|  |
| --- |
| 2009303476 |

|  |  |  |
| --- | --- | --- |
| 题 目 | AFDX网络时延算法分析与实现 | |
|  |  | |
|  |  |  |
|  | 学 院  专业名称 | 软件与微电子学院 |  |
|  | 软件工程 |
|  | 学生姓名 | 张松 |
|  | 指导教师 | 张涛 |
|  | 毕业时间 | 2013.7 |





**本科毕业设计论文**

**题 目**­­­­­­­­­­­­­­­­ AFDX网络时延分析

算法设计与实现

专业名称 软件工程

学生姓名 张松

指导教师 张涛

毕业时间 2013.07

摘 要

AFDX（Avionics Full Duplex Switched Ethernet，航空电子全双工以太网），是由ARINC 664规定的新一代航空电子网络系统。由于AFDX网络具有高带宽、强实时的属性，在未来的机载航空电子系统中，AFDX网络将得到广泛的应用。AFDX网络是基于传统商用以太网标准上，强制规定了网络的冗余、时延等相关属性的航空网络。航空数据网络(ADN)最重要的特性—QoS由多种因素决定，而决定服务质量的带宽必须得到保证，抖动、延迟和误码率等必须控制在一定范围内。AFDX网络采用了全双工的链路消除数据帧的冲突情况，然而由于交换机队列的延迟的随机性，必须要对网络中包传输的响应时间的确定上限进行计算。

论文对计算上述时延上限的两种方法进行了研究和实现，本毕业设计所做的主要工作如下：

1）实现了网络演算方法，一种用来计算端到端时延上限，使网络满足协议保证的约束的方法。本论文根据网络演算中到达曲线、服务曲线的概念，应用最小加代数理论，将网络中物理设备视为独立的节点进行分析，得到了虚链路经过整个网络的时延上限。

2）实现和改进了计算分布式网络中流的时延上限的轨迹法。本论文基于轨迹法和AFDX网络特性，建立并简化改进了轨迹法的数学模型，分析了将轨迹法应用于工业AFDX网络的过程。在软件中，基于流的到达时间、开始处理时间，根据定义中拓扑结构，实现了该算法。同时，本论文还应用模型检验的方法，对简单拓扑下的虚链路的端到端响应时间作了计算，指出了基本模型的缺陷，并提出了一种改进方法。

3）利用所完成的分析工具软件，检验网络实例是否满足所设计的时延上限，并通过仿真测试，得出实际的时延数据，验证所实现的两种算法。

论文描述了在时延分析工具中实现上述两种算法的具体过程，结合具体工业数据，针对相关实例，计算得出了特定拓扑参数下的时延上限。论文得出了基本轨迹法优于基本确定性网络演算方法的结论，并提出了一种对于轨迹法的改进方法。

关键词：AFDX，最差端到端响应时间，时延上限，网络演算，轨迹法

ABSTRACT

AFDX (Avionics Full Duplex Switched Ethernet), standardized as ARINC 664 is a major upgrade for avionic systems. Because of the high speed and real-time property, AFDX will be widely applied in the avionic system in the future. AFDX is an avonic system, based on the traditional commercial Ethernet standard, and meets the mandatory certification of network redundancy and delay bound. The most important characteristics of an ADN(Aircraft Data Network) are QoS(Quality of Service). Various ADN attributes determine the QoS. But a guaranteed bandwidth, limited jitter, upper bounded transmit latency and a low BER must be provided. AFDX elimates the conflicts of data frame by the full-duplex link. But because of the stochasticity of delays in the switch queues, it is necessary to calculate the deterministic bound of packets’ response time in the network.

This paper studies and implements the two approaches to calculate the upper bound above. This diploma project’s work is as follows:

1) This paper implements the Network Calculus, as an approach to calculate the end-to-end upper-bounds for certification purposes. According to the concepts of arrivial curve and service curve in NC, and the theory of min-plus, this paper analyses the devices in the network as individual nodes and calculates the upper bounds of virtual links across the whole network.

2) This paper implements and improves the trajectory approach, as a method to calculate the upper bounds of flows in the distributed network. According to the trajectory approach and the AFDX network’s features, this paper sets up, simplifies and improves the model of the trajectory approach in maths and analyzes the process of applying the approach to industrial AFDX network. The software developed implements the algorithm according to the defined network topology, the arrivial time and starting conducting time. Meanwhile, this paper calculates the end-to-end response time of virtual links in a simple network topology using model checking. This paper also points out the drawbacks of the base model and raises an approach to improve it.

3)Utilizeing the implemented analysis software, the paper’s work checks whether the network example meets the upper bounds designed. Besides, the simulation part gets the real data of delays, and verifies the two algorithms implemented.

This paper describes the detailed process of implementing the above two algorithms in the network delay analysis tool, and calculates the upper bound of delays of a specific topology’s configurations, with the real industrial case and data. This paper also concludes that the base trajectory approach is better than the base network calculus method. Moreover, this paper gives an improvement of the trajectory approach.

**KEY WORDS**：AFDX, The worst case end-to-end response time, Upper bound, Network Calculus, Trajectory approach

目 录

[**第一章 绪论** 6](#_Toc358304292)

[1.1课题来源与背景 6](#_Toc358304293)

[1.2国内外研究现状 6](#_Toc358304294)

[1.3课题研究意义 6](#_Toc358304295)

[1.4论文主要结构 7](#_Toc358304296)

[**第二章 AFDX网络时延基础知识** 8](#_Toc358304297)

[2.1AFDX网络时延分析概述 8](#_Toc358304298)

[2.1.1AFDX的概念 8](#_Toc358304299)

[2.1.2AFDX网络流量特性 8](#_Toc358304300)

[2.1.3AFDX网络时延组成 8](#_Toc358304301)

[2.2虚链路 8](#_Toc358304302)

[2.3网络时延上限 11](#_Toc358304303)

[**第三章 确定性网络演算** 12](#_Toc358304304)

[3.1流量模型 12](#_Toc358304305)

[3.1.1聚集函数，离散和连续模型 12](#_Toc358304306)

[3.1.2虚延迟，数据积压 13](#_Toc358304307)

[3.1.2到达曲线和服务曲线 13](#_Toc358304308)

[3.1.3时延上限 15](#_Toc358304309)

[3.2 AFDX网络中的时延模型 15](#_Toc358304310)

[3.3算法流程设计 16](#_Toc358304311)

[3.4 算法实例分析 18](#_Toc358304312)

[**第四章 轨迹法** 22](#_Toc358304313)

[4.1轨迹法理论模型 22](#_Toc358304314)

[4.1.1模型参数定义 22](#_Toc358304315)

[4.1.2模型分析 23](#_Toc358304316)

[4.1.3忙时段累加过程 24](#_Toc358304317)

[4.1.4最差端到端响应时间 25](#_Toc358304318)

[4.2轨迹法时延分析模型 26](#_Toc358304319)

[4.2.1轨迹法在AFDX网络中的对应量 26](#_Toc358304320)

[4.2.2流方向的简化处理 27](#_Toc358304321)

[4.2.3对到达时间与输入抖动的简化 28](#_Toc358304322)

[4.2.4时延上限公式的最终形式 28](#_Toc358304323)

[4.3轨迹法分析算法设计 29](#_Toc358304324)

[4.3.1迭代算法的实现描述 29](#_Toc358304325)

[4.3.2迭代算法实现的基本流程 30](#_Toc358304326)

[4.3.3算法实现的模型检验分析 35](#_Toc358304327)

[4.3.3算法实现的实例分析 36](#_Toc358304328)

[4.4轨迹法的改进 39](#_Toc358304329)

[**第五章 实例分析与验证** 43](#_Toc358304330)

[5.1工业数据实例 43](#_Toc358304331)

[**第六章 结论** 47](#_Toc358304332)

[6.1工作总结 47](#_Toc358304333)

[6.1成果展望 47](#_Toc358304334)

[**毕业设计小结** 52](#_Toc358304335)

[**附 录** 53](#_Toc358304336)

第一章 绪论

1.1课题来源与背景

传统429总线技术可靠性高，但传输速率低[1]，随着航空应用渐趋复杂，传统的429总线技术已不能满足新一代航空应用对于高带宽、大数据量传输的需要。AFDX网络[2][3]作为新一代总线技术，利用了商用以太网的现成技术和开放标准，它在IEEE802.3上，增加了一些特殊限制和功能[4][5]，提供了一个具有确定性和服务保证的网络[6][7]。在A380等新一代航空飞机上，AFDX网络已经被实际应用，并且在提高数据传输速率、简化布线的复杂程度、降低飞机负载等方面，体现出明显的优势[3]。波音公司也基于AFDX数据网络研发生产了787飞机，随着AFDX技术的应用的增多，和对AFDX协议标准的广泛承认，生产AFDX网络设备、部署AFDX网络的代价会逐步降低，AFDX网络将作为新一代航空电子骨干网络，在航空领域将获得得大规模的应用。在分析AFDX网络可靠性时，要对整个交换机集群[8]进行研究，而不是简单地重复累计经过交换机的时延。还要求计算包的传输时延，从而控制数据流的端到端响应行为特征，保证网络的实时性、可靠性。

1.2国内外研究现状

R. L. Cruz等人提出了分析计算网络时延，进行网络分析的确定性网络演算算法[9]。Le Boudec等人将其发展成为成熟的网络演算算法[10]，并针对工业以太网的特点和物理设备提出了诸多网络数据模型。Charara. H等对于将确定网络演算应用于AFDX网络做了研究[11]。在国内，北航的杨云等基于该方法，对AFDX网络数据帧的延迟上界做了计算分析[12]。

轨迹法是较新的计算网络延迟的方法，它首先由Steven Martin等人在并行和分布式处理会议上提出[13]，并且经过经过不断完善[14][15]，形成了成熟计算最差情况下网络时延的方法。Bauer等对于AFDX网络下的轨迹法，做了大量的分析研究[16][17]。国内的熊华刚等人使用了递归的计算方法在AFDX网络中实现了轨迹法[18]。

目前，国内仍然缺少针对轨迹法的理论的分析研究成果，同时采用这两种分析算法，同时缺少对我国AFDX网络配置特点，进行分析计算的，成熟可用的分析工具。

1.3课题研究意义

在软件中实现网络演算，进一步分析研究AFDX网络的轨迹法的应用，针对网络实际配置，不断改进优化算法，对于AFDX网络在国内的普及应用，验证网络部署的可行性，有着重要理论价值和对于航空网络工业的工程价值。

传统上，对于以太网的上述特性的分析主是要基于该成熟的网络演算理论。

最初，轨迹法在分布系统中被提出，现在，轨迹法的相关理论已经能够作为一种成熟算法，在工业领域针对AFDX网络的特性进行应用。

本文基于AFDX网络，在分析工具中，根据网络演算的流量模型[19]，实际应用网络演算[20]相关理论，并在基本轨迹法基础上，构建面向流和节点的数据模型，以虚链路为基础[21]，进行对网络整体分析计算。本文轨迹法采用的是基本的FIFO策略，即以到达时间顺序确定优先级高低，而不额外添加优先级的限制[22]，从而得出更为合理的时延上限。同时本文结合了网络演算方法和轨迹法，相互对照，提出对于基本轨迹法的改进。本文提出了上述算法在分析工具中的软件实现，包括了算法模块的架构设计、对实现的改进方法和针对实例数据的分析结果。在实例分析中，利用仿真平台相关的原理设计[23] [24] [25]，借助对航空嵌入式系统进行评估的相关理论成果[26]，搭建的实际测试平台，对实现和改进的算法进行验证。

实现了相关算法的时延分析工具，以及本文对于轨迹法的AFDX实现的分析研究，在AFDX网络广泛推广使用，新一代航空网络总线技术研究深入开展的背景下，对于给予AFDX技术的物理设备部署，路由配置优化，网络拓扑设计都将有着借鉴意义和工业应用价值。

1.4论文主要结构

本文共分为六个章节：

1. 第一章：绪论。介绍了课题的来源与背景知识，国内外对于AFDX网络时延分析的现状，本文所做工作的理论意义和工程价值。
2. 第二章：AFDX网络时延基础知识。主要对虚链路相关知识和网络时延产生原因进行了分析研究。
3. 第三章：确定性网络演算算法分析与实现。对于网络演算算法在AFDX网络中的应用做了分析，并详细描述软件实现过程。
4. 第四章：轨迹法。分析了轨迹法的基本思想，模型对应概念和在AFDX网络分析工具中的改进实现。
5. 第五章：借助仿真测试平台，在工业实例配置下，对算法结果的验证与比较。

