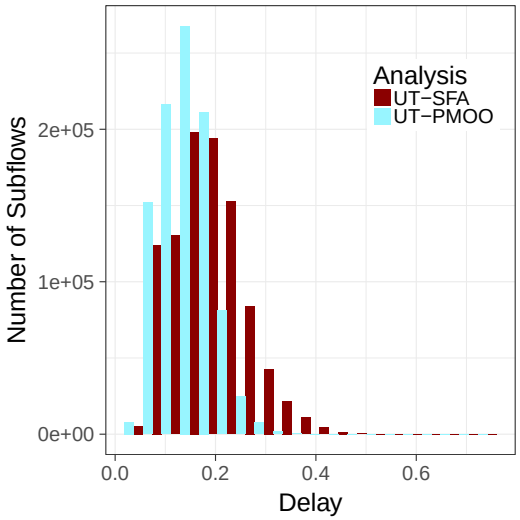
MFF and EIB. Therefore, we show in Figure 5.13 the relative difference from MFF with respect to UT, and in Figure 5.14 the relative difference of EIB with respect to UT.

Figure 5.12: UT Histograms - Small GLP



(a) UT for 1 flow per server

(b) UT for 4 flows per server

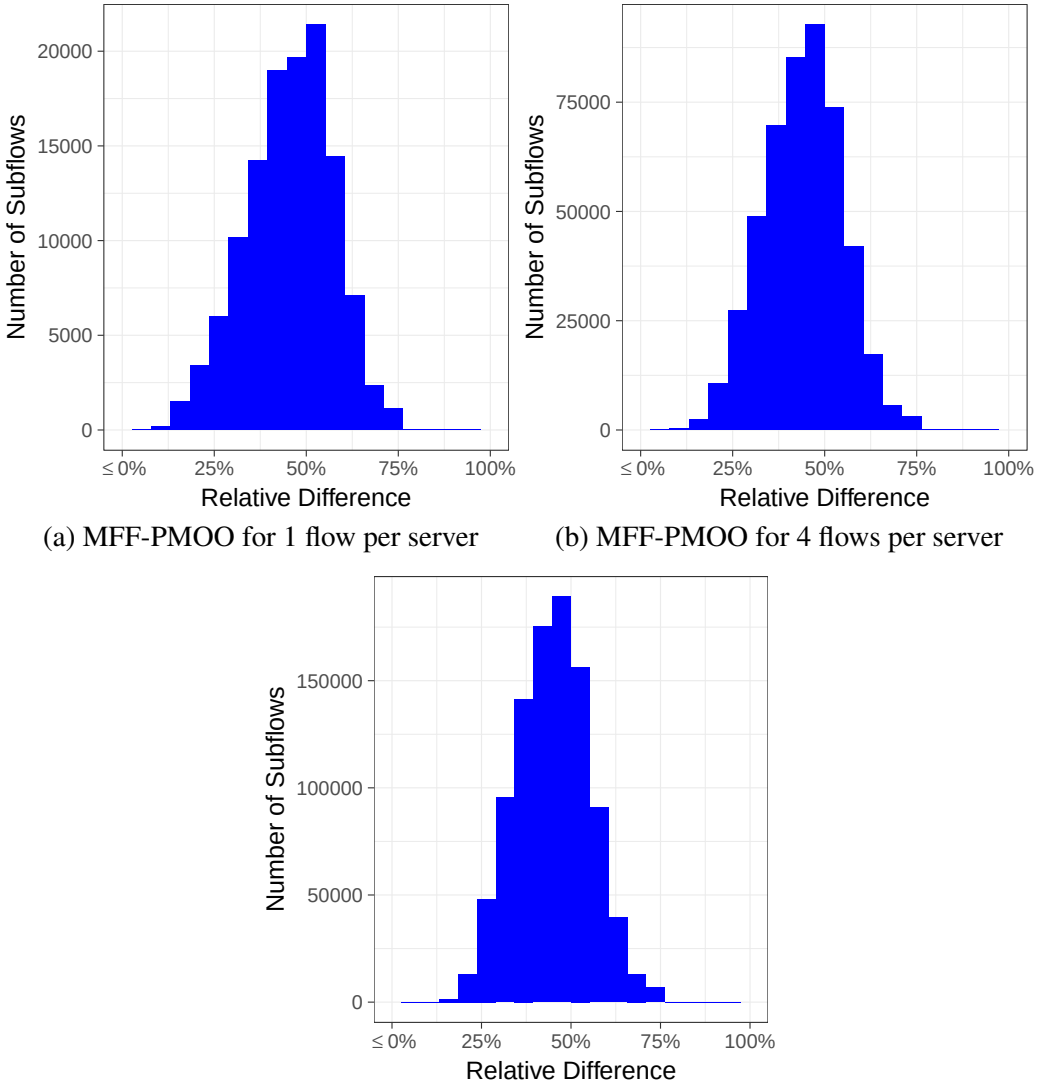


(c) UT for 8 flows per server

Source: Author

The relative difference of MFF with respect to UT is seen in Figure 5.13, for all network sizes. The histograms of Figure 5.13a, Figure 5.13b and Figure 5.13c show all the same distribution. This means that as the network sizes are increased, the relative difference is not changing considerably. As a result, we conclude that both analyses are scaling similarly. This behaviour can also be seen for EIB.

