CRANFIELD UNIVERSITY

TAO CHEN

DEVELOPMENT AND SIMULATION OF HARD REAL-TIME SWITCHED-ETHERNET AVIONICS DATA NETWORK

SCHOOL OF ENGINEERING

MSc by Research Thesis

August 2011

Cranfield University

School of Engineering

MSc by Research Thesis

Academic Year 2010 - 2011

TAO CHEN

DEVELOPMENT AND SIMULATION OF HARD REAL-TIME SWITCHED-ETHERNET AVIONICS DATA NETWORK

Supervisor: HUAMIN JIA

August 2011

ABSTRACT

The computer and microelectronics technologies are developing very quickly nowadays. In the mean time, the modern integrated avionics systems are burgeoning unceasingly. The modern integrated modular architecture more and more requires the low-latency and reliable communication databus with the high bandwidth. The traditional avionics databus technology, such as ARINC429, can not provide enough high speed and size for data communication, and it is a problem to achieve transmission mission successfully between the advanced avionic devices with the sufficient bandwidth.

AFDX(Avionics Full Duplex Switched Ethernet) is a good solution for this problem, which is the high-speed full duplex switched avionic databus, on the basis of the Ethernet technology. AFDX can not only avoid Ethernet conflicts and collisions, but also increase transmission rate with a lower weigh of the databus. AFDX is now adopted by A380,B787 aircraft successfully.

The avionics data must be delivered punctualy and reliablely, so it is very essential to validate the real-time performance of AFDX during the design process. The simulation is a good method to acquire the network performance, but it only happends in some given set of scenarios, and it is impossible to consider every case. So a sophisticatd network performance method for the worst-case scenario with the pessimistic upper bound requires to be deduced. The avionic design engineers have launched many researches in the AFDX simulation and methods study. That is the goal that this thesis is aimming for.

The development of this project can been planned in the following two steps.

In the first step, a communication platform plans to be implemented to simulate the AFDX network in two versions – the RTAI realtime framework and Linux user space framework. Ultimately, these frameworks need to be integrated into net-ASS, which is an integrated simulation and assessment platform in the cranfield's lab.

The second step deduces an effective method to evaluate network performance, including three bounds(delay,backlog and output flow), based on the NC. It is called Network Calculus. It is an internet theory keeping the network system in determistic way. It is also used in communication queue management. This mathematics method is planed to be verified with simulation results from the AFDX simuation communication platform, in order to assure its validity and applicability.

All in all, the project aims to assess the performance of different network topologies in different avionic architectures, through the simulation and the mathematical assessment. The technologies used in this thesis benefit to find problems and faults in the beginning stage of the avionics architecture design in the industrial project, especially, in terms of guarantee the lossless service in avionics databus.

Keywords:

AFDX, VL, end-to-end latency, backlog, arrival curve, service curve, RTAI, RTnet, NC, out flow, ES, Lmax, BAG, RRS

ACKNOWLEDGEMENTS

Great thanks for my supervisor Dr. Huamin Jia. I have got huge loads of suggestions and instructions in the GDP and IRP process. Without him, I can not make the more and more progress.

My thanks are also given to the library staff for their very kindly help.

I am grateful for my classmates for their encouragements and supports. We finished our GDP successfully with a great team.

Thanks to my landlord Mike. He is a man with great kindness. His patience and helps make me feel like at home.

I also say thanks for AVIC and CFTE for providing me this precious opportunity.

It is impossible to finish my study without the supports from the numbers in my family, especially for my parents. They look after my son for one and a half year without any complain. I own my husband a debt for his continued supports for my study.

Thanks for everyone who have given me help during my illness period. I can not insist on finishing the course of study without them.

Love is great!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AFDX	Avionics Full Duplex Switched Ethernet
ASS	Assembler thread
BAG	Bandwidth Allocation Gap
COM	Communication
DIS	Dispatcher thread
ES	End system
FIFO	First in first out
FMS	Flight management system
Jitter	Delay in switch queue
IMA	Integrated module architecture
ID	Identification
Inf	Infimum
IP	Internet Protocol
Latency	End-to-end delay
Lmax	The maximum size of the message frame
net-ASS	Integrated simulation and assessment platform
NC	Network Calculus
NAV	Navigation
RTAI	Real time application interface

RRS	Round-robin scheduler
Rx	Receive
Tx	Transmit
Three bounds	Delay, backlog, output flow
UDP	User Datagram Protocol
VBR	Variable bandwidth rate
VL	Virtual link

NOMENCLATURE

$\lambda_R(t)$	Peak rate functions
δ_T	Burst delay functions
$\beta_{R,T}$	Rate-latency functions
$\gamma_{r,b}$	Affine functions
v_T	Step functions
$u_{T,r}$	Staircase functions
$\otimes$	Min-plus convolution
$\bar{f}$	Sub-additive closure
$\oslash$	Min-plus deconvolution
$v(f,g)$	The maximum vertical deviations
$h(f,g)$	The maximum horizontal deviations
α	Arrival curve
β	Serve curve
D	Delay bound
$R(t)$	Input cumulative function
$R^*(t)$	Output cumulative function
DF_{PX}	End-to-end latency
SDF_{px}	The delay in switches
px	VL path

1 Introduction

1.1 Background

Nowadays, It is commonly recognised by manufacturers that databuses, such as ARINC 429 or 629, are not enough capable for the high quality transmission in integrated avionics systems. Avionics Full Duplex Switched Ethernet(AFDX), defined by ARINC 664 part 7, seems like a good solution for this problem.

Avionics AFDX plays a fundamental role to build different sophisticated avionics databus system architectures. Avionics engineers provide detail design for various logical and physical configurations in system architectures, in terms of packets scheduling, redundancy management, end-to-end latency and backlog size, etc, which bring considerable challenges for avionics system engineers.

AFDX is based upon the Ethernet. There are no collisions with the full-duplex switched Ethernet instead of the half-duplex Ethernet, but packets may be transmitted latency (delay) due to so many factors, such as the multi flows congestion in one switch, and the delay in the worst situation may be over an upper limitation of AFDX transmission configuration. On the other hands, it is possible that a transmission buffer can be overflow theoretically. But if the buffer configuration (backlog) in avionics network systems is defined correctly, overflow can be successfully avoided, so important considers in databus design are deterministic performance, maximum backlog, lossless service and low latency time.

Some researches have been launched to highlight the importance of transmission performance issues in AFDX for the industries and government, In order to effectively and consistently guide databuses design process. Further avionics communication research also requires to be done on evaluation of the deterministic performance and architecture topologies in order to guarantee the lossless deterministic service. Basically, these researches require AFDX simulation platform for validation and testing. In this task, the achievement of target goes through two ways: the simulation development and mathematical

deduction. In conclusion, this project plans to benefit avionics databuses design process through simulation and mathematical ways during the beginning design period in effective ways, whether in academic or industrial area.

1.2 Research objectives

In this project, An AFDX simulation framework has been successfully built. Based on this platform, there are some studies for evaluation and simulation of distributed avionics databus network AFDX during the development and design process, such as performance analysis, especially in the data lossless assurance and transmission performance. In this thesis, aims of researches are listed below.

- To implement an AFDX simulation platform, it simulates the behaviour of the AFDX avionic network protocol in avionics topology structures. That can be applied for a wider integrated avionics communication simulation environment, and it can also be used to evaluate the deliver performance in avionics databus architecture design.

  The framework has two versions:

  (1) Linux user space version, which can be used on the Linux user space operation system. It is a normal Linux application, and not the hard real-time software. Its real-time performance is not very critical.

  (2) RTAI/RTnet version, which can be used on the RTAI/RTnet kernel operation system. It is a hard real-time switches Ethernet software. Its real-time performance is critical.

- To deduce a set of mathematical equations, based on the NC (Network Calculus) theory, for evaluating the network performance including three bounds (delay, backlog, output flow), in order to guarantee network lossless deterministic performance.

- To compare with simulation testing results for such performance parameters. The effectiveness and application value of mathematical methods would be verified.

The project was started by Domingo Diez Barrero in 2009, a Cranfield SOE student. From then on, two students did deep researches on his basement. They really did fabulous work for the AFDX simulation framework. However, there still exist some problems and errors in previous projects.