第二章 AFDX网络时延基础知识

2.1AFDX网络时延分析概述

2.1.1AFDX的概念

AFDX(Avionics Full Duplex Switched Ethernet)[3]，航空电子全双工以太网，是ADN(航空数据网络)的数据通信标准之一，由ARINC664协议定义。AFDX是一种基于传统的满足IEEE802.3标准的以太网的串行数据传输方法。AFDX网络以光纤或铜缆为传输介质，链路的传输速率达到10Mbps和100Mbps。AFDX网络通过流量控制和冗余拓扑保证网络的可靠性，通过虚链路VL保证带宽的分配。

2.1.2AFDX网络流量特性

AFDX，作为高速航空以太网，要求了确定性的时延和网络冗余，对于网络的可靠性有着严格的要求。AFDX在传统以太网协议的基础上，如IEEE802.3等，增加了额外限制约束，以保证网络数据传输的确定性。在终端系统(End System)上，数据流被整形，在交换机(AFDX Switch)上，数据流要被流量控制和确定的路由策略所约束。在ARINC 664标准的约束下，传统以太网的不确定性被消除，即不存在由于冲突导致的丢包问题。但是由于网络中的数据包被延迟发送，以满足标准对于数据流特性的限制，从而问题主要集中于交换层：即不同数据流在竞争交换机资源时导致的时延问题。

2.1.3AFDX网络时延组成

AFDX网络的时延，即一个数据帧经过AFDX网络所需要的时间，由两个部分组成：可变部分和不可变部分。在端口上的处理时间和链路传输时延均为不可变时延，在缓冲队列中的排队的交换时延是可变时延。

2.2虚链路

在AFDX网络中，定义了虚链路的概念：虚链路是基于静态网络路由信息的逻辑链路概念，包括源终端、目的终端、经过的交换机的路径信息定义，还包括对于链路上传输的包的特性，如最大帧长Smax、最小帧长Smin等的定义，此外，还有对于数据流的特性如带宽分配间隔BAG(Bandwidth Allocation Gap)等的定义。简单地说，BAG就是对于每个VL，包与包之间的发送时间的最小时间间隔。在ARINC664P7标准中，对于虚链路在终端和交换机上的属性都有规定。如对于最大允许的抖动、子VL的数量、不同VL的优先级、服务类型、冗余网络的选择等，如下表：

表1-1 虚链路属性

|  |  |  |
| --- | --- | --- |
| Parameter | ES | Switch |
| BAG  (Bandwidth allocation GAP) | Yes | Yes |
| Frame Size | Yes | Yes |
| Max. allowed Jitter | Yes | Yes |
| No. of Sub-VLs | Yes | No |
| Account Type | No | Yes |
| Priority | No | Yes |
| Network Selector | Yes | No |
| Skew Max | Yes | No |

带宽分配间隔BAG的保证，是在终端上完成的。即终端对VL进行流量整形，延迟帧的发送时间，使数据帧的发送满足一定的周期性，具体的延迟整形情况见下图：

BAG

图1-1 数据帧整形延迟过程

从图中可以看出，AFDX网络对数据帧进行整形，产生的效果是延迟帧的发送，使数据帧的突发性降低，更具有规律性。但是在AFDX网络中还是可能存在数据帧的间隔小于BAG的突发情况，在这种情况下，交换机的流量监管模块会丢掉这些突发的包。

BAG是以毫秒为单位，基于BAG的概念，不同VL对于交换机资源的竞争具有一定的规律性，这是本篇论文的研究基础之一。

Smax是在某个VL上传输的帧的最大帧长，最大帧长的范围是从64字节到1518字节，每一个VL都可以定义各自的最大帧长。从而，在交换机缓冲上，每一个VL都有一个最大处理时间C和最大等待时间W(在FIFO、非抢占调度策略下)。

子VL是满足同一VL属性的多条物理链路，在通常情况下，多条子VL共用一条VL的一段路径，然后分离并到达不同的目的端系统。子VL类似于多播，但是与多播不同，同一虚链路的多条子VL的数据内容并不一致，只是共用了部分路径和具有共同的VL属性。不同的子VL之间通过一定的调度策略，发送子VL的数据帧，如下图描述了对子VL使用循环调度，对不同VL使用FIFO调度策略的示意图：

子VL

循环调度

VL FIFO调度

图1-2 子VL的复用过程

在网络拓扑中，VL从源端系统出发，经过多个交换机，到达目的端系统。由于存在子VL，目标端系统可能不止一个。由于经过的是全双工链路，在端系统和交换机上，端口分为发送端口和接收端口，具体的网络示意图如下：

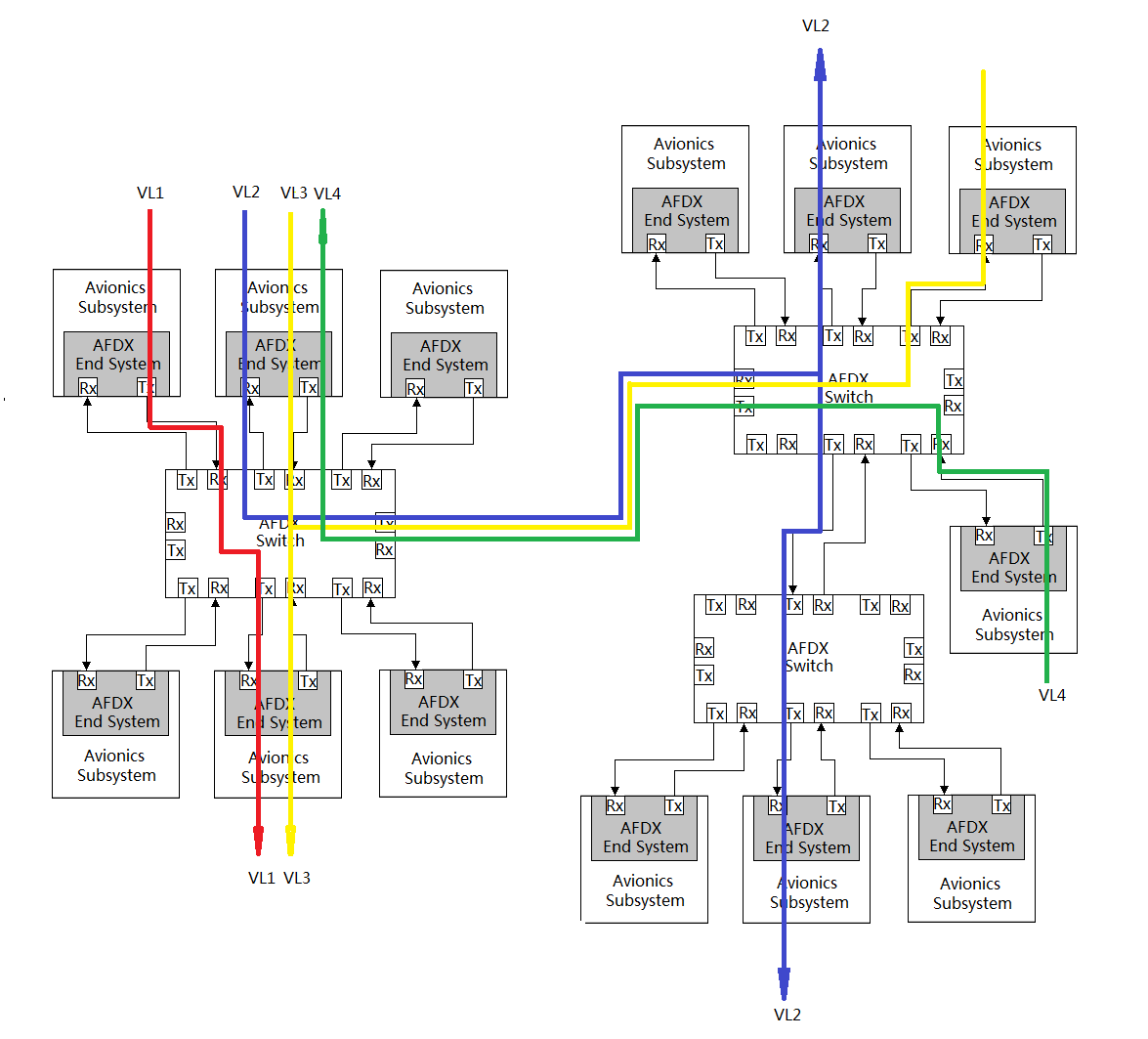


图1-3 AFDX网络拓扑中的VL

2.3网络时延上限

在AFDX网络中，数据包必须以虚链路的方式发送，即数据包必须满足静态的逻辑链路的约束，同时它的流量特性还要遵守虚链路的对于不同业务流的优先级、带宽分配等特性的规范。

在网络拓扑中，多条虚链路会竞争端系统和交换机的资源，在同一条物理链路上复用多条逻辑链路，造成相互影响，结果是时延的不确定性。如在FIFO的，非抢占调度策略下，后到达的数据帧必须等待先到达的数据帧完全处理，才能接受处理，离开缓冲区。所以，时延具有不确定性，

第三章 确定性网络演算

3.1流量模型

网络演算基于最基本的概念：到达曲线，服务曲线和整形器[10]。它将网络中的所有物理设备抽象为由一个或几个网络模型组成的独立的延迟元素，所有经过同一网络模型数据流，在输入时进行叠加，视为一个新的数据流。每一种网络模型[27]-[30]，都对数据流产生不同的影响，主要是产生延迟和对缓冲的要求。

3.1.1聚集函数，离散和连续模型

聚集函数：在时间段内，数据流所传输的比特数。

通常情况下，取,且是广义递增的。在实际的流量系统中，总存在最小的时间粒度(比特，字，单元或包)，因此对于的取值，总是一个由离散时间值组成的有限集合。但是，无论函数是连续还是离散的，为了研究方便，我们通常只取连续的时间量。以连续时间变量为基础，要么是连续函数，这种情况下，模型就被称为流模型，要么，函数是左连续函数或右连续函数。图3-1-1描述的是流模型，图3-1-2描述了右连续的函数模型。图中描述输入函数(即输入的聚集函数)和输出函数(即输出的聚集函数)，表明了以上术语的含义和所做的假设。和是在连续时间变量上的连续函数(即流模型)；在这种情况下，对于每一个时间单元，包都是以比特为单位到达的。和也是建立连续时间基础上的，但是只有一个包被完全接收，才认为这个包到达。

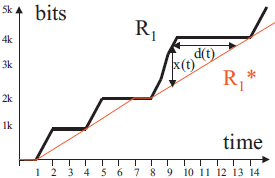


图3-1-1连续函数模型

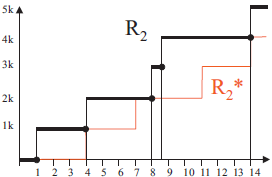


图3-1-2不连续函数模型

速率函数*r*：聚集函数的导数，满足。

对于实际问题中的离散模型，我们能够将其转换为离散模型：

其中，函数*S*是离散函数，而*R*是左连续函数。

3.1.2虚延迟，数据积压

在输入函数和输出函数的基础上，能推出以下的虚延迟的概念：

虚延迟：如果某一比特在到达时，在这一比特到达之前的所有比特都已经被处理，那么这一比特数据所经历的延迟就是虚延迟。对于虚延迟的计算，是要求满足输入函数在时刻*t*的函数值不大于输出函数在之后某时刻的函数值，在这种情况下的最小时间差。从图中来看，即到达曲线和服务曲线的水平差距。

数据积压：在时刻t的数据积压为，其中为输出函数，*R*为输入函数。

3.1.2到达曲线和服务曲线

到达曲线：。是一个广义递增函数，满足对所有的时间,使得都满足条件的限制。

一般地，到达曲线是用一个线性函数表示的。

漏桶：接受输入的数据流，拥有最大容量b，且最初为空的设备。漏桶在不空的时候，以一个恒定的速率输出数据。

如果漏桶溢出，就称输入数据是不符合约束的，否则称输入数据是满足约束的。图3-2说明了漏桶的概念：

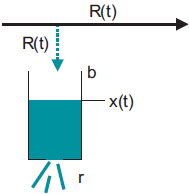


图3-2漏桶

如图所示，输入数据满足输入函数,漏桶的最大容量为b，漏桶的输出速率为r，在时刻t，漏桶的实际容量为。满足：

即，在时刻t，为已经到达而未输出的，且存在于缓冲池中的数据的累积量。具有属性漏桶速率r和缓冲大小b的漏桶控制器，能够将数据的输入流整形为受到到达曲线限制的输出流。