Figure 5.13: Relative Difference Histograms for MFF-PMOO - Small GLP



(a) MFF-PMOO for 1 flow per server

(b) MFF-PMOO for 4 flows per server

(c) MFF-PMOO for 8 flows per server

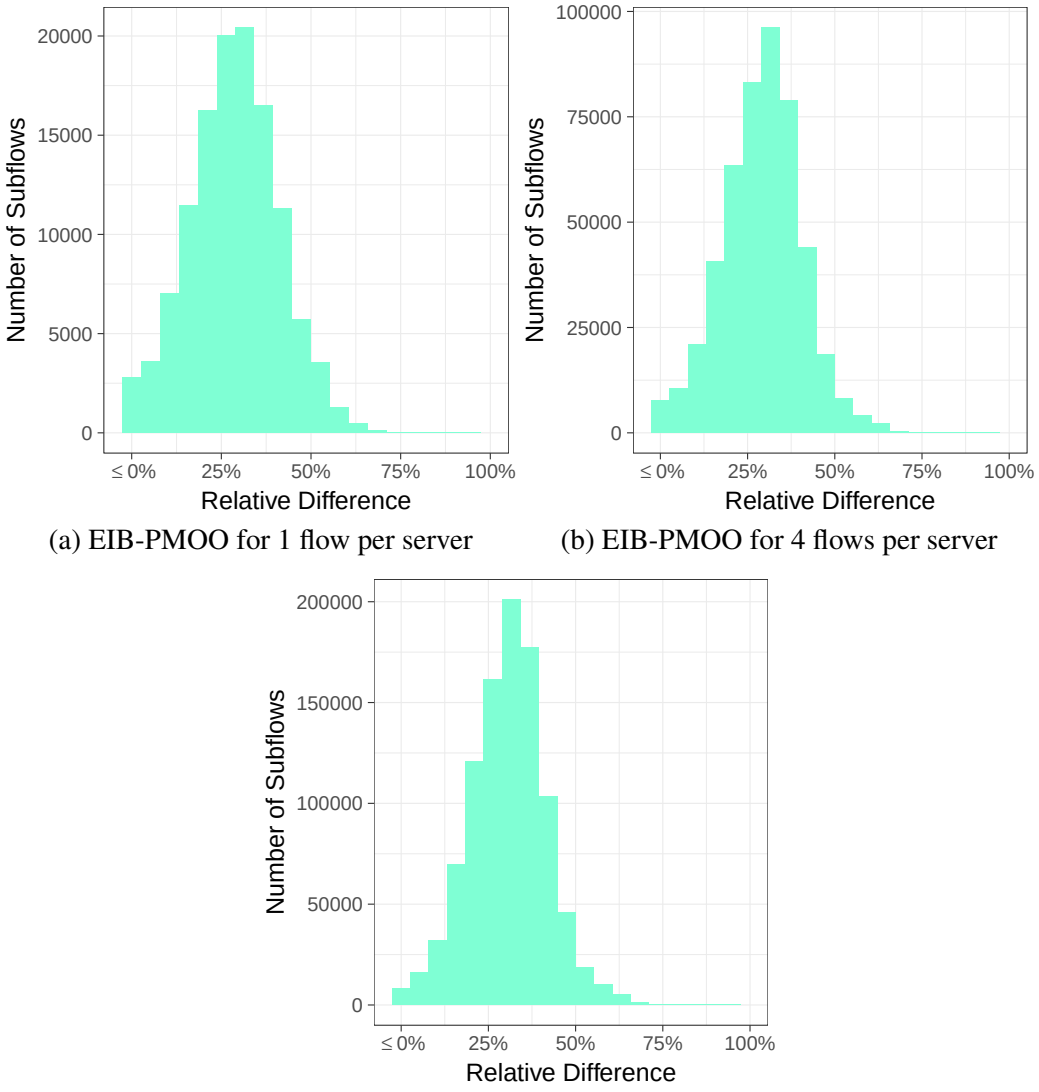
Source: Author

Figure 5.14 shows the relative difference of EIB with respect to UT. As with the MFF analysis, EIB seems to maintain the same distribution as the size of the network increases. This indicates that EIB is also scaling similarly as the other analyses.

Even though all analyses seem to scale equally, we see that MFF has a lesser overall pessimism then EIB. This is seen by the position in which the curve in the histograms of the relative differences are. For all histograms of Figure 5.13, the relative difference of MFF with respect to UT is around 50%. On the other hand, for EIB in the histograms of Figure 5.14 the relative difference is constantly around 25%. Therefore, we conclude that MFF does outperform both EIB and UT for these experiments.

In Table 5.3, we show the percentage of subflows in which SFA outperforms

Figure 5.14: Relative Difference Histograms for EIB-PMOO - Small GLP



(a) EIB-PMOO for 1 flow per server

(b) EIB-PMOO for 4 flows per server

(c) EIB-PMOO for 8 flows per server

Source: Author

PMOO. Differently from the AFDX experiment, UT maintained similar values for all network sizes. These values were similar to the ones of MFF, which makes sense if one thinks of the PMOO worst-case mentioned in Section 5.1. Since both MFF and UT have the same worst-case, in this type of networks they would perform similarly when looking at this metric. Since the service curves in this experiment are so large when compared to the AFDX, the networks here have an overall lesser server utilization. Because of that, the UT worst-case does not affect it as much, since the servers are not being overloaded as was the case in the AFDX experiments. Moreover, as we increase the number of flows per server, we see in the table that the percentage of flows for SFA outperforming PMOO grows for EIB. This could indicate that the EIB PMOO worst-case shown in Section 5.1 is more common than the MFF and UT ones.

Table 5.3: Percentage of Subflows where SFA Outperforms PMOO - Small GLP

Algorithm	*1 Flow per Server*	*4 Flows per Server*	*8 Flows per Server*
UT	7.8%	0.8%	0.1%
EIB	20.0%	28.9%	32.0%
MFF	6.0%	0.8%	0.1%