- They did the Linux user space AFDX simulation software with so many logic errors.
- They did not build a hard real-time simulation software.
- They did nothing about the research on network performance.

All of these objectives have been achieved in this project. The major contribution by author was to successfully achieve the framework as the two-version software running on the RTAI/RTnet and Linux, and successfully find a new way to evaluate network performance- three bounds.

1.3 Arrangement

In order to achieve these objectives listed above, the previous works done by previous students should be given a better understanding on it. We should be familiar with the lab circumstance and install the RTAI/RTnet successfully. We also spent some time in modifying codes in the previous software done by previous students, in order to correct the AFDX simulation framework behaviours.

The project starts form the May 2011 to the August 2011. It includes several different phases:

- The previous project study.
- The assessment of project status.

- Being familiar with the lab circumstance.
- The literature review and background study, including RTAI/RTnet, AFDX and NC theory.
- Implementation and design simulation program.
- Testing and debugging simulation program.
- Evaluating for the network performance-three bounds.
- The validation for methods and simulation testing.
- Writing documentations and the thesis.

1.4 Contents

This thesis contents are planed as follows: Chapter 2 introduces the AFDX and RTAI/RTnet basic knowledge. Chapter 3 shows the simulation software framework in the detail design. After describing the running and testing process, then gives simulation results. Chapter 4 illustrates the method for estimating network performance which is employed to calculate three bounds in the AFDX databus on the NC theory. The qualities of mathematic model are analyzed and try to compare with simulation results. Based on the requirements of the AFDX simulation project, the framework has been built for the simulating test. The messages transmission simulation tests and results are described clearly in Chapter 5. At last, Chapter 6 gives a conclusion for this project and some suggestions for the future work.

2 AFDX and RTAI Introduction

2.1 AFDX

The Avionics Full Duplex Switched Ethernet (AFDX) standard for the transmission of packets between Avionics computers and advanced devices is defined in the IEEE 802.3 and ARINC 664(Part 7), which is one of the electrical and protocol specifications. It has one thousand times speeds highly faster than ARINC 429, AFDX concept has been adopted by Airbus.

AFDX is based upon Ethernet. The reason for why it acts as such an attractive technology is that there are no collisions with full-duplex switched Ethernet instead of the half-duplex Ethernet, but packets may be transmitted delay due to multi flows congestion in one switch.

It is possible that a Tx or Rx buffer can be overflow theoretically, but if the buffer configuration in an avionics network system are defined correctly, overflow can be successfully avoided.

End systems and avionics subsystems

As Figure2-1 shows, an avionics subsystem collects data from controllers, sensors and actuators, then transmits the data to other subsystems through ES(End system), and the End system could also be used to connect the Internet through Gateway. The avionics computer systems are capable of containing several avionics subsystems adopting partitions technology.

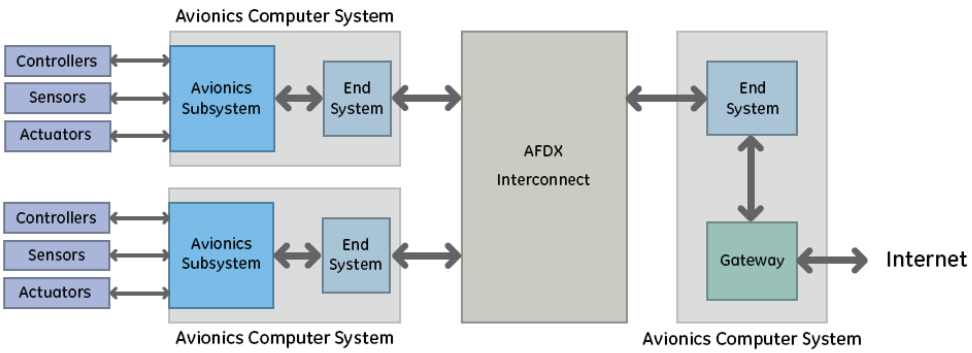

Figure 2-1 AFDX Network[8]

AFDX Communications Ports

Avionics subsystems send messages to each other through the communication ports. There are two types of communications ports: sampling and queuing ports, and detail information are described in ARINC 653.

In Figure2-2 and Figure2-3, the major difference between the sampling and queuing ports is mainly in reception. A sampling port stores a single message in a buffer, and this buffer will be overwritten by an arriving message[8]. Reading a message by application does not remove it from the buffer, and therefore it can be read several times. Due to this reason, each port must have a freshness indication. It could tell whether the Avionics subsystem is sending the same message repeatedly or not.

A queuing port has sufficient buffer to store a fixed number of messages, and new messages are putted to the queue. Reading and removing the message from the queuing port adheres to FIFO principles[8].

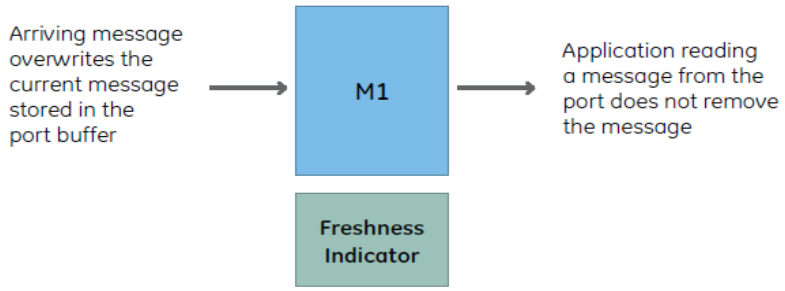

Figure 2-2 Sampling Port at Receiver[8]

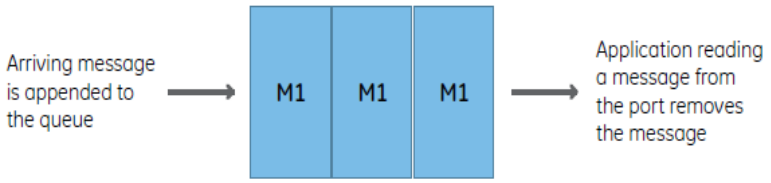

Figure 2-3 Queuing Port at Receiver[8]

Virtual Links: Packet Routing in AFDX

Figure2-4 is the format of the AFDX destination address. It includes a 16-bit unsigned value called a Virtual Link ID which is used to route the message in the network communication.

AFDX switches are configured to route an input message to one or more destination links. Ethernet frames associated with one Virtual Link ID must be one, and only one original end system, and one or more destination ports. The Figure2-5 shows an example.

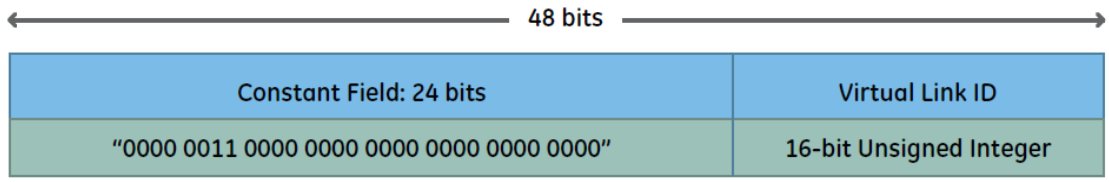

Figure 2-4 Format of Ethernet Destination Address in AFDX Network[8]

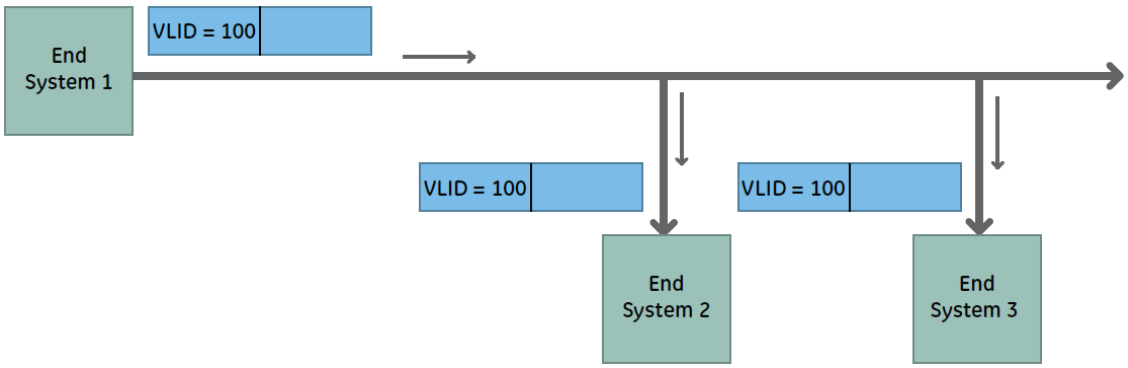

Figure 2-5 Packet Routing Example[8]

Figure 2-4 shows the formant of Ethernet Destination Address in AFDX Network, and Figure 2-5 means that one virtual link with ID=100 delivers messages form the End System1 to the two destination End System 2 and 3.