如果到达曲线满足，那么在任意的时间窗口上，流上所传输的数据都被所限制，这种情况下被称为流的峰值速率。当数据流经过以为最大物理比特率的链路，所产生的到达曲线即满足这种性质。如果到达曲线满足，其中b是一个常量，这意味着数据流上发送的比特数最多为b。

一般的，如果是在漏桶控制下的数据流输出曲线，那么满足。即数据流的源最初突发发送b比特数据，并且随后发送速率不超过。详细证明见[10] p11, Lemma 1.2.2。

一般地，有，。

其中是可接受的数据单元到达的最早时间，是数据单元之间的理想的时间间隔，是数据单元中包的大小。

服务曲线：。且，其中是输入函数，是输出函数，其中是最小加代数中的卷积符号，表示。

即对于任一时间t，总能找到时间s，使得。如果经过一个网络节点后，一个数据比特所经历的最大延迟有上限T，即对于所有的时间t，虚延迟。即如果最大时延有上限T，那么对于所有的时间t，都有，或者：对所有的s。

特别的，脉冲函数具有性质：当时，，且时，。

所以，如果满足，就有服务曲线。更为一般地，我们研究的节点有服务曲线图，其中，R是调度服务的处理速率，T是第一个数据比特到达后，直到被处理转发的时延。图3-3说明了服务曲线的概念：

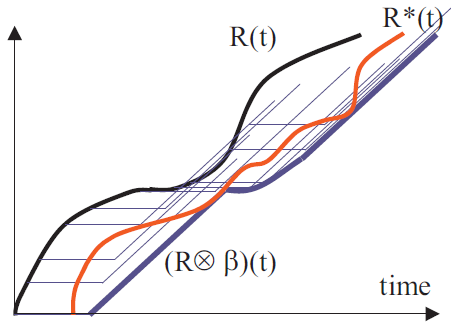


图3-3 服务曲线

3.1.3时延上限

设，其中，是到达曲线，是服务曲线。是对所有，的最大值，从图上看，即到达曲线和服务曲线之间的最大水平差距。这样的定义下，有结论：虚延迟满足，对所有的时间t，有。详细证明见[10] p23, Theorem 1.4.2

如果输入流被漏桶限制，即输入流具有这样形式的到达曲线：。同时假设节点对数据流的调度处理满足服务曲线。如果，延迟上限即。如果，延迟就无上限。示意图如下：

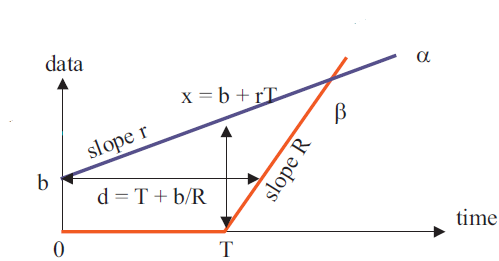


图3-4 时延上限

3.2 AFDX网络中的时延模型

在AFDX网络中，数据流经过端系统的流量整形，满足上节对于漏桶的性质定义，具体的对应关系如下：

到达曲线,

在到达曲线中，b是突发数据量，如果抖动为零，b 就是单个数据帧的长度，在计算最大时延时，由于到达曲线和服务曲线均为广义递增的函数，就有。而*r*是数据帧流量满足的到达速率，是由数据流自身的特性决定的，与BAG和最大帧长有关，等于。

服务曲线：*，*

其中，*C*是链路速率，L是交换机处理的技术时延，即数据帧从上一个端系统或交换机的输出端口输出到完全被交换机缓冲接收，所经历的时延。

到达曲线的变化：，是下一个节点上新的到达曲线。是最小加代数中的反卷积，。

假设每个节点只有一个数据流，则根据3.1.2节的相关内容，如果交换机采用的是FIFO调度策略，第一个数据比特经过交换机，只受到本数据流下一个数据帧的影响。由于同一数据流的数据帧的发送速率不可能超出链路的物理速率，即下一个数据帧的第一个比特到达时，上一个数据帧必然已经接收完毕，而且开始接受处理转发。所以在这样的情况下，数据帧经历的时延上限为上一个数据帧处理转发的时延，用突发函数描述，有，相关变量定义与上文一致。

到达曲线在对突发函数做了反卷积后，在图像上，表现为向左平移了的距离，或是向上平移了的距离，如图：

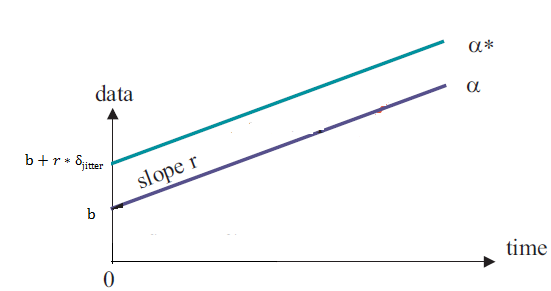


图3-5 到达曲线变化

上述假设是基于单个数据流的基础，但是根据最小加代数相关理论，不同到达曲线具有可加性，包括对于到达速率*r*和突发值*b*的可加性。而且，在同一节点上，服务曲线相同。如果在节点上，将多个数据流复用为一条数据流，分别独立计算各个节点的对数据流产生的时延的最大值，最后累加起来，就能得到包的端到端响应时间的上限。

3.3算法流程设计

基于上述关于网络演算的基础理论，并结合AFDX网络的相关参数定义[31]，在AFDX算法模块中实现网络演算算法。

算法中，将交换机的发送端口视为网络元件[16][17]，在每个网络元件上，都进行到达曲线和服务曲线最大水平差距的计算，在端系统的输出端口上，只存在技术时延，具体算法执行步骤如下：

将网络中所有的有VL经过的的发送端口，视为网络节点放入集合中，记集合为A。r记为集合A中任一节点的到达曲线的到达速率属性, r的初值为0；记b为突发值，b的初值为0；记技术时延为T，链路速率为C；标记虚链路为；标记任一虚链路的经过的节点为；标记集合A中所有的节点为。

D1.[对*i*进行循环]对*i*= *n-1, n-2, …, 1,0.* 实施步骤2，然后转到3。

D2.[对*j*进行循环]对*j=-1, -2, …, 1, 0，*实施本步骤和步骤3，的每一个节点，其中，对节点的*r*属性，置。

D3.[比较*j*与0]若,对节点的*b*属性，置，否则，。

D4.[对*k*进行循环,算数值]对*k=m-1, m-2, …, 1, 0，*节点的延迟属性置为：。

D5.[对*i*进行循环]对*i*= *n-1, n-2, …, 1,0.* 实施步骤6，然后转到8。

D6.[对*j*进行循环]对*j=h-1, h-2, …, 1, 0，*实施步骤7。

D7.[比较*j*和0]如果，置的总时延；否则，置。

算法的流程图如下：

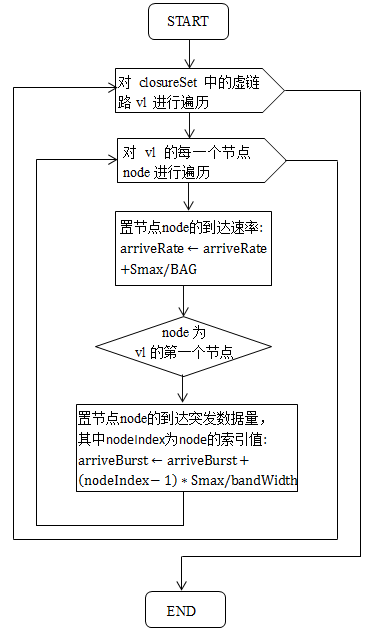


图3-6 NC算法步骤1-3流程图

3.4 算法实例分析

3.4.1网络配置参数

在实际AFDX网络中，具体数据参数如下：

网络中的物理链路是全双工的，链路带宽100Mbps；最大帧长64~1518bytes；BAG范围：1,2,4,8,16,32,64,128ms；交换机的技术时延16ms（技术时延数据来自[17]）；

将上述数据统一单位，时间以计，帧长以bits计，带宽以Mbps计，并取范围内的数值，有：最大帧长4000bits，BAG取4000，技术时延16，链路带宽100Mbps。

在到达曲线中，不考虑复用时，第一个节点上的到达曲线的到达速率属性r为：

得。

数据突发量：

得。

在服务曲线中，延迟属性等于技术时延：

。

假设交换机以线速处理转发数据帧，速率属性等于链路速率：

得。

到达曲线经过每一个网络元件，产生抖动对应突发函数，的延迟属性为：

得。

在我们的实例中，网络拓扑简图如下，图中节点是端系统或交换机的输出端口，且不考虑目的端系统：

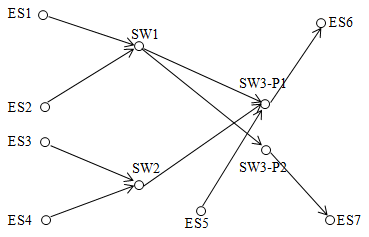


图3-7 实例网络拓扑分析图

如图3-7所示，在网络中有5个源端系统，两个目的端系统，三个交换机，其中交换机SW3有两个输出端口要考虑。虚链路的相关静态路径定义为：

表3-1 虚链路的路径定义

|  |  |
| --- | --- |
|  | {ES1, SW1, SW3-P1} |
|  | { ES2, SW1, SW3-P2} |
|  | { ES3, SW2, SW3-P1} |
|  | { ES4, SW2, SW3-P1} |
|  | { ES5, SW3-P1} |

3.4.2执行过程分析

由于每一个虚链路在经历抖动后得到新的到达曲线，并在端口节点上累加，根据上节提出的算法，执行过程中的具体变量数值变化为：

对虚链路进行遍历，首先考虑，经过的端口节点(不存在标识冲突时，用端口所在的交换机或端系统标识，且不考虑目的端系统)序列为{ES1,SW1,SW3-P1}。

对ES1的输出端口上到达曲线的速率属性进行累加，arriveRate的值增加到1(单位:)，由于ES1为虚链路的源端系统，对应的arriveBurst值为0(单位：bits)。

继续对经过的路径进行遍历，考虑SW1的输出端口，由于经过此端口，arriveRate由0增加到1(单位:)；而在SW1的输出端口已经不再是的源端系统，到达曲线的流量的突发值属性不再为0，为的值：4000(单位：bits)。

在SW3的P1输出端口，考虑到的影响，到达曲线arriveRate属性增加到1(单位:)；经过SW1后，经历了时延抖动，在SW3的P1端口产生的流量突发值为4040(单位：bits)，其中增加的40bits为抖动能够延迟的数据量。

继续对虚链路、、、进行遍历(它们对应的路径分别为{ ES2,SW1,SW3-P2}、{ ES3,SW2,SW3-P1}、{ ES4,SW2,SW3-P1}、{ ES5,SW3-P1})。各虚链路在到达后续端口时，要考虑之前经历的端口产生的抖动对后续端口到达曲线突发值的影响，而对端口的到达曲线的到达速率只是简单累加计算，各个端口在每一次从虚链路对端口进行计算时，端口的到达曲线的相关属性变化如下 ：

表3-2 NC算法步骤1-3中各端口到达曲线属性值变化

|  |  |  |  |
| --- | --- | --- | --- |
| arriveRate(p9~p1) | arriveBurst(p9~p1) | 计算对应  的端口号 | 步骤 |
| 1 | 0 | p1 | 1 |
| 1 1 | 4000 0 | p2 | 2 |
| 1 1 1 | 4040 4000 0 | p3 | 3 |
| 1 1 1 1 | 0 4040 4000 0 | p4 | 4 |
| 1 1 2 1 | 0 4040 8000 0 | p2 | 5 |
| 1 1 1 1 1 | 4040 0 4040 4000 0 | p5 | 6 |
| 1 1 1 1 1 1 | 0 4040 0 4040 4000 0 | p6 | 7 |
| 1 1 1 1 1 1 1 | 4000 0 4040 0 4040 4000 0 | p7 | 8 |
| 1 1 1 1 2 1 1 | 4000 0 4040 0 8080 4000 0 | p3 | 9 |
| 1 1 1 1 1 2 1 1 | 0 4000 0 4040 0 8080 4000 0 | p8 | 10 |
| 1 2 1 1 1 2 1 1 | 0 8000 0 4040 0 8080 4000 0 | p7 | 11 |
| 1 2 1 1 1 3 1 1 | 0 8000 0 4040 0 12120 4000 0 | p3 | 12 |
| 1 1 2 1 1 1 3 1 1 | 0 0 8000 0 4040 0 12120 4000 0 | p9 | 13 |
| 1 1 2 1 1 1 4 1 1 | 0 0 8000 0 4040 0 16120 4000 0 | p3 | 14 |