Source: Author

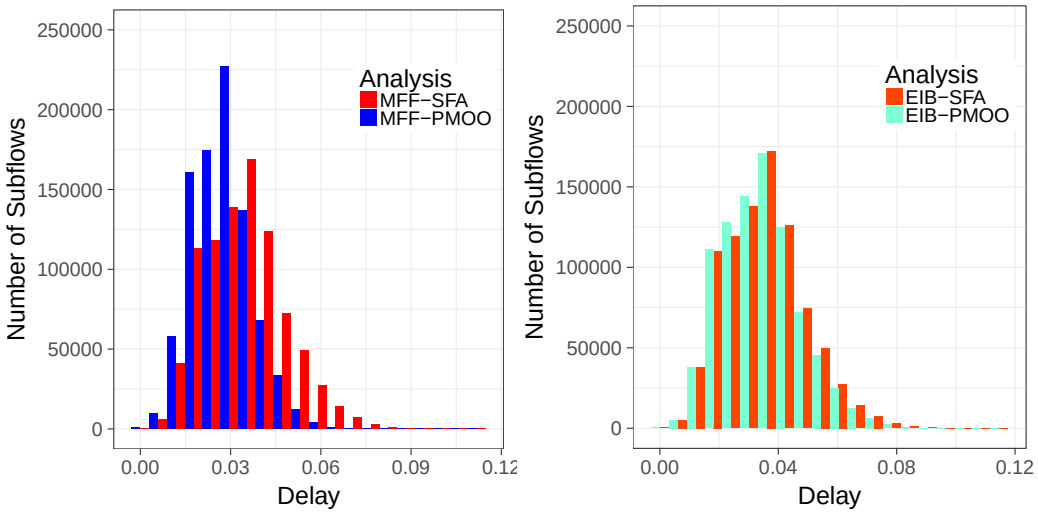
For the second experiment, we doubled the number of devices in the network, but maintained one flow per server. This modification of the number of devices increases the number of flows, the complexity of the interference pattern, and the average flow path length. Because UT is too pessimistic, as in the AFDX larger experiment, we removed it from these larger experiments. This was done in order to achieve cases of higher server utilizations with the other analyses. Even so, in Table 5.4 we see a similar behaviour to the previous GLP experiment, for the same number of flows per server (Table 5.3). This indicates that this behaviour comes from an local overload of servers, rather than a more complex interference pattern or path length, since the service curves and arrival curves were not modified, as well as the flows random distribution. This is also indicated by the histograms in Figure 5.15, which follow a similar pattern to the histogram of Figure 5.10a. For this topology, the MFF analysis yields the tightest bounds, specially the PMOO version. This is seen by the PMOO version (in blue) occuping the leftmost bins, whereas the SFA version (in red) spreads more over the bins to the right in the histogram of Figure 5.15a.

Table 5.4: Percentage of Subflows where SFA Outperforms PMOO - Large GLP

Algorithm	*Percentage of Flows*
UT	—
EIB	22.7%
MFF	2.2%

Source: Author

Figure 5.15: Histograms for MFF and EIB - Large GLP



(a) MFF for Large GLP

(b) EIB for Large GLP

Source: Author

6 CONCLUSION

This chapter presents the concluding remarks. We first present a brief summary of this work, and we highlight its contributions. Next, we conclude discussing future work.

6.1 Summary and Contributions

We began the study of DNC in Chapter 2, where we explored the network modelling of DNC. In this same chapter, we showed the most common unicast analyses, the TFA, SFA and PMOO analyses.

In the next chapter we showed some works that aimed at comparing different approaches to the problem of analysing multicast networks. After that, we also presented the two traditional multicast analyses offered by DNC, the Multicast TFA and the Unicast Transformation.

In Chapter 4 we presented the most recent algorithms for multicast analysis present in the literature. These are the Explicit Intermediate Bounds and the Multicast Feed Forward analyses.

The next chapter started by discussing some behaviour known to the PMOO analysis. Not only that, but it also described how this behaviour is translated to the multicast analyses. More specifically, the MFF, EIB, and UT analyses. After this initial discussion, the experiment results were shown. These experiments were the delay bounding of all flows of different networks, using an AFDX topology and an Internet-like one. Since these networks were randomly created, each experiment had 1000 repetitions.