Message flows

Figure2-6 shows a message M being delivered from AFDX Port 1 in the avionics computer with End system1, then to Port5 in the End system2 and

Port6 in the End system 3. Figure2-7 shows the payload and header of the Ethernet frame. In the Ethernet payload, it includes a IP packet header and payload. The IP packet payload also has a UDP packet header and payload. The messages sent by the Avionics subsystem are contained in UDP packet payload.

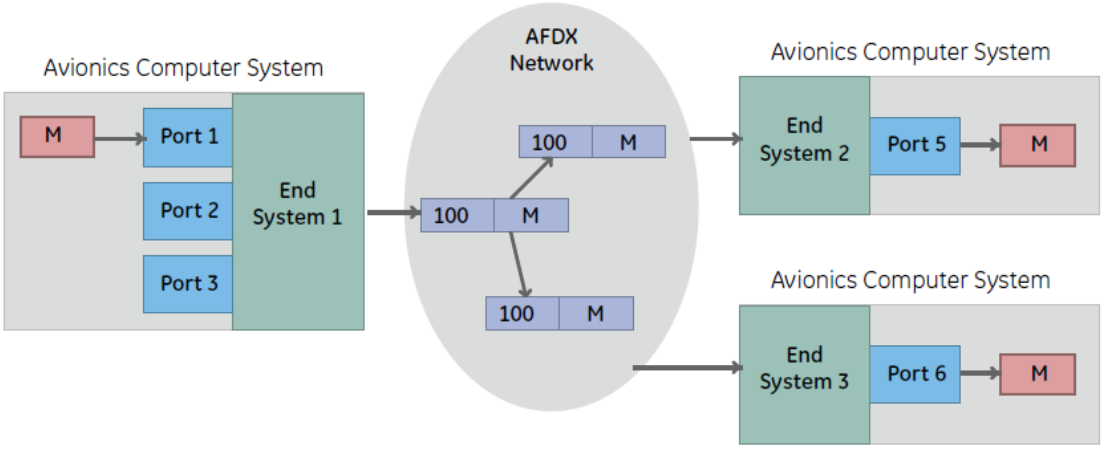

Figure 2-6 Messages Sent to Port 1 by the Avionics Subsystem[8]

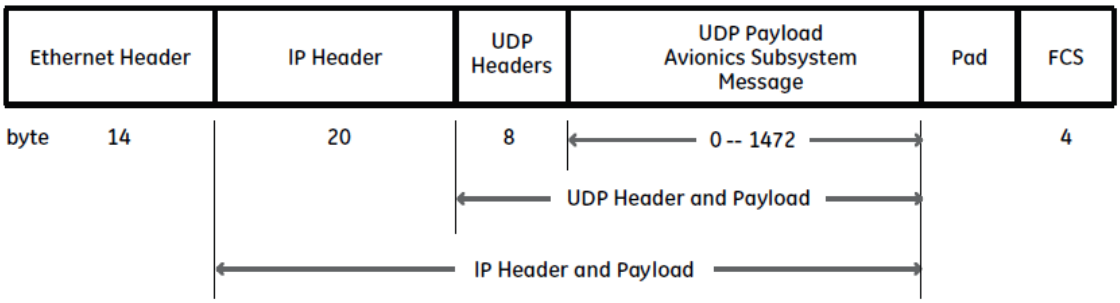

Figure 2-7 Ethernet Frame with IP and UDP Headers and Payloads[8]

Redundancy Management

AFDX systems have two independent switch networks, according to the ARINC 664, which are the A and B Networks. That is redundancy management. End System transmits each packet through both networks. Therefore, normally, each End System will get two copies of each message, showed in Figure2-8. End Systems need an approach to distinguish replicas that coming from the A and B networks.

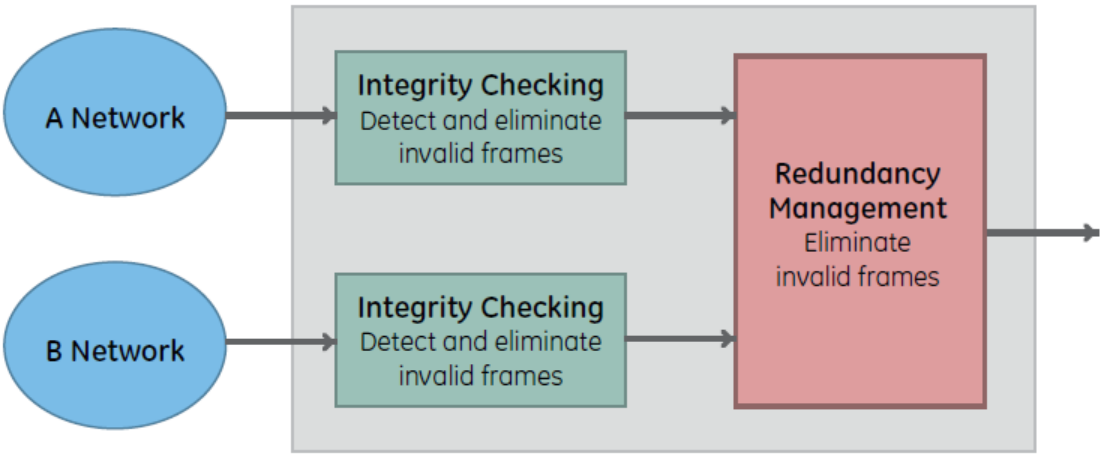

Figure 2-8 Receive Processing of Ethernet Frames[8]

Virtual Link Scheduling

Figure2-9 illustrates the Virtual Link Scheduling. The Virtual Link Scheduler controls each virtual link within its assigned bandwidth, BAG and Lmax limitation. It also introduces jitter for multiplexing within acceptable limitations. BAG is the maximum time interval between the first bit of frame in one particular Virtual link, and Lmax means the maximum size of the message frame.

The following formulas are maximum allowed jitter of each virtual:

$$\max\_jitter \le 40\mu s + \frac{\sum_{j\in\{set\ of\ VLs\}}(20 + L\max_j)\times 8)}{Nbw}$$

$$max\_jitter \le 500\mu s$$

N_{bw} is the link bandwidth (100 Mbps).

A virtual link can have more than one sub Virtual link. Figure2-10 shows three sub Virtual links in one virtual link. The Virtual Link selects packets from the sub-Virtual link queues in a round-robin manner.