表3-2中的端口号对应关系如表3-3所示。

在表3-2中，arriveRate(p9~p1)列，记录了端口p9~p1的到达曲线的到达速率属性值在每一步的计算中的变化情况，前12行在对应属性值没有初始化时，对数值在表中的位置留空。arriveBurst (p9~p1)列，记录了端口p9~p1的到达曲线的突发属性值在每一步的计算中的变化情况，前12行在对应属性值没有初始化时，对数值在表中的位置留空。

如后两列计算对应的端口号和相应步骤所示：步骤1~3是对经过的端口节点(1、2、3)进行计算的，步骤4~6是对经过的端口节点(4、2、5)进行计算的，步骤7~9的计算是对经过的端口节点(6、7、3)实施的，步骤10~12是对经过的端口节点(8、7、3)进行计算的，步骤13~14是对经过的端口节点(9、3)进行计算的。

表3-3 端口标识与端口号对应关系

|  |  |
| --- | --- |
| 端口标识 | 端口号 |
| ES1 | p1 |
| SW1 | p2 |
| SW3-P1 | p3 |
| ES2 | p4 |
| SW3-P2 | p5 |
| ES3 | p6 |
| SW2 | p7 |
| ES4 | p8 |
| ES5 | p9 |

根据上一节提出的算法，执行算法计算每一个独立节点上对应的时延，相应的详细过程如下：

首先得到各节点到达曲线相关属性值：如表3-2最后一行所示，在遍历计算了每一个虚链路的每一个端口节点后，各个端口节点的到达曲线属性值为p9的到达曲线的arriveRate属性值为1(单位:)，arriveBurst属性值为0(单位：bits)；p7的到达曲线的arriveRate属性值为2(单位:)，arriveBurst属性值为8000(单位：bits)；其他的端口节点与前述类似，在表中相应位置都有记录。

如节点是源端口，则数据流经过整形后发送，经历的时延为发送时延，即 = 40，如果节点非源节点，则根据3.1.2节的延迟上限公式进行计算，延迟上限为：

其中，T对应了每个输出端口上服务曲线的技术时延属性16，R是链路速率，b对应了arriveBurst的值，arriveRate需要在计算3.1.2节中阐述的对缓冲大小的要求时去考虑。

这样，对于p1，d = = 40，对于p3，d = 16 + = 177.2，对其他节点也进行类似计算，得到表3-4所示的结果。

表3-4 节点时延计算结果

|  |  |
| --- | --- |
| nodeDelay | 端口 |
| 40 | p1 |
| 96 | p2 |
| 177.2 | p3 |
| 40 | p4 |
| 56.4 | p5 |
| 40 | p6 |
| 96 | p7 |
| 40 | p8 |
| 40 | p9 |

按照算法步骤5-7，对VL经过的每一个节点，累加计算各个节点上的时延，得到VL的总时延，在表3-5中，对3.3节中的步骤5~7中对每一个VL的处理分为1~3三步(对VL(5)为两步)，每步都改变相关VL最终的NC算法时延，如表3-4所示：

表3-5 VL总时延计算数值变化

|  |  |  |
| --- | --- | --- |
| NCDelay | VL(ID) | 算法分步 |
| 40 136 313.2 | 1 | 1~3 |
| 40 136 192.4 | 2 | 1~3 |
| 40 136 313.2 | 3 | 1~3 |
| 40 136 313.2 | 4 | 1~3 |
| 40 217.2 | 5 | 1~2 |

表3-5的第一列反映了相关VL在每一步中最终的NC时延结果的变化情况，其中，第三个数值为最终结果，即VL(1)的NC时延为313.2，VL(2)为192.4，等。

3.4.3分析报告结果

分析软件中得到的报告结果，如下：



图3-8 实例分析报告

软件的具体执行流程对应了3.4.2节的分析过程，软件生成的分析报告，与上述算法分析结果相同，即表3-4中的最后结果。VL(1)的NC时延为313.2，VL(2)为192.4，VL(3)的NC时延为313.2，VL(4)为313.2，VL(5)的NC时延为217.2。

第四章 轨迹法

4.1轨迹法理论模型

与网络演算针对数据流的特性进行分析，计算时延不同，轨迹法是基于包的竞争调度，考虑处理时间段的最差端到端响应时间分析算法[13]。由于考虑了包在整个传输路径上，受到的源于自身数据流的包和其他数据流的竞争处理时间的影响，而不是将同一数据流所经过的节点序列视为孤立的，轨迹法在计算最差情况下，端到端响应时间，计算结果一般优于网络演算方法。下面对基本轨迹法的分析是建立在FIFO和非抢占式调度策略的基础上的：

4.1.1模型参数定义

根据[13]，对相关参数进行如下定义。

网络中存在n个流组成的集合{}。每一个流服从一个确定的路径, 是一个有序节点序列，序列的第一个节点作为流的输入节点。此外，一个流是这样定义的：

，集合 {} 中的一个分散的流*；*

*，*流在时间t所产生的包；

*，*包g所属的流的索引值；

，流的端到端最后期限，也就是能够被接受最大的端到端响应时间。在时间t产生流一个包必须在时间 到达。

*，* 流最差情况的响应时间；

， 流的两个连续包之间的最小到达时间间隔（通常称为周期）。

，流在节点h上的最大处理时间。一般地，如果，；

，流在它的输入节点的最大释放抖动。如果在包的产生时间和包开始被调度器处理的时间之间，存在一个不为零的延迟，则一个包服从一个的释放抖动。

*，*包m在节点h上的最晚开始时间；

[*, … ,*]或[1*, … ,q*]，服从流的路径，其中 (相反地，) 是在网络中流访问的第一个节点(相反地，最后一个节点)；

*，*路径的基数，即被流访问的节点数目；

*，*流在节点h之前访问的节点；

，流在节点h之后访问的节点；

，流在路径上访问第一个节点；

，流在路径上访问最后一个节点；

最慢节点：研究的流在路径上所访问的经历延迟时间最长的节点。

，流在路径上访问最慢节点： ，；

，流在路径上访问最慢节点：，；

，流的包从源节点到达h节点花费的最短时间；

，流的包从源节点到达h节点花费的最长时间；

，表示max(0，1 + )；

两个节点之间的延迟下限。

两个节点之间的延迟上限。

， = 。

为包 在节点h上的到达时间

，如果优先级高于或等于，且在时间t之前产生的包，在时间t都已经被处理完了，那么t就被称为优先级对应的空闲时间。

优先级对应的忙时段为时间段[*t*, )，其中，*t*和都是空闲时间。且在(*t*, )内，没有优先级对应的空闲时间。

，为包m在节点*q*上被处理时，所在的忙时段。

, 为内被处理的第一个包。

，为和m之间的来自节点*q-1*的第一个包。

4.1.2模型分析

在模型中，作如下假定：

如果有两个不同的分别服从路径的流和服从路径的流相交，即而且，即如果流与流相交，则流在离开流的路径之后不再访问的任一节点。

在FIFO，非抢占策略的调度机制下，一个包m一旦开始接受处理，处理过程就不会被其他包打断，即不会被延迟。所以轨迹法研究的是包m在某个节点h上的最晚开始处理时间，得到最晚开始处理时间，再加上处理时间长和链路延迟（输出节点不加），做累加后，就能得到最差端到端响应时延。

轨迹法引入了忙时段的概念，在一个包处理时所在的忙时段上，由于包必须在来自路径的上一个节点的前一个包之后被处理，如果前一个包在同一忙时段内，前一个包还必须在忙时段的第一个包之后被处理，所以对于节点q上的包m，要分析来自上个节点的且早到达的q的包，和忙时段的第一个包，做出标记如下

2

1

*p*(2)

*p*(1)

f(2)

*p*(1)

f(1)

……

*p*(h-2)

*p*(h-1)

f(h-1)

h-1

*p*(h-1)

f(h)

*p*(h)

h

……

q-1

f(q-1)

p(q-1)

q

m

p(q-1)

f(q)

图4-1 影响包m的忙时段

4.1.3忙时段累加过程

为了计算包m在节点q上的最晚开始时间，要对上述忙时段中的时间片段进行相加，即：在节点1上从包 *f*(1) 到 *p*(1)的处理时间 从包 *f*(2) 到 *p*(2)的处理时间 () 在节点q上从 *f*(q) 到 (m 1) 。其中，减掉，是因为在计算来自上一个节点的且到达时间在前面的包时，前一个包所经历的延迟与同一忙时间段的第一个包有关，但是要考虑自身到达时间比前一个包的到达时间晚，这个延迟不应该计入。

在计算端到端最晚开始时间的时候，如果对任一节点 h ，有 (4-0)，所得结果的值更大，更符合最差情景，这时有*p*(h1) *f*(h)，即对任一节点h，不存在使得h 。也就是，所有在[f(h), p(h1))之间包，都是在节点h上第一次穿过路径的。因此，在最差情况下，在任意节点h ，都有p(h) f(h。从而有：

(4-1)

其中，可变的部分记为，有

(4-2)

这样，根据定义，对于任何在 [1, q) 之间的节点h，f(h)是第一个在忙时段被处理而且来自节点h的包。除此之外，f(h) 是在忙时段中被考虑的最后一个包。总的来说，如果我们对在忙时段和 中处理的包进行计数时，只有)被计算了两次。

根据S. Martin在[13]第4节中的分析，

可以将m经历的节点序列分为三个部分：在之前访问的节点，节点 自身，和在 之后访问的节点。 等于：

节点

在之前的节点

. (4-3)

在之后的节点

4.1.4最差端到端响应时间

从第一个节点到最后一个节点，对忙时段上的包进行编号：从到 ；考虑到节点h上的包g的处理时间小于或等于，对于重复计数的包，对应的处理时间小于或等于，从而有[13]

(4-4)

如果从流的角度，分析上式，得到（[13]的引理2）：

(4-5)

其中

从而，端到端响应时间上限为：

(4-6)

其中：

其中，对于流*j*对流*i*产生的抖动，有：

，

，

，.

= ，

其中，t是包在开始节点的到达时间。

4.2轨迹法时延分析模型

4.2.1轨迹法在AFDX网络中的对应量

在AFDX网络中，AFDX交换机在接收端口没有输入缓冲，只在每一个输出端口有一个FIFO缓冲。整个网络的流量从终端系统输入和输出，在相连的物理设备之间只有一条物理链路。虚链路定义了一条从一个源端系统到一个或多个目的端系统的静态的通信通道。端系统之间的数据帧的传输都是通过虚链路进行的。一条虚链路与另一条虚链路相交次数不多于两次。且一个虚链路离开交换机后就不会再经过该交换机。

在基本轨迹法中，每一个节点上的包的处理都是基于FIFO策略和非抢占式调度算法的。每一个流的两个连续的包之间有一个最小时间间隔，且有一个可接受的最大的端到端响应时间上限，来自流的每一个包在节点h上都有一个最大处理时间；每一个流都有一个确定的路径；在链路上，规定了一个传输时间上限，和下限。

相应的，在轨迹法中的网络元素的相关定义，在AFDX网络中对应如下：

节点h对应了网络中某一个源端系统的输出端口或者AFDX交换机输出端口；节点之间的连接对应了AFDX交换机之间的，或AFDX交换机与端系统之间的铜缆或光纤；链路上的传输时延，且等于交换技术时延；每一个流对应一条虚链路；

包之间的最小时间间隔；最大处理时间，其中R是处理速度，按照线速计算；

4.2.2流方向的简化处理

在基本轨迹法中，相交的流可能是同向的，也可能是反向的，导致与不一定相同。不同时，有，如图：

（1）



（2）



图4-2，，， 和

在AFDX网络中，一条虚链路不可能两次经过同一条交换机。而且，根据4.2中的假定，虚链路不可能相交两次。因此，如果存在和，有，，即虚链路与虚链路的流向相反，则有如下情景：