We concluded, based in such experiments, that for the vast majority of cases MFF yields the tightest bounds. It constantly outperforms EIB, specially in networks with higher utilizations. Moreover, both analyses outperform UT, which is one of the traditional multicast analyses for DNC.

We also found out that MFF is very resilient to the PMOO worst-case for networks with low to medium utilizations. This means that for the vast majority of subflows PMOO outperforms SFA. However, for networks with very high server utilization SFA might be more suitable. EIB on the other hand is much more prone to this PMOO worst-case. Therefore, for networks with complex flow interference patterns EIB SFA can be a better approach than EIB PMOO. To counter this, one could use the EIB aggregate arrival bounding mentioned in (BONDORF; GEYER, 2016), which computes EIB with both

PMOO and SFA for each subpath and uses the best value.

However, MFF still outperforms EIB for most cases, even in such worst-case situations. Therefore, we conclude that for networks in general MFF is the most suitable multicast analysis. For most cases, one should use the MFF PMOO version. However, if the network being analysed has a very high utilization, the MFF SFA can offer the tightest bounds.

6.2 Future Work Directions

Because DNC has a high complexity, our numerical experiments focused on small networks, since each experiment had 1000 repetitions. Therefore, we would like to have as a future work the execution of even larger networks. One example being the A320 shown in (TOBECK, 2017), which is much larger than the A350 used in our experiments. Moreover, we would also like to vary not only the number of flows per network, but also the amount of sinks per flow, in order to verify how this affects the end delay of each subflow. This could make MFF outperform EIB even more, since EIB breaks the multicast flows depending on the subpaths.

Another possibly interesting experiment is the study of the runtimes of the analyses. EIB's initial step performs some computation to the whole network, creating small flows. When bounding the delays of each flow of the network, this causes the paths to be much shorter. This could cause EIB to greatly outperform MFF time-wise, since MFF's analyses are independent of each other, and operate on much longer paths. Since EIB can be much faster than MFF, one could potentially use it to analyse larger networks, in which it would be impractical to use MFF. This means that by loosing the delay bounds one can potentially gain performance regarding the execution time, allowing for the analysis of much larger networks.

REFERENCES

BAUER, H.; SCHARBARG, J.; FRABOUL, C. Applying and Optimizing Trajectory approach for performance evaluation of AFDX avionics network. In: **Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)**. [S.l.: s.n.], 2009. p. 1–8. ISSN 1946-0759.

BONDORF, S.; GEYER, F. Generalizing network calculus analysis to derive performance guarantees for multicast flows. In: **Proceedings of the 10th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2016)**. [s.n.], 2016. Available from Internet: <https://disco.informatik.uni-kl.de/discofiles/publicationsfiles/BG16.pdf>.

BONDORF, S.; SCHMITT, J. Calculating accurate end-to-end delay bounds – you better know your cross-traffic. **EAI Endorsed Transactions on Future Internet**, ACM, v. 16, n. 11, 1 2016.

BONDORF, S.; SCHMITT, J. B. The discodnc v2 – a comprehensive tool for deterministic network calculus. In: **Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)**. [s.n.], 2014. Available from Internet: <https://disco.cs.uni-kl.de/discofiles/publicationsfiles/BS14.pdf>.

BU, T.; TOWSLEY, D. On distinguishing between internet power law topology generators. In: **Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies**. [S.l.: s.n.], 2002. v. 2, p. 638–647 vol.2. ISSN 0743-166X.

CATTELAN, B.; BONDORF, S. Iterative design space exploration for networks requiring performance guarantees. In: **Proceedings of the IEEE/AIAA 36th Digital Avionics Systems Conference (DASC 2017)**. [s.n.], 2017. Available from Internet: <https://disco.cs.uni-kl.de/discofiles/publicationsfiles/CB17.pdf>.