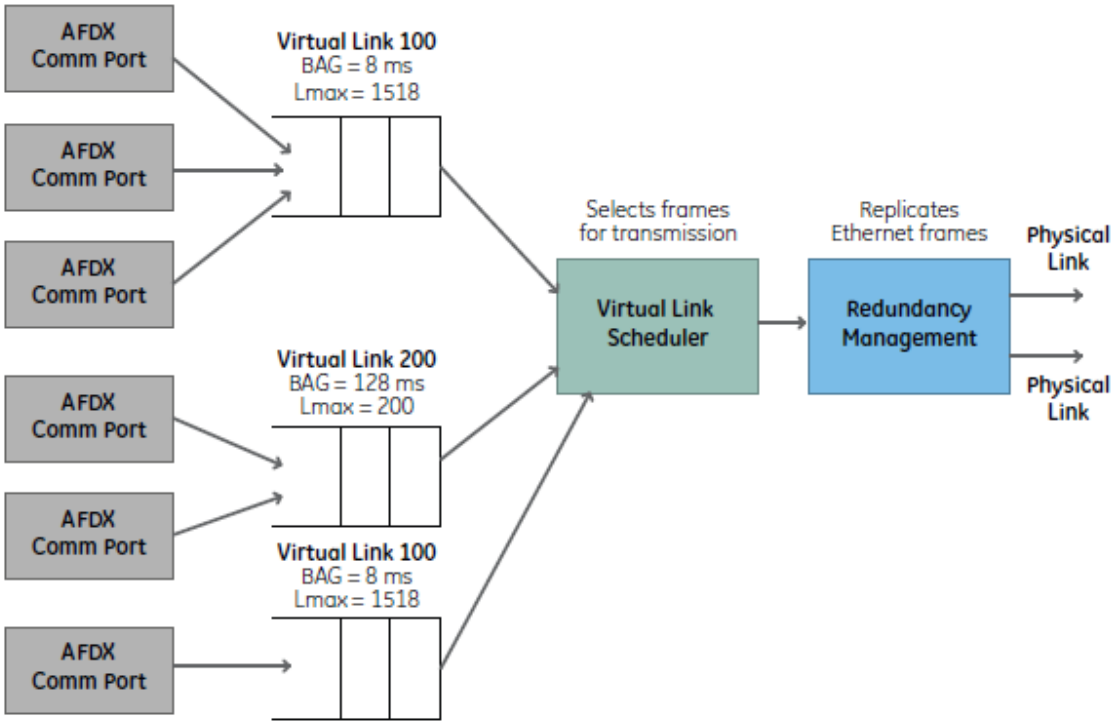

Figure 2-9 Virtual Link Scheduling[8]

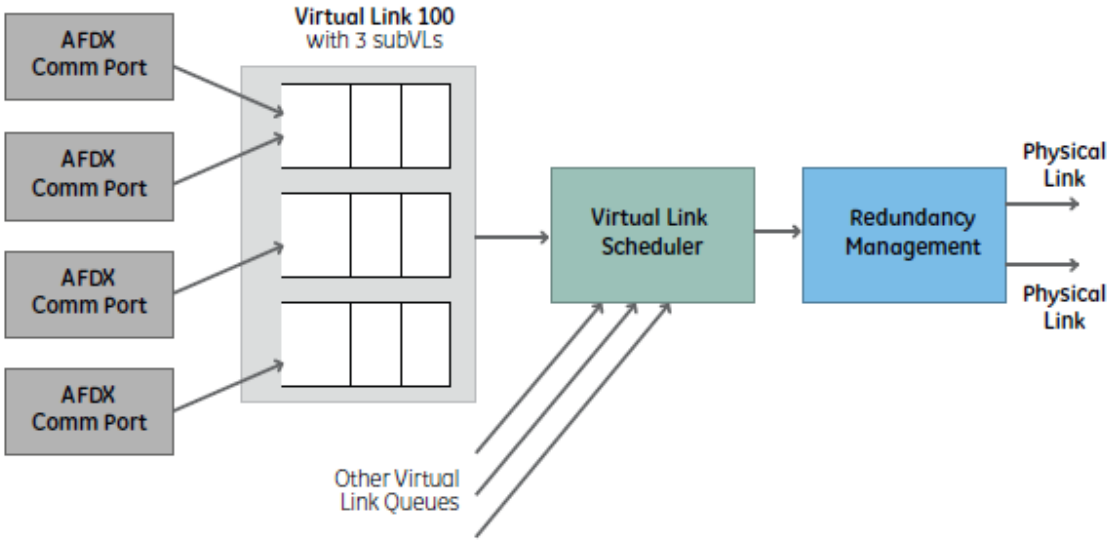

Figure 2-10 SubVirtual Link Scheduling[8]

Jitter

Figure2-11 shows the Virtual Link Regulators create output streams with zero-jitter. When the Regulator outputs two or more message into one Virtual Link Scheduler, it would introduce Jitters. Ethernet frames which arrive at input

buffer of the scheduler unit at the same time, would experience delay in a queue(jitter).

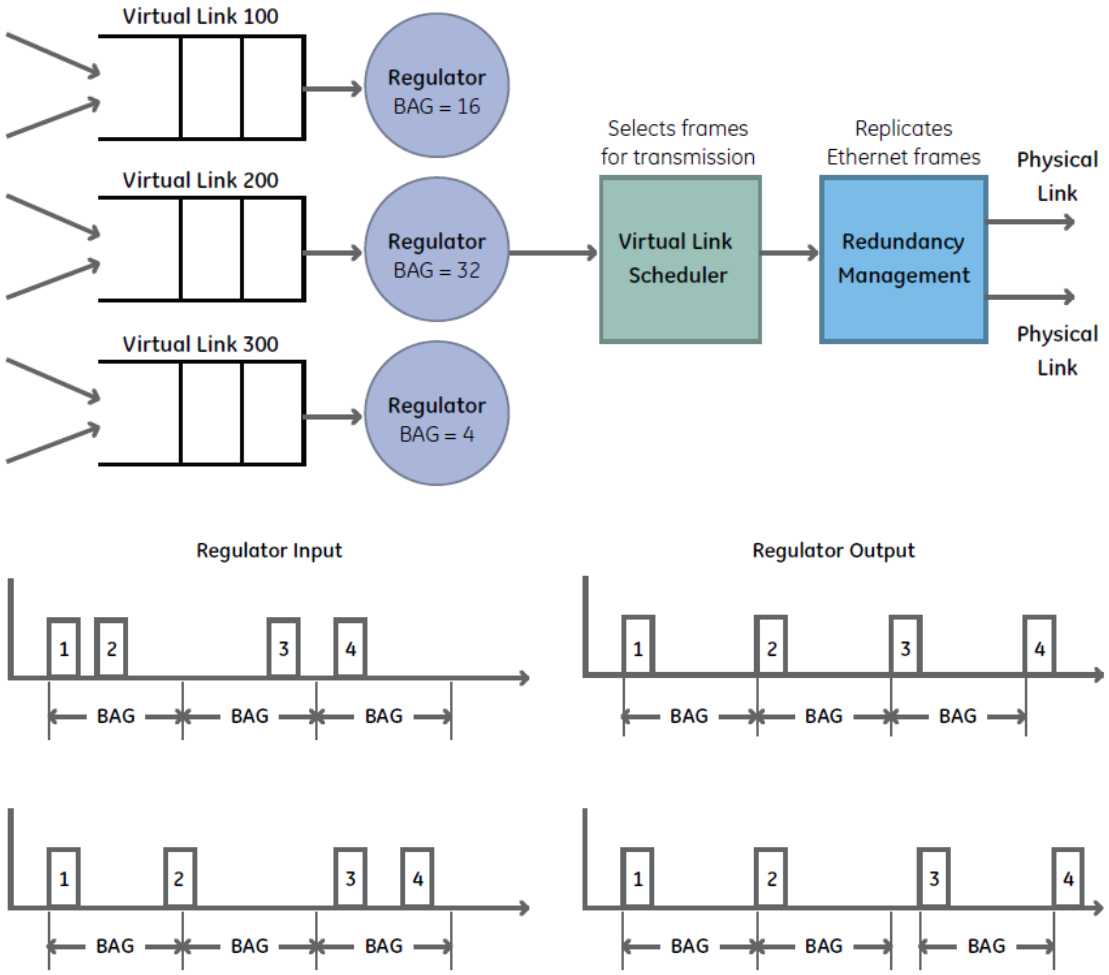

Figure 2-11 Role of virtual link regulation[8]

The AFDX Protocol Stack

This section describes the overall AFDX transmission and reception protocol stacks. The protocol layers are divided into four parts. Figure2-12 shows the AFDX Tx Protocol Stack process.

Reception is the reverse of transmission, showed in figure2-13.