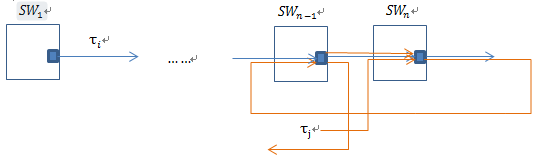


图4-3虚链路方向相反相交

如图，与重合的路径上，路径两端的端口所在的交换机分别为和，即和所在的交换机。

由于与只能相交一次，在离开路径后，就不再经过的路径，联系图4-2，它们所经过的输出端口序列恰好相反，所以流经过的输出端口之后，还必须经过的输出端口。由于在和的输出端口之间，与的路径是重合的，即与都先后经过的输出端口和的输出端口。

由于是与的路径重合时，经过的第一个交换机，这样必须有一个不是来自的物理链路，使得经过该物理链路进入，且经过同一个输出端口。这样，虚链路就经过交换机两次，与“在AFDX网络中，一条虚链路不可能两次经过同一条交换机”的前提不符，所以，不存在这种相交的流（虚链路）反向的情况。

这样，在计算端到端时延上限的公式中，。

4.2.3对到达时间与输入抖动的简化

在时延上限公式中，存在参数流的包在开始节点的到达时间t和流自身的抖动输入抖动，并且指定了t的变化范围：。在我们的模型中，约定以所研究的输入流的包，在第一个节点上的开始到达时间，为时间原点，即认为为常量0，。并认为包输入节点的抖动为0，即。从而的变化范围中的没有意义。流的包在经过的任一节点上的最大处理时间都相同，等于。物理链路上的最大和最小延迟都等于技术时延*L*。

4.2.4时延上限公式的最终形式

在上两个小节对公式中的相关量,做简化，并记和和在其路径上任一节点的最大处理时间，记为流的包*m*在节点*h*上的最晚到达时间，为流的包*m*在节点*h*上的最早到达时间，取作，*L*为物理链路上包传输的技术时延，从而可得：

(4-7)

代入AFDX网络相关参数得公式最终形式：

(4-8)

其中流对流产生的抖动：

(4-9)

，

，.

= ，

4.3 轨迹法分析算法设计

4.3.1迭代算法的实现描述

根据Steven Martin在[14]中的迭代算法思想，对上节最终的公式形式，迭代算法的实现如下：

(i)确定网络中与流直接或间接相交的流的集合 (包括)。

(ii)对每一个流，初始化，对于流的其他节点*h*，有。

并对所有的节点*h*初始化。

对做初始化, =

其中，。

(iii)进行如下迭代运算：

D1.[置q的值]置。

D2.[迭代]置，实施步骤3，转到8

D3.[对中的运算]遍历中的虚链路，对任一虚链路，实施步骤4

D4.[遍历节点]对路径上的节点进行遍历，对任一节点*h*实施步骤5~7

D5.[计算最晚开始处理时间]运用公式(4-8)，利用得到的的值，计算在节点*h*上的最晚开始时间

D6.[计算最晚到达时间]如果，即*h*是虚链路的路径上，和相交的第一个节点之前的节点，计算的值：

D7.[计算响应时间上限]如果，即*h*是虚链路路径上的最后一个节点，计算响应时间：

D8.[迭代条件判断]如果对中的任一虚链路，如果在经过与其他任一虚链路相交的第一个节点*h*上，的包的在第*q+1*层次上，最晚到达时间都等于第*q*层的最晚到达时间，则结束算法，否则转到2。

4.3.2迭代算法实现的基本流程

在算法实现中，由CAlgorithm类完成对相关的基础类，如CVirtualLink、CNodeDelay的初始化和遍历操作，CAlgorithm类还在函数中定义了局部变量如virtualLinksSet、vlNode等。算法的实现流程图如下：

第一步

将与虚链路直接或间接相交的

虚链路放入集合中

CAlgorithm::putRelatedVLsIntoSet

第二步

初始化最晚到达时间、最晚开始处理时间、M等

CAlgorithm::initiateStartMaximum

ArriveTime

第三步

执行迭代运算

CAlgorithm::iterateCaculateDelay

图4-4轨迹法算法执行流程图

迭代算法部分的执行流程图如下,其中，virtualLinksSet是上述算法第一步得到的直接相交或间接相交的虚链路组成的集合：

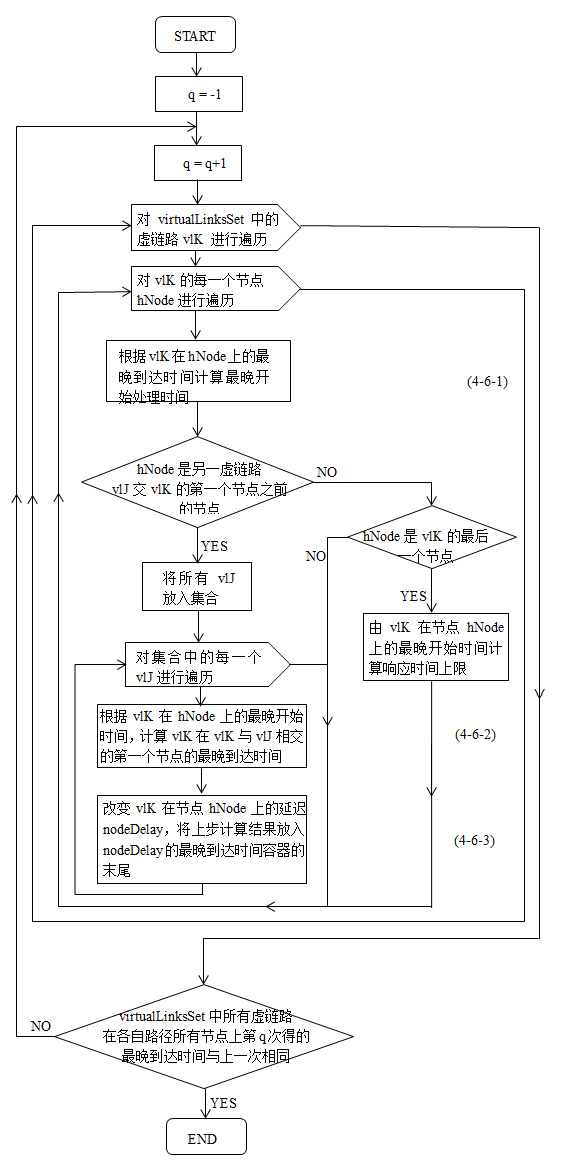


图4-5算法的迭代部分流程图

迭代过程包含在一个循环内部，循环的条件为：virtualLinksSet中的所有虚链路经过的的所有节点的最晚到达时间与上次循环结束时得到的结果相同。在分析工具算法模块的相应实现中，每一个虚链路virtualLink在它经过的每一个节点vlNode上都对应了一个延迟对象nodeDelay，在nodeDelay中定义了一个容器成员，该容器在每次循环后，将得到的新的最晚到达时间附加到容器末尾。相关CVirtualLink类和CNodeDealy类的UML图如下：



图4-6迭代过程中的CVirtualLink类和CNodeDealy类

其中，对于步骤4-6-1，由在hNode上的最晚到达时间,计算最晚开始处理时

间,步骤的流程图如下,其中，VLsSet即virtualLinksSet：

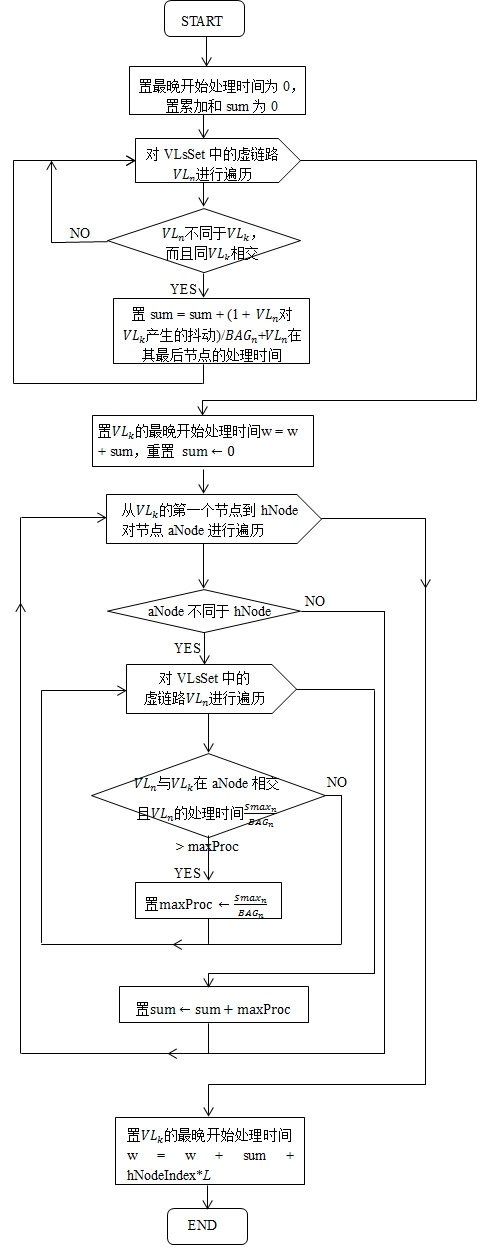


图4-7最晚开始时间计算流程图

4.3.3算法执行的实例分析

本节使用与3.4节相同的网络拓扑配置数据，进行轨迹法算法实现的过程的分析。

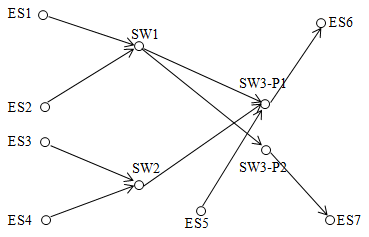


图4-8 实例的网络拓扑

重复3.4节的网络拓扑，如图4-8，重复端口的标识与端口号的对应关系，在表4-1中：

表4-1 端口标识与端口号对应关系

|  |  |
| --- | --- |
| 端口标识 | 端口号 |
| ES1 | p1 |
| SW1 | p2 |
| SW3-P1 | p3 |
| ES2 | p4 |
| SW3-P2 | p5 |
| ES3 | p6 |
| SW2 | p7 |
| ES4 | p8 |
| ES5 | p9 |

在CAlgorithm类中执行实现了的算法过程：

首先，函数CAlgorithm::putRelatedVLsIntoSet将与所研究的虚链路直接或间接相交的虚链路，包括本虚链路，放入virtualLinksSet。

对于，得到虚链路集合{, , , , }。

再执行CAlgorithm::initiateStartMaximumArriveTime，初始化最晚到达时间、最晚开始处理时间、M等属性值，其中最晚开始处理时间还要放入延迟对象的相关属性值容器中。

得到如下结果：

表4-2 初始化虚链路的属性值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性值  VL | | startMaximumArrive | worstBeginningProcessTime | M\_Delay |
|  | P1 | [0] | 0 | 0 |
| P2 | [56] | 0 | 56 |
| P3 | [112] | 0 | 112 |
|  | P1 | [0] | 0 | 0 |
| P2 | [56] | 0 | 56 |
| P3 | [112] | 0 | 112 |
|  | P1 | [0] | 0 | 0 |
| P2 | [56] | 0 | 56 |
| P3 | [112] | 0 | 112 |
|  | P1 | [0] | 0 | 0 |
| P2 | [56] | 0 | 56 |
| P3 | [112] | 0 | 112 |
|  | P1 | [0] | 0 | 0 |
| P2 | [56] | 0 | 56 |

初始化结束后，对virtualLinksSet中的虚链路，进行迭代运算：

首先置，得q = 0。

对virtualLinksSet中的虚链路进行遍历，首先是对进行。

①从的第一个节点开始P1开始，计算最晚开始处理时间。如图4-7所示，在第一个循环后，由于P1是路径上的第一个节点，没有其他虚链路与相交，所以得到值0；重置sum=0，进入第二个循环，在第二个循环中，hNode是唯一一个也是最后一个节点，sum的值不会增加；此外，hNodeIndex为零，得到的最晚开始处理时间计算得0。

又知P1是交的第一个节点P2之前的节点，则在图4-6所示的算法中，执行步骤(4-6-2)和(4-6-3)之后，得到在节点P2上的最晚到达时间为0 + 16 = 16，并将16放入在节点P2的延迟属性的最晚到达时间容器中。