CHANG, C.-S. **Performance Guarantees in Communication Networks**. New York, NY: Springer-Verlag, 2000.

CHARARA, H. et al. Methods for bounding end-to-end delays on an afdx network. In: **18th Euromicro Conference on Real-Time Systems (ECRTS'06)**. [S.l.: s.n.], 2006. p. 10 pp.–202. ISSN 1068-3070.

GRIEU, J. **Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques**. Thesis (PhD) — Institut National Polytechnique de Toulouse, France, 2004.

HAMSCHER, A. **Large-Scale Numerical Evaluation of Network Calculus Analyses**. [S.l.], 2018. Bachelor Thesis.

HOTESCU, O. et al. Towards quality of service provision with avionics full duplex switching. In: . [S.l.: s.n.], 2017.

KEMAYO, G. et al. A forward end-to-end delays analysis for packet switched networks. In: **Proceedings of the 22Nd International Conference on Real-Time Networks and Systems**. New York, NY, USA: ACM, 2014. (RTNS '14), p. 65:65–65:74. ISBN 978-1-4503-2727-5. Available from Internet: <http://doi.acm.org/10.1145/2659787.2659801>.

Le Boudec, J.-Y.; THIRAN, P. **Network Calculus: A Theory of Deterministic Queuing Systems for the Internet**. Berlin, Germany: Springer-Verlag, 2001.

MIGGE, J. **L'ordonnancement sous contraintes temps-reel : un modele a base de trajectoires**. 204 p. p. Thesis (PhD), 1999. Thèse de doctorat dirigée par JEAN MARIE, ALAIN Sciences et techniques Nice 1999. Available from Internet: <http://www.theses.fr/1999NICE5341>.

PULIAFITO, A.; TRIVEDI, K. S. Deterministic network calculus analysis of multicast flows. In: ____. **Systems Modeling: Methodologies and Tools**. 1. ed. [S.l.]: Springer International Publishing, 2018. chp. 5.

SCHMITT, J.; ZDARSKY, F. A.; FIDLER, M. Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch ... In: **27th IEEE International Conference on Computer Communications (INFOCOM 2008)**. Phoenix, AZ, USA: [s.n.], 2008. Available from Internet: <https://disco.cs.uni-kl.de/discofiles/publicationsfiles/SZF08-1.pdf>.

SCHMITT, J. B.; ZDARSKY, F. A. The DISCO Network Calculator – A Toolbox for Worst Case Analysis. In: **Proc. ValueTools**. [S.l.: s.n.], 2006.

SCHMITT, J. B.; ZDARSKY, F. A.; MARTINOVIC, I. Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once. In: **Proceedings of GI/ITG International Conference on Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB)**. [S.l.: s.n.], 2008. p. 1–15.

STAROBINSKI, D.; KARPOVSKY, M.; ZAKREVSKI, L. A. Application of Network Calculus to General Topologies Using Turn-Prohibition. **IEEE/ACM Transactions on Networking**, v. 11, n. 3, p. 411–421, 2003.

TECHNICAL REPORT AERONAUTICAL RADIO INC. **ARINC Specification 664: Aircraft Data Network Parts 1 27**. [S.l.], 2002–2005.

TINDELL, K.; CLARK, J. Holistic schedulability analysis for distributed hard real-time systems. **Microprocess. Microprogram.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 40, n. 2-3, p. 117–134, abr. 1994. ISSN 0165-6074. Available from Internet: <http://dx.doi.org/10.1016/0165-6074(94)90080-9>.

TOBECK, N. Enforcing domain segregation in unified cabin data networks. In: **2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)**. [S.l.: s.n.], 2017. p. 1–8.

TOMASIK, J.; WEISSER, M. A. Internet topology on as-level: Model, generation methods and tool. In: **International Performance Computing and Communications Conference**. [S.l.: s.n.], 2010. p. 263–270. ISSN 1097-2641.