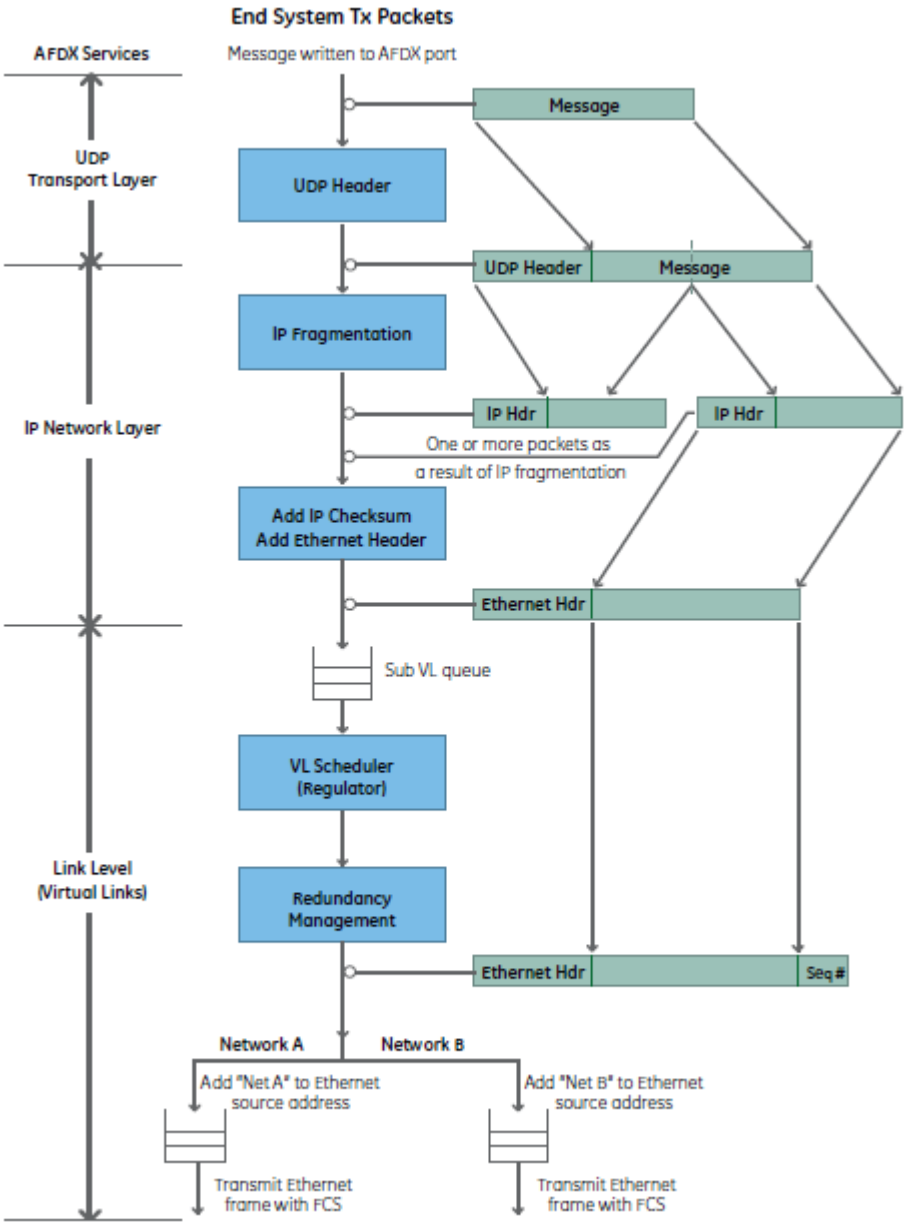

Figure 2-12 AFDX Tx Protocol Stack[8]

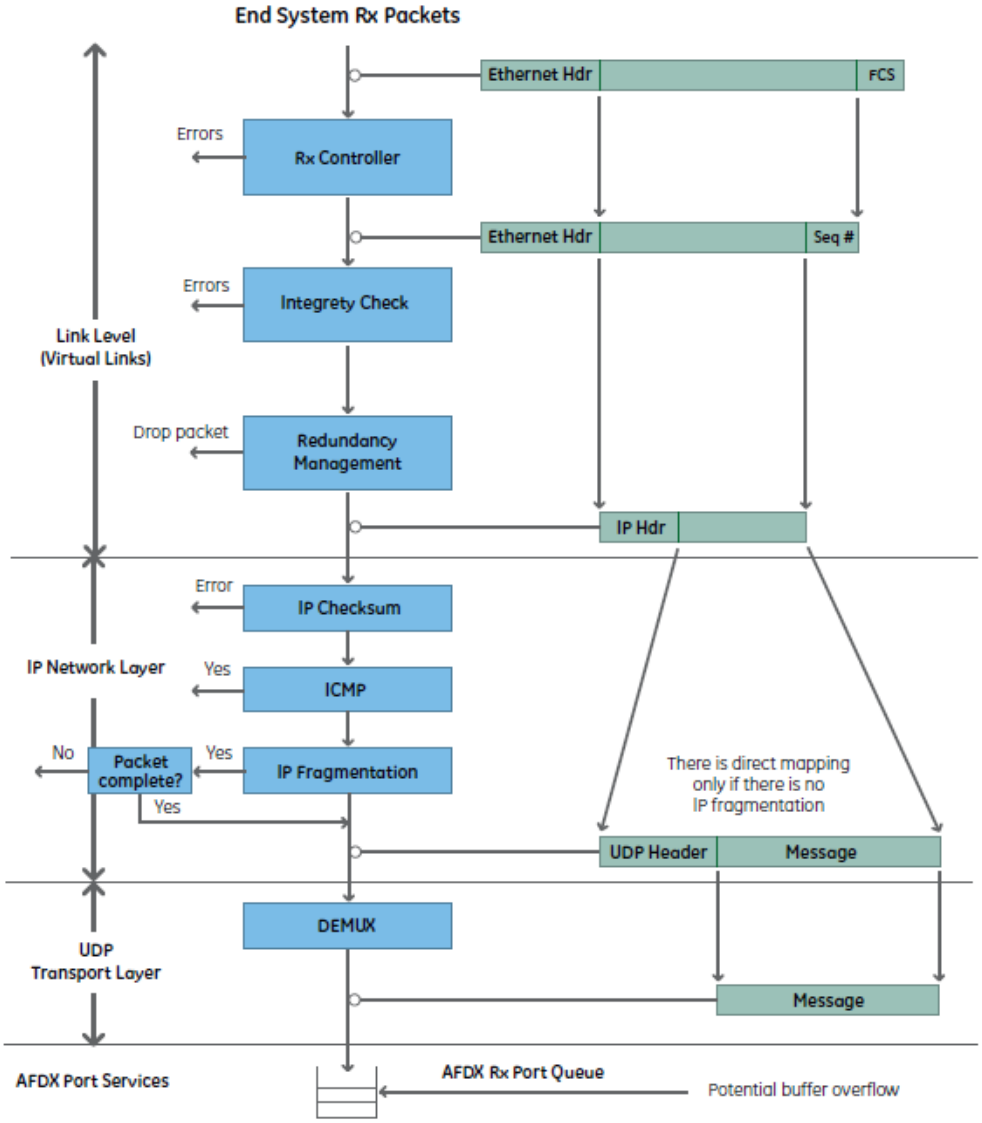

Figure 2-13 AFDX Rx Protocol Stack[8]

2.2 RTAI/RTnet and Linux

Real-Time Task

In military and industrial domains, the specific task to be performed requires varying level response of the real-time computer. As the results, there are two different types: soft or hard real time task.

Soft real-time: In soft real-time systems, the loss of an occasional data, in the soft real-time kernel system, cannot bring serious problems to the whole performance, and the average performance is still acceptable. The soft real-time applications period timer is not critical. It could be interrupted by the soft real time system events, and missing frame also could happen. However some techniques can compensate for missing data. The problem is that the compensating method can only derive the lost data, and it is not actual data, because the actual data have been missed.

Hard real-time: The hard real time systems guarantee system repeatable responses, which could be thousandths or millionths of a second. The hard real-time applications period timer is very critical. It can never be interrupted by the hard real time system events, and missing frame also could not happen. The hard real time system never compensate for the worst-case performance using the average case, and never to miss the control deadlines, so the hard real-time system is one of the most challenging technologies in the computer field.

The differences between two type tasks are not very apparent in only one-single-task system. However in multi-tasking operating systems, the demand for hard one becomes more and more inevitable, such as Linux.

Real-Time Operating System (RTOS) refers to the system that can apply two types real time performance. In other words, it could has hard real time performance or soft, and do not need both. A good example is Linux.