②对经过的节点继续进行遍历，即对P2进行上述计算。由在节点P2上的最晚到达时间计算最晚开始处理时间，执行图4-7所示的算法执行流程。

在图4-7所示的流程中的第一个循环，对sum进行累加，增加的值为对包的的处理时间，即40；重置sum = 0之后；在第二个循环中，对的经过非最后一个节点的节点，即第一个节点P1，在该节点上，只有虚链路，从而sum的值增加40；最后对的最晚开始处理时间加上在链路上的延迟，得到最终值40 + 40 + 16 = 96。

由拓扑知，P2是、交的第一个节点P3之前的节点，执行步骤(4-6-2)和(4-6-3)，在P2节点的最晚开始处理时间基础上，加上链路上的延迟16，得到在节点P3上的最晚到达时间，为96 + 16 = 112。

③继续遍历路径上的下一个节点P3，由在节点P3上的最晚到达时间，即上步计算所得96，计算最晚开始处理时间，执行图4-7所示的算法执行流程。

在流程的第一个循环中,，累加在节点P3以及P3之前的节点上，与直接相交的虚链路的对包的处理时间（同时考虑抖动的影响），包括、、、，得sum = 40 + 40 + 40 + 40 = 160；重置sum = 0之后，在第二个循环中，对的经过非最后一个节点的节点，包括第一个节点P1，第二个节点P2，在这两个节点上，找到与直接相交的虚链路中对包的最长处理时间，同上，为40，进行累加后，sum的值增加到40；最后对的最晚开始处理时间加上在两条链路上的延迟，得到最终值160 + 80 + 16 + 16 = 272。

回到图4-6所示的主流程，P3已经是经过的最后一个节点，所以在加上对包处理的最大处理时间40，得到利用轨迹法得出的最差端到端响应时延312。

以上是对的时延上限的计算过程，循环层次q = 0。再对，分别执行上述运算。

一次循环结束后要对循环条件进行判断，即对P2、P3、P7进行检验，每一次循环，计数为q，得到如下条件判别表：

q = 0时：

表4-3 第一次循环条件判别表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性值  VL | | startMaximumArrive |  |  | 是否  相等 |
|  | P2 | [56,56] | 56 | 56 | 是 |
| P3 | [112,152] | 112 | 152 | 否 |
|  | P2 | [56,56] | 56 | 56 | 是 |
|  | P7 | [56,56,] | 56 | 56 | 是 |
| P3 | [112,152] | 112 | 152 | 否 |
|  | P7 | [56,56] | 56 | 56 | 是 |
| P3 | [112,152] | 112 | 152 | 否 |
|  | P3 | [56,56] | 56 | 56 | 是 |

q = 1 时：

表4-4 第二次循环条件判别表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性值  VL | | startMaximumArrive |  |  | 是否  相等 |
|  | P2 | [56,56,56] | 56 | 56 | 是 |
| P3 | [112,152,152] | 152 | 152 | 是 |
|  | P2 | [56,56,56] | 56 | 56 | 是 |
|  | P7 | [56,56,56] | 56 | 56 | 是 |
| P3 | [112,152,152] | 152 | 152 | 是 |
|  | P7 | [56,56,56] | 56 | 56 | 是 |
| P3 | [112,152,152] | 152 | 152 | 是 |
|  | P3 | [56,56,56] | 56 | 56 | 是 |

所以，两次循环之后循环的终止条件满足，循环结束。在实现的分析软件中，得到时延分析报告的应用轨迹法得到的结果为：



图4-9 实例分析报告

4.4轨迹法的改进

4.4.1实例的模型检验分析

本节应用与3.4节相同的数据实例，应用模型检验方法进行如下分析。

其中，模型检验方法是指对于小规模的网络拓扑，对所研究的数据流中的包，在传输过程中所可能经历的场景都进行考虑，找出最差的场景，计算端到端响应时延的方法[32]。

对虚链路的响应时间的计算过程，作如下推导：

在每个节点上的最长处理时间等于，其他的虚链路的最长处理时间相同。物理链路上的延迟等于技术时延16。

在输入节点，即的输出端口p1，的处理不受到其他虚链路的影响，它所经历的时延为。在端口p2，的包在交换缓冲中的等待时间受到的包的影响。最差情景如图：

1

p1

p4

2

=1

1

p2

f(p2)=2

图4-10 p2上的最差情景

在p4的最差情景中，包1和包2同时到达，但是包1在包2之后接受处理，虚链路的包1经受的时延为40+40=80。

的包经历了*sw1*的输出端口上的最差情景后，经过16的物理链路时延，到达*sw3*的输出端口p3。由于交换机的调度采用了FIFO策略，的包1至少比的包晚一个的最差处理时间到达，所以的包2虽然来自同一个节点p2，不会直接延迟包1的处理。但是在最差情况下，会作为p3的忙时段的上处理的第一个包，影响其他在的包1之前到达的，且来自非虚链路或的包的处理，即满足p(p2) f(p3)。如图：

1

2

p2

3

4

p7

5

p9

1

p3

4

5

3

图4-11 p3上的最差情景

的包在p3上位于忙时段的末尾，忙时段处理的第一个包为来自节点p7的包4。在最差情况下，包4和来自p9的包5同时到达p3，并开始处理。包1和与4来自同一个节点的包3同时到达，但根据FIFO策略，包5先被处理；同时，结合最差情况的考虑，包1在包3之后被处理。于是，如图包1在4、5、3之后接受处理，考虑包1的到达时间和包4的到达时间，包1在p3上经历的延迟为40 + 40 + 40 + 40 – 40 = 120。

于是，的包在经历了p1、p2、p3上的等待时延和处理调度，以及物理链路上的时延后，得到的最差端到端响应时间为 40 + 16 + 80 + 16 + 120 = 272。

类似的，可以得到其他4条虚链路在模型检验下得到的时延：

表4-5 初始化虚链路的属性值

|  |  |
| --- | --- |
| 虚链路 | 模型检验 |
| VL(1) | 272 |
| VL(2) | 192 |
| VL(3) | 232 |
| VL(4) | 232 |
| VL(5) | 176 |

4.4.2结果比较与算法改进

对比4.4.1节模型检验分析与实际轨迹法的代码实现所得结果，可见对于虚链路VL(1)、VL(3)、VL(4)、VL(5)在模型检验中，所得的确切的时延上限，明显要比在轨迹法的代码实现下所得的时延上限要紧凑。仔细分析，发现VL(1)、VL(3)、VL(4)、VL(5)的时延上限结果要比在实际算法实现中的结果小40。而模型检验方法所得结果，是在对简单的网络拓扑，对所有的有限的状态，进行可能的有限分析后，得到时延上限结果。即在上述简单拓扑中，模型检验方法是根据所给参数，计算得出的精确结果。

以VL(1)为例，对两种方法的时延结果得出的分析过程，进行比较，可以发现：对于VL(1)，在节点P1和节点P2上，虚链路的最差端到端响应时间都分别相等，等于0和96。但是在P3上，两种方法对VL(3)和VL(4)所产生的影响的分析结果不同，分别使时延上限增加了176和216，相差40。

在基本轨迹法的算法实现中，在步骤③的第一个循环内，要对与直接相交的虚链路的对包的处理时间进行累加，包括虚链路、、、，共经历延迟160。其中，、分别要对的传输响应时间产生一个包的处理时间的延迟，共计两个包的处理延迟，即80。

在模型检验的方法中，根据图4-8相关的时延内容的分析，包1在p3上经历的延迟为40 + 40 + 40 + 40 – 40 = 120，其中，在计算的包1受到的、的时延影响时，以包3为时间原点，还要减除40的包1的到达时间，即包1必须在包4之后到达，而不能和包4同时到达。这样，、对包1产生的影响仅限40，达不到80。

3

4

p7

1

4

5

3

p3

图4-12

事实上，如图4-10，包1不可能同包4同时到达，因为包4和包3来自同一个前一个节点P7，包4和包3经过同一条物理链路到达P3节点。这样包4和包3在P7上的到达时间相差至少为40。根据FIFO的调度策略，如果包1和包4同时到达，包1必须在包3之前被处理。即包1处理时，到达的包3的处理还没有开始，从而包3就不可能存在于P3上的对于包1的忙时段内，不能对包1的传输产生延迟影响。

根据上述分析，需要对公式(4-1)进行改进，因为假设(4-0)不是在所有情况下都能成立。

在上述分析中，针对来自同一物理链路的包序列，后续的包与第一个包之间的到达必然存在一定的时间间隔。根据FIFO策略，所研究的包只能受到一个包的延迟影响，具体分析如下：

对于所研究的包m，如果包m在第一个包处理之后到达，不失一般性地，包m和第二个包同时到达，则第一个包在包m到达时已经处理完成，因为根据AFDX严格的协议约定，网络中不存在拥塞的状况，不可能存在来自上一个节点的包在下一个节点由于处理时间过长，被来自自身虚链路的包阻塞。所以必然已经处理完成，从而不在m的忙时段内。

如图4-11所示：

p

2

1

m

h-1

m

2

1

h

m所在忙时段

图4-13

如果包与第一个包同时到达，则第二个以及之后的包也不再忙时段内。因为包m要比来自同一物理链路的且在之后的包早到达，在FIFO策略下，包m在这些包之前被处理，从而在包m处理完之后的时间点，满足“在m之前到达的包都已经处理”。从而对于m对应的优先级的忙时段已经结束，在FIFO策略下，就是对于m所对应的忙时段已经结束。所以之后处理的包，包括之后来自同一物理链路的包都不在这个忙时段内。

示意图如下：

p

2

1

m

h-1

2

m

1

h

m所在忙时段

图4-14

所以的条件不一定成立，假设 (4-0) 就不一定成立，，所以要对公式 (4-8) 进行改进。

根据上述分析，要在公式 (4-8) 中，减掉来自同一物理链路的其他包的处理时延，只留下在该节点的一个处理时间最长的包的对应的处理时间即可，改进后的公式如 (4-10)。

(4-10)

对于所附加的条件，指到达，且来自同一条物理链路的所有虚链路对各自包处理时间的集合。公式在原有的处理时间中，找出最长的处理时间，丢弃了其他的处理时间，得到的时延上限更为紧凑。

应用上述改进后的轨迹法，得到分析报告如下：



图4-15 实例分析报告

这样，在3.7节的简单网络模型中，得到的分析结果，就和经过有限次的对所有状态的分析之后，利用模型检验方法得到的时延分析结果相同（并不是对于所有的网络拓扑模型都能应用模型检验的方法，也不能保证改进后的结果都是准确地等于模型检验所得的结果）。

在软件的最终实现中，均采用改进后的轨迹法。

第五章 实例分析与验证

5.1 工业数据实例测试与验证

5.1.1网络拓扑设计数据

如图5-1所示，参照工业实际网络配置数据，我们设计以下网络拓扑：共计部署了37台AFDX端系统，10台AFDX交换机，并在网络拓扑中静态定义了10条虚链路。各个交换机均是工作在100M线速处理模式下，网络中使用的是100M的连接，共计连接51条物理链路。