The Real-Time Linux Solution

The Linux kernel is scheduled by the Linux schedulers. The Linux task must be idle when real time applications are busy. That means, Linux applications only execute when real time applications are free. Interrupt can never be blocked by the Linux task. As the results, it is possible that the hardware can be controlled by the software simulation interrupt.

In Linux, the real-time system is always capable to respond the interrupt in any state: in kernel mode or a user process, disable or enable interrupts. It is

unnecessary for the real time kernel wait for any resources released by the Linux user space.

There exist two major hard real-time Linux systems: RTLinux and RTAI. RTLinux was launched at the New Mexico Institute of Technology, by Michael Barabanov under the direction of Professor Victor Yodaiken. Real-Time Application Interface (RTAI) was implemented at the Dipartimento di Ingeneria Aerospaziale, Politecnico di Milano by Professor Paolo Mantegazza.

The real time applications do not run on the user space. On the contrary, the real time modules run in the kernel address space. The kernel modules can be dynamically loaded into or removed from the real time kernel system.

RTAI Solution

RTAI has the preemptive and deterministic performance. It allows the use of all standard Linux functions, applications and drivers. RTAI has many features without compromising performance.

The list of RTAI features includes:

- POSIX 1003.1c compatibility.
- POSIX 1003.1b compatibility.
- RTOS IPCs(inter-process communications).
- Memory Dynamic Allocation.
- Binding PERL.
- interface /proc.
- LXRT.
- support FPU.
- scheduler.

RTAI can offer interrupt response of 20 μSec, 4μSec timer, and periodic tasks in 100KHz or one-shot task in 30KHz. The system hardware imposes the major limitation, but is not the real-time application software self [4].

LXRT

(1) overview

LXRT implements a real time application by calling RTAI functions in the Linux user space [8]. The applications could be launched as soft tasks with the memory protection from the normal user space. It is easy to transform a soft task into hard one after the functionality of application getting the developer's satisfactions.

(2) LXRT summary

The summary of LXRT is listed here:

LXRT provides applications the right to execute as the soft module in Linux:

- The Linux memory protection scheme protects tasks to execute successfully.
- It is easy to divide the task into hard and soft real-time components.
- Standard Linux debug tools can debug the LXRT task.
- After its successful debugging, LXRT can transform the task into the kernel space.

Difference between Linux and RTAI/RTnet programme interface

The differences between the Linux and RTAI/RTnet programme interface have been illustrated in Table2-1.

Table 2-1 Differences between Linux and RTAI/net programme interface

item	RTAI/net	Linux
Thread handle	int	pthread_t
Mutex	SEM	pthread_mutex_t
Semaphore	SEM	sem_t
Mailbox	rt_mailbox	-
Message queue	-	mqd_t

Receive mail	rt_mailbox()	-
Message queue	-	mq_receive()
Wait semaphore	rt_sem_wait()	sem_wait()
Post semaphore	rt_sem_signal()	sem_post()
Lock mutex	rt_sem_wait()	pthread_mutex_lock()
Unlock mutex	rt_sem_signal()	pthread_mutex_unlock()
Create socket	rt_dev_socket()	socket()
Close socket	rt_dev_close()	close()
Bind device	rt_dev_bind()	bind()
Send data	rt_dev_sendto()	sendto()
Recvfrom	rt_dev_recvfrom()	recvfrom()

RTAI/RTnet application circumstance configuration

The RTAI/RTnet application executing circumstance can be configured following the following steps. These are actual steps for RTAI/RTnet configuration before executing its applications:

- To unmount the linux network device.
- To mount the RTnet network device.
- To insert the RTAI modules.
- To insert the RTLX modules.
- To insert the RTnet modules.
- To start RTnet.
- To configure the RTnet device IP Address and switch them on.
- To rount and solicit the RTnet network computers.
- To use *rtping* test the communication connect status.

List2-1 gives an example of script configuration file.

```
#!/bin/sh

mknod /dev/rtnet C 10 240
ifconfig eth0 down
rmmod e100

insmod  ///rtai_hal.ko
insmod  ///rtai_lxrt.ko
insmod  ///rtai_sem.ko
insmod  ///rtai_rtdm.ko
insmod  ///rtai_mbx.ko

insmod   ///rtnet.ko
rtnet start

rtifconfig rtlo up 127.0.0.1
rtifconfig rteth0 up 192.168.1.1

rtroute add 192.168.1.8 00:D0:B7:07:23:20 dev rteth0
rtroute solicit 192.168.1.8 dev rteth0
```

List 2-1 Example of RTAI/net script configuration file

3 AFDX Simulation

3.1 Introduction

The main missions of the AFDX Simulation have two main parts:

- To develop the simulation framework on the real-time operation system to simulate the AFDX protocol, with common devices and normal software. The framework has two versions, one is a Linux user space software, and the other is a real-time RTAI software.

- The simulation platform could be eventually used into the net-Ass, which is a developing circumstance in the Dr. Huamin Jia's laboratory in the Cranfield University, supporting simulation tests for the network performance, which shall be also used as a communication platform for achieving avionics network functions in the further integrated avionics project researches.

The project has been developed by three previous Cranfield University research students. They have implemented the architecture and framework as the simulation software. They have done brilliant works, which have been mainly concluded below:

- Requirements gathering and analysis phase.

- A runnable version of the AFDX simulation platform running on the Linux operation circumstance.

- A simulation configuration: the way to configure the AFDX simulation circumstance context, specifically using the configuration file to assign the value of principal parameters, including the BAG, Lmax, etc.

- Achievement of AFDX databus partial functions which includes the address management, message traffic, VL scheduling, data dispatcher, sequence and the queue management.

- Database management, which is used to save and load the IP address configurations and port information in respective End System computers, according to the simulation databus architecture configuration.

- Process management, which can record the whole simulation process, in terms of errors, problems, warnings and success status information.

The achievement done by the previous students have been listed above. However there are several existing problems in these pervious simulation projects. They did a Linux user space AFDX simulation software with so many logic errors, and they did not build a hard real-time simulation software, and they did nothing about the research on network performance.

- No hard real-time simulation software could run on the RTAI/RTnet successfully, and the lab circumstance of hard real-time systems did not come up to expectations.

- The soft real-time simulation framework achieves major AFDX functions. However some important communication functions did not successfully achieve and transmission errors are still in the run-time simulation process, such as message confusion, queue management faults, so none of previous applications transmit the messages between End system computers correctly.

- No researches for the network performance and transmission testing.

All these problems listed above have been solved and achieved in this project, aiming to build two versions of the AFDX simulation platform with the same software architecture, and one is a set of normal software running on the Linux operation systems, the other is a real-time RTLX software running on the RTAI/RTnet background. All of them are capable for supporting network performance research on AFDX data bus. What this project did is mainly listed in Table 3-1.

Table 3-1 Differences between previous and this project

Objectives	Previous projects	This project
Linux simulation software	They did it with so many logic errors.	The simulation framework could execute successfully.
RTAI/RTnet simulation software	No	The hard real-time version framework has been successfully built.
Network performance analysis	No	Some equations have been deduced to evaluate the three bounds of network performance.

3.2 Requirement analysis

As discussed above, previous students have launched the analysis requirement for this research. The major requirements of this project are:

- The platform could simulate the AFDX databus behaviours.
- The simulation system could be achieved as a real-time program running on the RTAI.
- The framework can be achieved as a Linux standard user space program with the same architecture as the real time version.
- The network structure and topology could configure in the framework.
- The framework could be able to record the simulation process and results.
- The framework could support the research on the network performance.

- The framework could be used in the integrated avionics simulation system.
- This project could find a way to estimate the network performance of AFDX.

The detail of requirements for the simulation platform can be found in [9]. This chapter only lists some major selection of requirements in Table3-2, together with the code compliant state, and integrated with the status of development of these requirements.