在分析软件中，设计相关网络拓扑，进行端系统、交换机和虚链路的配置：

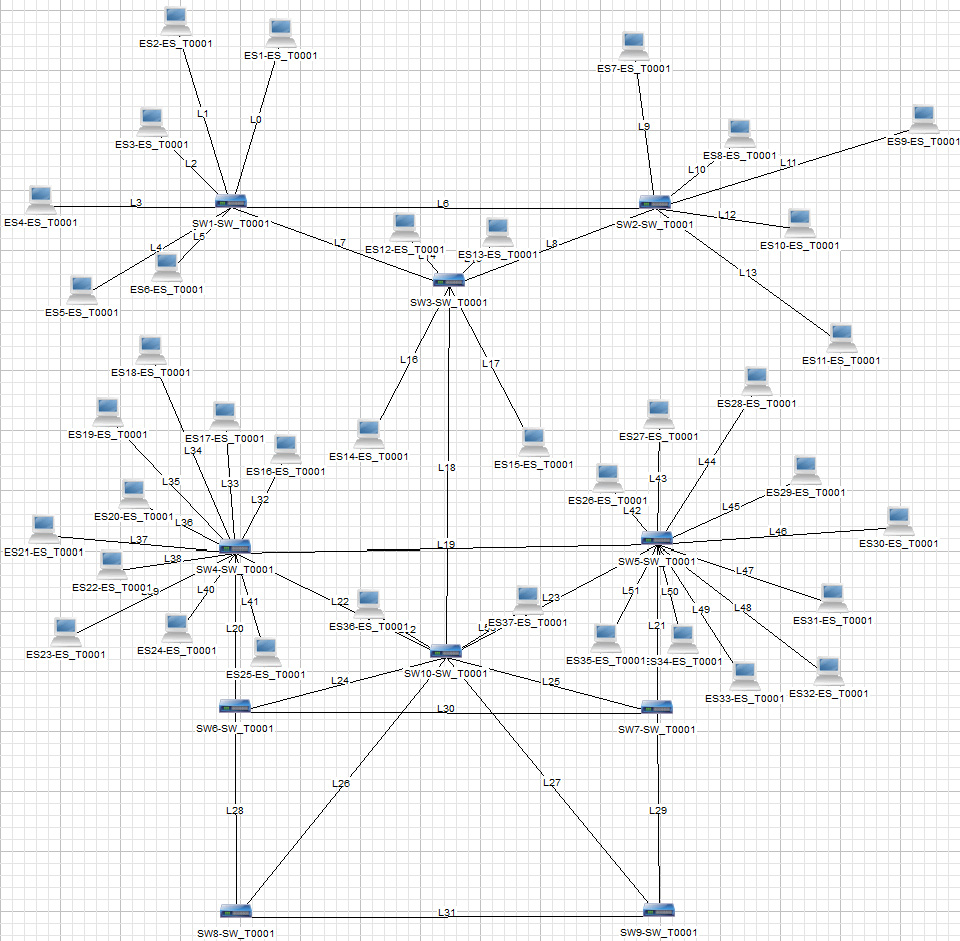


图5-1 仿真拓扑设计

对于静态的虚链路的设计，各虚链路路径定义为：

表5-1 仿真测试虚链路路径配置

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VL(ID) | 路径 | | | | |
| VL(1) | ES1-ES\_T0001 | SW1-SW\_T0001 | SW3-SW\_T0001 | SW2-SW\_T0001 | ES7-ES\_T0001 |
| VL(2) | ES3-ES\_T0001 | SW1-SW\_T0001 | SW6-SW\_T0001 | SW2-SW\_T0001 | ES8-ES\_T0001 |
| VL(3) | ES4-ES\_T0001 | SW1-SW\_T0001 | SW3-SW\_T0001 | SW2-SW\_T0001 | ES10-ES\_T0001 |
| VL(4) | ES16-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES25-ES\_T0001 |  |
| VL(5) | ES17-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES26-ES\_T0001 |  |
| VL(6) | ES18-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES27-ES\_T0001 |  |
| VL(7) | ES19-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES28-ES\_T0001 |  |
| VL(8) | ES23-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES31-ES\_T0001 |  |
| VL(9) | ES24-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES32-ES\_T0001 |  |
| VL(10) | ES20-ES\_T0001 | SW4-SW\_T0001 | SW5-SW\_T0001 | ES29-ES\_T0001 |  |

根据虚链路所承载的业务属性，即根据需求得到最大最小帧长值，在分析工具中，最大、最小帧长的配置数据如表5-3，用于计算时延上限，生成分析报告，评估网络部署的可行性：

表5-2设计虚链路的相关属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性  虚链路 | 最大帧长(byte) | 最小帧长(byte) | BAG(m) | 设计  上限() |
| VL1 | 156 | 64 | 32 | 800 |
| VL2 | 1436 | 64 | 64 | 1300 |
| VL3 | 1436 | 64 | 64 | 1300 |
| VL4 | 1436 | 64 | 64 | 1000 |
| VL5 | 1436 | 64 | 8 | 1000 |
| VL6 | 1436 | 64 | 32 | 1000 |
| VL7 | 1436 | 64 | 64 | 1000 |
| VL8 | 1436 | 64 | 64 | 1000 |
| VL9 | 1218 | 64 | 8 | 1000 |
| VL10 | 1408 | 64 | 8 | 1000 |

利用本设计开发的分析工具，按照两种算法计算得出时延上限，时延分析报告如下表：

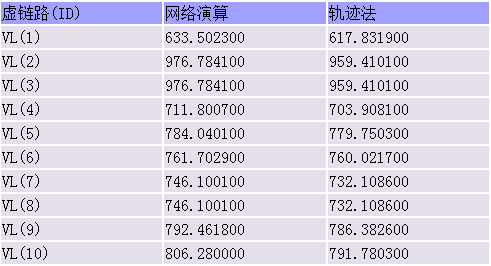


图5-2 分析工具时延分析报告

5.1.2仿真测试流程与平台配置

通过AFDX仿真测试平台，每个虚链路按照自身的BAG属性调整发送速率，得到数据包在网络中传输的端到端时延数据。在仿真测试过程中，共计发送7180个分组，统计所得数据，得到每条虚链路的响应时间上限的平均值。在仿真平台上，部署相关的端系统和交换机，对应属性如下：

表5-3 仿真测试端系统及交换机配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 主机端 | | | 交换机 | |
| 处理器 | 操作系统 | AFDX-ES  芯片核心 | 端口数 | 线速处理 |
| PowerPC | VxWorks | ARM9 | 24 | 是 |

在仿真平台测试中，要发送大量分组，所发送的分组采用的相关配置如表5-4所示，并记录测试所得的最大时延值：

表5-4仿真测试虚链路的发包配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 属性  虚链路的  数据包 | 帧长(byte) | BAG  (m) | 中间结果  有效位数 | 拟发送  分组数 |
| VL1 | 148、152、156 | 32 | 14 | 718 |
| VL2 | 1422、1430、1436 | 64 | 14 | 718 |
| VL3 | 1422、1430、1436 | 64 | 14 | 718 |
| VL4 | 1422、1430、1436 | 64 | 14 | 718 |
| VL5 | 1422、1430、1436 | 8 | 14 | 718 |
| VL6 | 1422、1430、1436 | 32 | 14 | 718 |
| VL7 | 1422、1430、1436 | 64 | 14 | 718 |
| VL8 | 1422、1430、1436 | 64 | 14 | 718 |
| VL9 | 1422、1430、1436 | 8 | 14 | 718 |
| VL10 | 1422、1430、1436 | 32 | 14 | 718 |

5.1.3仿真测试结果

表5-5仿真测试结果

|  |  |
| --- | --- |
| 虚链路 | 最大时延结果() |
| VL1 | 561 |
| VL2 | 850 |
| VL3 | 733 |
| VL4 | 581 |
| VL5 | 701 |
| VL6 | 553 |
| VL7 | 424 |
| VL8 | 602 |
| VL9 | 773 |
| VL10 | 681 |

从表5-5的仿真测试结果，可以看出，通过仿真测试平台，利用发送端和接收端测试程序，计算得出的结果均符合设计上限，同时也在有分析工具计算得出的最大时延的范围内。时延结果有一定的规律性，分析如下：测试的VL2和VL3经过的网络节点较其他虚链路多（比VL4~VL10多出一个交换机），所以在设计网络拓扑时，根据分析工具得出的结果，以及在仿真测试中，得到的结果，比平均值都要高。

第六章 结论

6.1 工作总结

本论文实现了AFDX时延分析工具的算法模块，对网络演算和轨迹法做了详尽的描述，并结合AFDX网络的实际特性，对理论模型做了简化和改进，对算法从基础的理论到最终结合工业实际参数的实现，做了逐步分析和验证。最后还借助AFDX仿真测试平台，做了验证测试，对算法的正确性和结果的规律性，作了分析。对于轨迹法，本论文指出了基本轨迹法对于来自同一条物理链路的包的处理上，存在的缺陷，并提出了一种改进的方法。

具体工作如下：

(1)根据网络演算的基础理论，构建了流量模型，并定义了与AFDX网络相关的参数，对网络演算的算法流程进行设计，在简单实例中分析算法执行流程，做了代码实现。

(2)将基本轨迹法引入AFDX网络模型，构建了轨迹法模型与AFDX网络模型相关量之间的联系，将流和节点的概念与AFDX中虚链路和端口的概念关联起来，设计了基本轨迹法实现中的迭代过程和其他流程，在分析软件中，实现了该算法，并对执行数据做了分析。同时，还在分析了相关数据后，指出了基本轨迹法的缺陷，针对多个虚链路的包，通过同一条物理链路到达节点的情况，改进了算法公式，附加了限制条件。并对改进后的轨迹法所得结果，与原先结果做了比较验证。

(3)在仿真测试平台上，利用分析软件验证了设计的可行性，还定义了测试所用数据包的相关参数，网络中虚链路、交换机等的配置状况，做了大量的数据测试后，得到时延结果，与分析软件的分析报告做了比对，验证得实际测试结果与算法所得结果基本相符。

6.2成果展望

由于时间有限，测试环境和条件也受到诸多限制，本设计所完成的仿真测试数量还不够多，定义的虚链路数量还不能完全同实际的工业拓扑配置对应。对于基本的网络演算方法，本设计只做了基本实现，该算法在AFDX拓扑配置参数下，有较大改进的空间，改进工作还要进一步实现。本设计所实现的时延分析工具在大规模拓扑和更为复杂的网络环境下，还有进一步增加配置变量，优化算法实现的可能，相关内容还需要进一步深入研究和在软件实现中完成。

参考文献

[1]支超有，李振水等.大飞机先进机载数据总线AFDX[C].大型飞机关键技术高层论坛暨中国航空学会2007年年会论文集，深圳,2008.

[2][Brajou, F.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Brajou,%20F..QT.&searchWithin=p_Author_Ids:37424237000&newsearch=true) [Ricco, P.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Ricco,%20P..QT.&searchWithin=p_Author_Ids:37424237200&newsearch=true) AFDX-Based Flight Test Computer Concept[J]. Instrumentation & Measurement Magazine, IEEE, August. 2005, 8(3):55-58.

[3]ARINC664 P7. Aircraft Data Network, Part 7-Avionics Full Duplex Switched Ethernet Network[S]. June 27, 2005.

[4]窦文华.高可靠千兆以太网交换机研究[D].长沙:国防科学技术大学，2008.

[5]石改辉，张原，李达.全双工交换式以太网研究[J].通信技术，2007(5):51-52.

[6]陈昕，杨杰，周拥军.航空全双工交换以太网冗余管理机制研究[J].计算机工程与应用，2009，45(2):102-105.

[7]华军.以太网在航空实时通信系统中的应用研究[D].上海:上海交通大学，2006.

[8]陈娟娟.以太网交换机集群管理功能的设计及实现[D].南京:南京航空航天大学，2006.

[9]R. L. Cruz. A Calculus for Network Delay. Part I: Network Elements in Isolation, Part II: Network Analysis[J]. IEEE Transactions on Information Theory, Jan. 1991, 37(1): 114-141.

[10]J.-Y. Le Boudec and P. Thiran. [Network Calculus: A Theory of Deterministic Queuing Systems for the Internet](http://ica1www.epfl.ch/PS_files/NetCal.htm)[M]. Berlin: Springer. 2001, 5-78.

[11]Hussein Charara, Jean-Luc Scharbarg, Jerome Ermont, et al. Methods for bounding end-to-end delays on an AFDX network[C]. Proceedings of the 18th Euromicro Conference on Real-Time Systems. Germany: IEEE Press, 2006: 1-4.

[12]杨云，熊华钢.计算AFDX延迟的网络演算方法[J].电光与控制，2008，15(9):57-60.

[13]S. Martin and P. Minet. Schedulability analysis of flows scheduled with FIFO: Application to the Epedited Forwarding class[C]. Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International Conference. Rhodes Island: IEEE Press, 2006: 1-7.

[14]Steven Martin. Pascale Minet. Improving the Analysis of Distributed Non-Preemptive FP/DP\* Scheduling with the Trajectory Approach[J]. Telecommunication Systems, 2005, 30(1-3): 49-79.

[15]Steven Martin. Pascale Minet. Worst case end-to-end response times for non-preemptive FP/DP\* scheduling[R]. France: Inria, 2005: 1-29.

[16]H. Bauer. JL Scharbarg and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network[C]. Emerging Technologies & Factory Automation, 2009. ETFA 2009. Mallorca Spain: IEEE Press, 2009: 1-8.

[17]H. Bauer. JL Scharbarg and C. Fraboul. Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach[J]. IEEE Transactions on Industrial Informatics, 2010, 6(4): 521-533.

[18]Luxi Zhao, Huagang Xiong, Qiao Li, Feng He. Using memo recursive computation in the Trajectory approach for the worst-case delay analysis of AFDX networks[C]. Electrical and Control Engineering(ICECE), 2011 International Conference. Yichang, China: IEEE Press, 2011: 5633-5638.