Table 3-2 Platform functions requirements and achieved status

Item	Contents	S	Development Status
FR-01	Linux user space version software	C	Implemented partly
FR-02	RTAI/RTnet version software	N	No
FR-03	Network performance analysis	N	No
FR-04	Implement of the AFDX network software framework, the application services: deliver and reception	C	Implemented partly
FR-05	The network service according to the ARINC 664 part 7	P	Implemented partly
FR-06	The port services	C	Implemented partly
FR-07	Sending service	C	Implemented partly
FR-08	Reception service	C	Implemented partly
FR-09	Service for record of the last	C	The function is implemented

	sending messages		
FR-10	Service for record of the last receiving messages	C	Implemented partly
FR-11	Virtual link function	C	implemented partly
FR-12	Transmission windows for VL	C	implemented partly
FR-13	Multicasting delivery mechanism for the VL	C	No
FR-14	Internal API and user API	C	Implemented
FR-15	Log service	C	Implemented
FR-16	Configuration	C	Implemented
FR-17	Linux user space framework	C	Implemented partly
FR-18	Functions for allocate memory dynamically	C	Implemented partly

3.3 Resolution for existing problems

There are many problems and errors in the previous student projects. Table3-3 lists the problems and errors in the previous projects, together with the finial resolutions in this project.

Table 3-3 Framework problems and solutions

Item	**Errors and problems in previous projects**	**The final resolutions in this project**
Linux user space version simulation software		
Queue management	Each queue has an END_QUEUE flag which represents the end of the queue. When the corresponding buffer pointer meets this flag, the pointer will be moved to the head of buffer. The error is that this flag is not special byte, so it is often confused with the actual data byte.	The solution is that we need to define a special byte to represent the end of the queue. However, the data byte could be anyone and it happens in any places in the buffer. As the resolution, the flag has been defined in the buffer size area with a byte which value is out of the buffer size limitation. The detail information described in the 3.6.2.
Log file	All running information are recorded in a log file, the problems is that log file has its size limits which can not record long enough. So induce to lose the actual messages. Another problem is that I/O read and write speed is too slow, result in poor impacts of the real-time timer quality.	Decrease the number of Sampling point. That could solve the size limit problems. To record the message in the buffer area, when during the running period, in stead of record the message in I/O file. When the simulation finished, the message will be recorded into the I/O file automatically.
Configuration	Configuration file can not load into framework.	Implemented.
RTAI mode	The kernel-call is different between the RTAI and Linux, based on this different kernel-call. Compatibility of the two version software arises.	That is one of an important difference between two versions of framework, which has been showed in Table2-1. The framework uses the correct callings in corresponding version.
RTnet socket functions in messages transmitting	The previous project did not finish this function.	This problem has been achieved successfully. When software call the RTnet recvfrom function, it is easy to cause the errors, even system halted.So that is why the framework

		need a good architecture in Rxer thread in RTAI hard real-time version. The detail information is in the section 3.6.2.
Sequencer number calculation logic error	The sequencer thread fragments the messages into the correct size, according to VL parameters, and save these fragmented data into correct buffers. However, this fragment method is wrong and after this, the fragmented messages have not been saved in the buffers in the correct position.	Change the wrong logic into a correct one. The flowchart has been showed in Figure3-11b.
Rxer thread receiving function	Something wrongs in the rxer thread receive period, such as received buffer size, data malposition, etc	Change the wrong logic into a correct one. The flowchart has been showed in Figure3-14.
Txer thread sending function	Something wrongs in the txer thread sending period, such as buffer size, loss data, etc	Change the wrong logic into a correct one. The flowchart has been showed in the Figure3-9b.
Rrs thread round robin algorithm	Some logic errors in the rrs thread round robins algorithm.	Change the wrong logic errors into the correct one. The flowchart has been showed in Figure3-12.
Vl thread socket functions	Some logic errors in the vl thread network socket functions.	Change the wrong logic errors into the correct one. The flowchart has been showed in Figure3-13.
Dispatcher thread dispatch functions	Some logic errors in the dispatcher thread dispatching functions.	Change the wrong logic errors into the correct one. The flowchart has been showed in Figure3-16.
assemble thread assemble functions	Some logic errors in the assemble thread assemble functions.	Change the wrong logic errors into the correct one. The flowchart has been showed in Figure3-17.
Framework architecture	The framework architecture is too complex to assure the software to keep a good real-time performance,	This architecture still uses in this project now, because changing architecture means change

	especially running on the lower performance computer. The main appearance of this problem is that the more and more messages will be lost in the queue in transitional threads, due to queue overflow.	everything in this software.
RTAI/RTnet hard real-time version simulation software		
Software	No	Implemented
AFDX network performance analysis		
Analysis ways	No	Implemented

3.4 Laboratory circumstance

Figure 3-1 shows the Net-AFS (The Switched-Ethernet Avionics Functions Simulation and Evaluation System) laboratory circumstance. It locates in the Dr. Huamin Jia's office in Cranfield University. It has three switches and eight computers with the Linux operation system. All of this project works have been done on it.

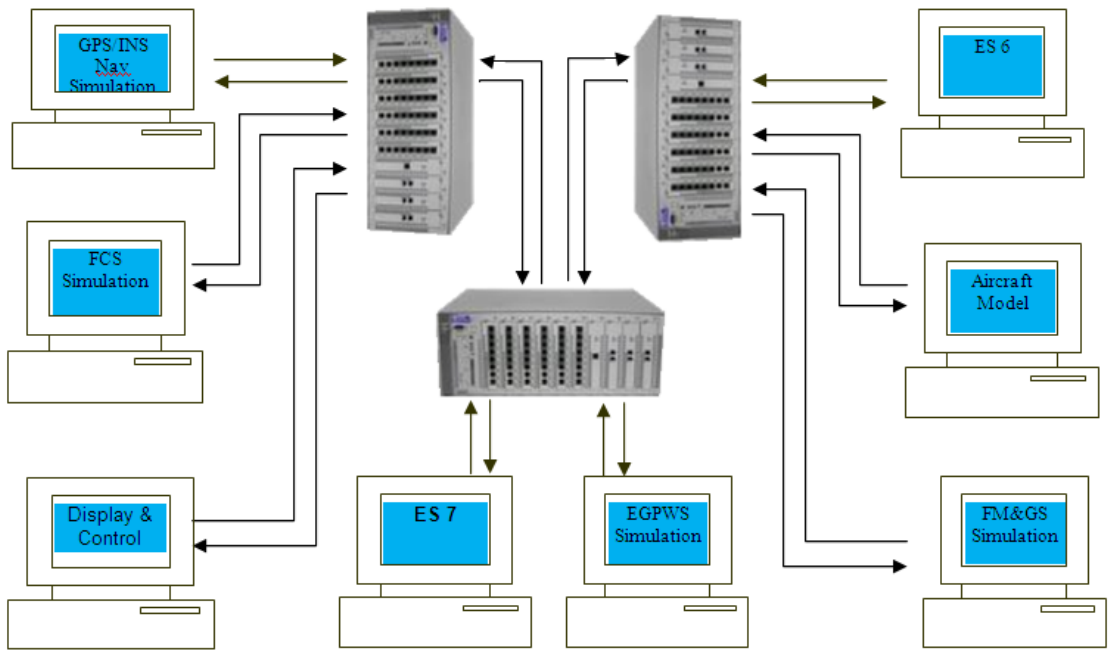

Figure 3-1 Net-AFS Laboratory circumstance

IP Address management

The IP Address distribution to avionics computers is an important issue in the AFDX simulation process. This project applies an effective way to manage the IP Address distribution.

The aircraft computing network mainly have four domains, including the aircraft control, the air company information service, the passenger information service and entertainment service and the passenger equipment domain. Each one has its own IP address limitation, which is showed in Figure3-2. The aircraft control domain consist two main functions, the flight and the passenger cabin core function. Flight and embedded control system's IP addresses belongs to the A class IP Address, and beside these, others belong to B class IP Address.