[19]张信明，陈国良.基于网络演算的流量整形模型[J].软件学报，2002，13(12): 2225-2230.

[20]王子君，许维胜等.控制网络的确定性延迟演算理论研究[J].电子学报，2006，34(2):380-384.

[21]徐科华.AFDX总线网络数据传输分析[J].民用飞机设计与研究, 2006(3):35-40.

[22]Steven Martina, Pascale Minetb and Laurent Georgec. Improving fixed priority schedulability with dynamic priority as secondary criterion[J]. Journal of Embedded Computing, 2006, 2(3): 327-345.

[23]田靖，田泽.AFDX-ES SoC虚拟仿真平台的构建与应用[J].计算机技术与发展，2010，8(20):192-194.

[24]王鹏.[AFDX交换机及交换芯片中关键模块的设计](http://www.cnki.net/kcms/detail/detail.aspx?filename=2009031201.nh&dbcode=CMFD&dbname=CMFD2009)[D].西安:西安电子科技大学，2008.

[25]王锦，张奕楠等.AFDX的分布式仿真[J].电光与控制，2008，15(8):76-80.

[26]Basu, Ananda, et al. Verification of an AFDX Infrastructure using Simulations and Probabilities[C]. First International Conference, RV 2010. St. Julians, Malta:Springer Berlin Heidelberg, 2010:330-344.

[27]张奇智，张彬，张卫东.基于网络演算计算交换式工业以太网中的最大时延[J].控制与决策，2005，20(1):117-120.

[28]Jean-Yves Le Boudec, Application of Network Calculus to Guaranteed Service Networks[J]. IEEE Transactions on Information Theory, May. 1998, 44(3):1087-1096.

[29]Lorne Mason. Tadeusz Drwiega. James Yan. Network Calculus Delay Bounds in Queueing Networks with Exact Solutions[C]. 20th International Teletraffic Congress, ITC20 2007. Ottawa, Canada: Springer Berlin Heidelberg, 2007:495-506.

[30]Anand, Madhukar, et al. Formal Modeling and Analysis of the AFDX Frame Management Design[C]. Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006. Ninth IEEE International Symposium. Gyeongju, South Korea: IEEE Press, 2006:393-399.

[31]郭利峰，王勇等.AFDX交换机的队列整形调度研究[J].计算机工程，2011，37(24):58-60.

[32]Bérard, Béatrice, et al. Systems and software verification: model-checking techniques and tools[M]. Berlin: Springer. 2010, 5-8.

致 谢

经过6个多月的努力，我的论文设计终于算完成了。在这期间，我得到了我的指导老师张涛老师的大力帮助和支持，在一些理论问题上，给了我不少的指导和启迪，我们项目的如期开展离不开指导老师的鞭策和鼓励。同时我们项目组的邱权、李萌同学也给了我很多的建议，我们相互学习，在软件实现过程中，共同解决了诸多难题，他们也为项目付出很多汗水。

在这期间，我们花了很多时间编写测试程序，查阅了大量资料，学院提供了良好的机房环境，学校图书馆的在线系统也给予了很大帮助。

同时，我的舍友高靖同学，在论文的写作方面给了不少意见，我们相互鞭策，最终完成了论文的撰写工作。

感谢336实验室的各位同学，他们忙碌的身影、简短而精粹的讨论，给实验室营造了良好的学习氛围，成为我撰写论文的很好帮助。

还要感谢已经毕业的杜学智学长，他在我对论文相关内容一无所知的时候，给了我很多的指导，使我对毕业设计的完成充满自信。

最后，再次感谢学院的各位老师和诸位同学给予的莫大的帮助和鼓励！

毕业设计小结

毕业设计是一门课程，是学生向学者转变的第一步，是对大学四年所学知识的实际应用和总结，是巩固作者所学的专业基础知识和锻炼作者解决问题能力的良好途径。

毕设的工作量大，内容繁杂，需要合理地安排。从最初的选题，开题，相关领域背景知识的学习，网络演算和轨迹法理论算法的学习；到软件需求报告的文档撰写，在软件中实现AFDX网络基本遍历算法，两种时延分析算法，算法实现代码的单元测试；到软件模块的集成测试，在仿真平台上完成实例验证测试；进一步完善软件开发文档，和继续测试。所有这些步骤都需要合理的时间安排，并且在各个时间段做足检查工作，不断督促项目成员完成各阶段工作。

代码的编写也是一大挑战，我们共计编写了上万行代码，没有一个良好的设计是不行的，这还要感谢张涛老师的耐心指导和两位同学的宝贵建议，使得我们项目的程序在修改、测试和扩展等环节，都少去了不少麻烦。在最后的软件模块集成时，每个人写的代码多少有不同，这时统一的接口规范就很重要了，项目组成员的积极沟通，也给我们的软件的最终实现带来了很多益处。

在后期的论文撰写方面，大量的文献资料的检索和阅读工作占据了大部分时间，同时论文格式的要求对于我们这些初学者来说也是很大考验。毕业论文撰写让我们真正理解了做一名学者的基本要求，也明白了做学问是个要静下心来，细致地研究，谨慎地分析，大胆地推测，周密地总结的过程。在这期间，我和同学之间的关系更紧密了，大家有了更多的时间在一起相互学习，相互促进。

论文写作也是一个漫长的过程，在这个过程中，我是边写作边学习，不断积累经验。在其中借鉴别人的写作风格也不失为一种学习方法。将所研究的内容展现出来，解释清楚，写出具有价值的文章，也是对学生的一种考验。在这期间，指导老师也给予了很大帮助，对我们的论文提出了多处修改意见，使我们的论文写作思路更加清晰完整，对论文也做了不少补充。

总之，在毕业设计的整个过程中，我觉得受益匪浅，它给我们将所学应用实践提供了平台，还为我们向学者发展提供了良好的锻炼机会。“学无止境，上下求索”，我将把毕业设计这段宝贵经历，作为人生的一大财富，无论在求学还是工作中，都能不断汲取养分，不断提升自己。

附 录

void CAlgorithm::startTRAlgorithm(){

//轨迹法

std::vector<std::vector<CVirtualLink\*>\*> closureSet;

std::vector<BOOL> putSign(savedVLS.size());

for(int i = 0; i < putSign.size(); i++){

putSign[i] = FALSE;

}

resetAllVLPropertities();

for(int i = 0; i < savedVLS.size(); i++){

//如果未考虑该VL从它开始计算闭包

if(putSign[i] == FALSE){

std::vector<CVirtualLink\*>\* vec = new std::vector<CVirtualLink\*>;

vec->push\_back(savedVLS[i]);

putSign[i] = TRUE;

//填充闭包

makeClosure(vec, &putSign);

//将闭包放入闭包集合

closureSet.push\_back(vec);

}

}

for (int i = 0; i < closureSet.size(); i++){

closureSet[i]->at(0)->caculateDelay();

}

}

void CAlgorithm::iterateCaculateDelay(std::vector<CVirtualLink\*>& virtualLinksSet){

//迭代运算

int q = -1;

do

{

q = q + 1;

for(int vlKIndex = 0; vlKIndex <virtualLinksSet.size(); vlKIndex++){

CVirtualLink\* vlK = virtualLinksSet[vlKIndex];

for(int vlKNodeIndex= 0; vlKNodeIndex < vlK->nodeList.size();

vlKNodeIndex++){

//从第一个节点到最后一个节点vlK->caculateWorstBeginProcessTime(virtualLinksSet, q, vlKNodeIndex);

std::vector<CVirtualLink\*> VLJsVector;;

CPortEntity\* hNode = vlK->nodeList[vlKNodeIndex];

if(vlKNodeIndex == vlK->nodeList.size() - 1 || vlK->existVLJCrossedLastOrPreFirst(hNode,virtualLinksSet, VLJsVector)){

for(int vlJIndex = 0; vlJIndex < VLJsVector.size(); vlJIndex++){

CVirtualLink\* vlJ = VLJsVector[vlJIndex];

if(vlJ != vlK){

if(hNode == vlK->nodeList[vlK->preNodeIndex(vlK->firstCrossedNode(vlJ))]){

CNodeDelay\* preNodeDelay = vlK->delayList[vlK->nodeIndex(hNode)];

CNodeDelay\* nodeDelay = vlK->delayList[vlK->nodeIndex(vlK->firstCrossedNode(vlJ))];

double startMaximumArrive = 0;

startMaximumArrive = (preNodeDelay->worstBeginningProcessTime –

0) + vlK->delayList[vlK->nodeIndex(hNode)]->conductTime

+ CAlgorithm::Lmax;

if(nodeDelay->startMaximumArrive.size()-1 <= q){ nodeDelay->startMaximumArrive.push\_back(startMaximumArrive);

}

}

}

}

if(vlKNodeIndex == vlK->nodeList.size() - 1){

double db = (vlK->delayList[vlKNodeIndex]->worstBeginningProcessTime - 0)

+ vlK->delayList[vlKNodeIndex]->conductTime;

vlK->properties[VL\_TRDELAY] = DoubleToVart(db);

}

}

}

}

}while(!CAlgorithm::allStartMaximumArriveTimeRemainTheSame(q,virtualLinksSet));

}

void CVirtualLink::caculateWorstBeginProcessTime(std::vector<CVirtualLink\*>& VLsSet, int q, int nodeIndex){

//计算最晚开始处理时间

delayList[nodeIndex]->worstBeginningProcessTime = 0;

double sum = 0;

for (int n = 0; n < VLsSet.size(); n++){

if(VLsSet[n] != this && isCrossed(VLsSet[n])){

int slNodeIndex = VLsSet[n]->slowestNodeIndex(this, nodeIndex);

if(slNodeIndex == -1){

continue;

}

sum += ( 1 + (0 + int(A\_Jitter(q, VLsSet[n]) )

/ VartToInt(VLsSet[n]->properties[VL\_BAG])) )

\* VLsSet[n]->delayList[slNodeIndex]->conductTime;

}

}

delayList[nodeIndex]->worstBeginningProcessTime += sum;

delayList[nodeIndex]->worstBeginningProcessTime += (1 + ((0 + 0)/VartToInt(properties[VL\_BAG])))

\* delayList[slowestNodeIndex(nodeIndex)]->conductTime;

sum = 0;

for(int anotherNodeIndex = 0; anotherNodeIndex <= nodeIndex;

anotherNodeIndex++){

if(anotherNodeIndex != slowestNodeIndex(nodeIndex)){

double maximumProcessTime = 0;

for (int n = 0; n < VLsSet.size(); n++){

if(isCrossedAtNode(VLsSet[n], nodeList[anotherNodeIndex])){

if(VLsSet[n]->delayList[VLsSet[n]->nodeIndex(

nodeList[anotherNodeIndex])]->conductTime > maximumProcessTime){

maximumProcessTime =

VLsSet[n]->delayList[VLsSet[n]->nodeIndex(

nodeList[anotherNodeIndex])]->conductTime;

}

}

}

sum += maximumProcessTime;

}

}

delayList[nodeIndex]->worstBeginningProcessTime += sum;

delayList[nodeIndex]->worstBeginningProcessTime -=

delayList[nodeIndex]->conductTime;

delayList[nodeIndex]->worstBeginningProcessTime +=

nodeIndex \* CAlgorithm::Lmax;

}

BOOL CAlgorithm::allStartMaximumArriveTimeRemainTheSame(int q, std::vector<CVirtualLink\*>& VLsSet){

//最晚到达时间判别条件

BOOL res = TRUE;

for(int vlKIndex = 0; vlKIndex < VLsSet.size(); vlKIndex++){

CVirtualLink\* vlK = VLsSet[vlKIndex];

for(int vlJIndex = 0; vlJIndex < VLsSet.size(); vlJIndex++){

if(vlJIndex != vlKIndex && vlK->isCrossed(VLsSet[vlJIndex])){

CPortEntity\* firstCrossedNode =

vlK->firstCrossedNode(VLsSet[vlJIndex]);

if(vlK->preNodeIndex(firstCrossedNode) != -1){

int firstCrossedNodeIndex =

vlK->nodeIndex(firstCrossedNode);

if(vlK->delayList[firstCrossedNodeIndex]->

startMaximumArrive[q+1] !=

vlK->delayList[firstCrossedNodeIndex]

->startMaximumArrive[q]){

res = FALSE;

return res;

}

}

}

}

}

return res;

}