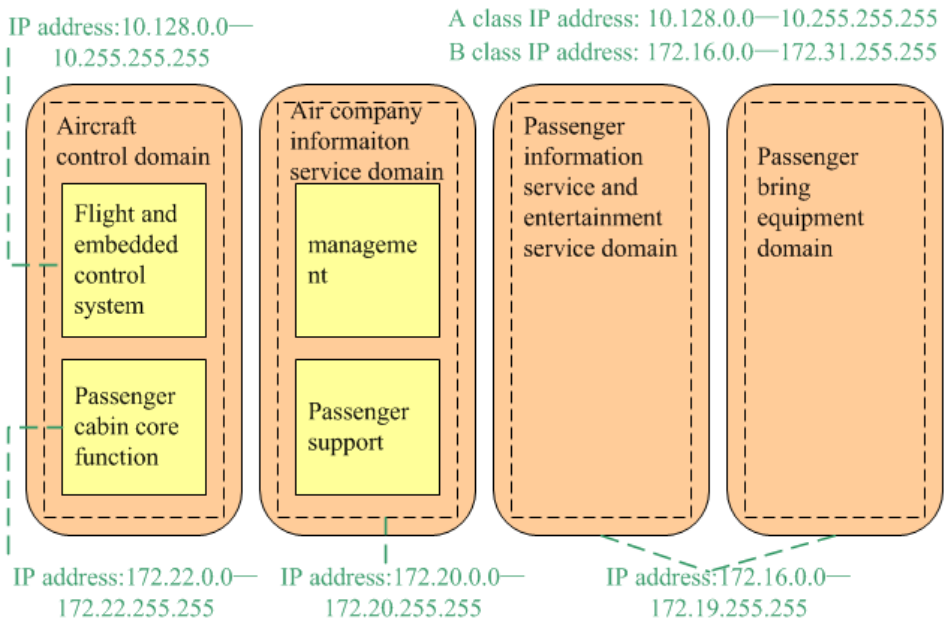

Figure 3-2 Aircraft computing network domains and IP address distribution

Figure3-3 indicates the AFDX IP addressing format in the ARINC 664 part 7. The first bit 0 is the class A identification sign, and followed by the private IP addressing in class A, 16 bits user defined ID and partition ID. The last two parts are defined in the detail design.

IP Unicast Addressing Format(source or unicast destination) 32-bits				
Class A 1 bit	Private IP address 7-bits	User Defined_ID 16-bits	Partition ID	
0	000 1010	“nnnn nnnn nnnn nnnn”	Spare field 3 bits	5-bits

Figure 3-3 IP Addressing Format

As we can see in Figure3-4, the network number 00001010 is equal to the A class address and private ID. According to ARINC 664 part 7, the IP address structure is divided into four parts (W/R/S/A), which is show in Figure3-5.

Net num	designer define
0000 1010	DWWW RRRR RSSS SSSS SSSS AAAA

Figure 3-4 IP address define format

The W part represents the AFDX sub-network number, identifying the position of avionics computers. For example, if avionics architecture is symmetry, the number 4 is used for representing the left part and number 5 is used for the right part.

The R level represents the region identification number, identifying the different regions.

The S Id in green colour represents the different cabinets and the S Id in black colour represents different IMAs.

The A level is the application level, identifying certain applications executing on a subsystem IMA (such as data loading).

An example is showed in Figure3-5.

Along this approach, the aircraft network can be divided into maximum six sub networks (0~6), maximum 30 regions (0~ 30) in each sub network, maximum 2046 PCs (0~2046) in each regions and maximum 13 applications (0~13) in each PC.

Here is an example in Figure 3-5. In this example, the “green circle” represents an application in the End system FMS1. Its IP address equals to 10.193.192.51. It is an aircraft network, so D=1. The sub network identification number W equals to 4, which means it is in the left area. The subsystem identification number(S) equal to 4. The application identification number(A) equal to 3. So the IP address equal to {0000 1010 1100 0001 1100 0000 0011 0011}, that is 10.193.192.51.

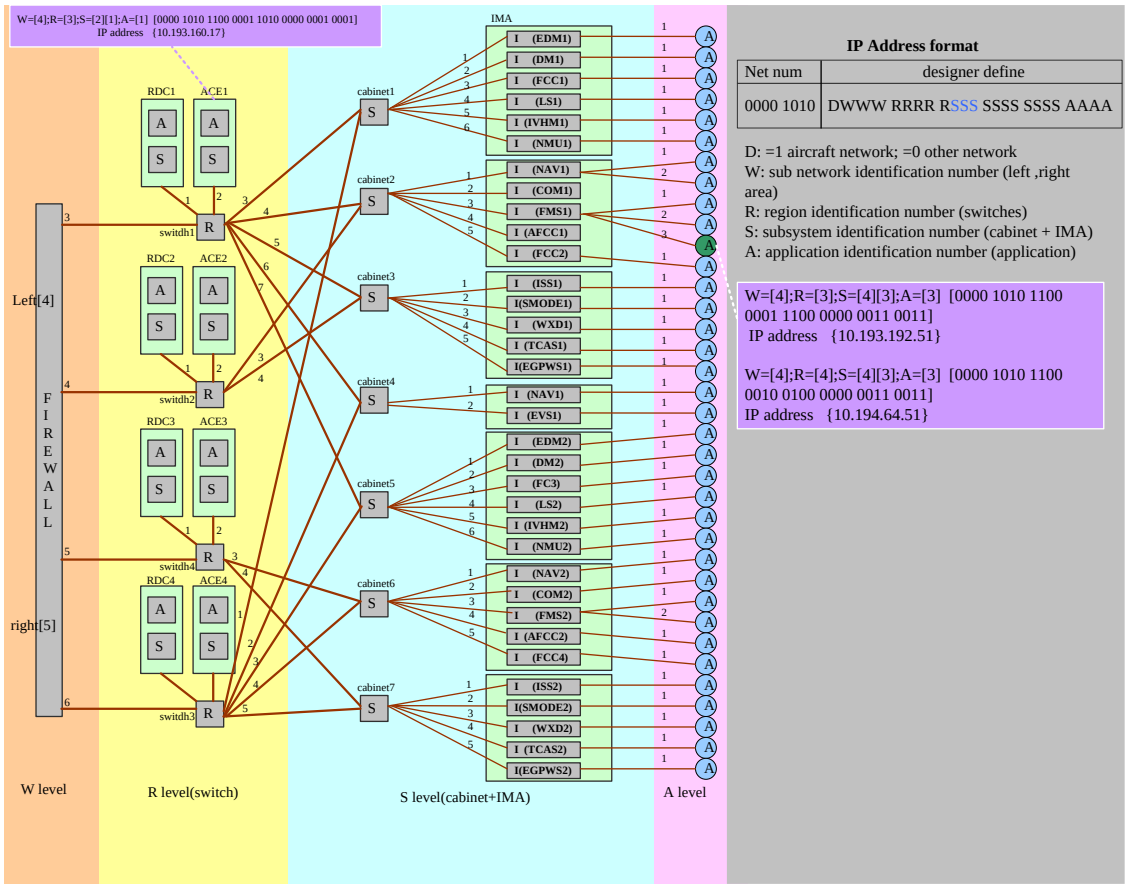

Figure 3-5 IP Address Format

3.5 Top-level architecture

The AFDX simulation platform development is based on the normal technology of network, which is adopted in the common consumer market, without adopting any custom especial hardware. This approach aims to cut down the cost of

building a simulation circumstance without compromise the lossless of the transmission performance.

The simulation framework architecture is the same with the previous projects in [9] and [10]. This project did many modifications in previous project software aim to correct AFDX simulation behaviours in Linux user space version software, because there are so many logical errors in the previous project software.

This project also built a new RTAI/RTnet hard real-time simulation software framework. Some works have been done in this project to transform the Linux user space software into the RTAI/RTnet real-time one with the same architecture.

The original architecture could be maintained, because its detail design already considers the possible change into the hard real-time RTAI application. The design aims for low changes in the source code.

The major differences between these two version software - Linux and RTAI/RTnet versions, are different system kernel calling functions, because running the RTAI/RTnet and Linux have their own system callings for the thread and process management, message queue, mailbox, buffer allocation and the socket API.

3.5.1 Static analysis

The AFDX simulation framework has several software entities, which includes:

- Tx process(*af*): it is developed for testing AFDX communication function purpose – sending messages to the AFDX framework process.
- Rx process(r*af*): it is developed for testing AFDX communication function purpose – receiving messages to the AFDX framework process.
- Façade: it is the static transmission function unit and can be incorporated into the Avionics Tx and Rx process software.
- AFDX process(*afdx*): This is the main simulation framework process.

- Logger module: it records the log file for the whole software entities in the platform.
- Configuration module: it is developed for configuration the AFDX communication simulation.

In order to clearly elaborate the framework entities, here gives a specific network topology example in Figure3-6.

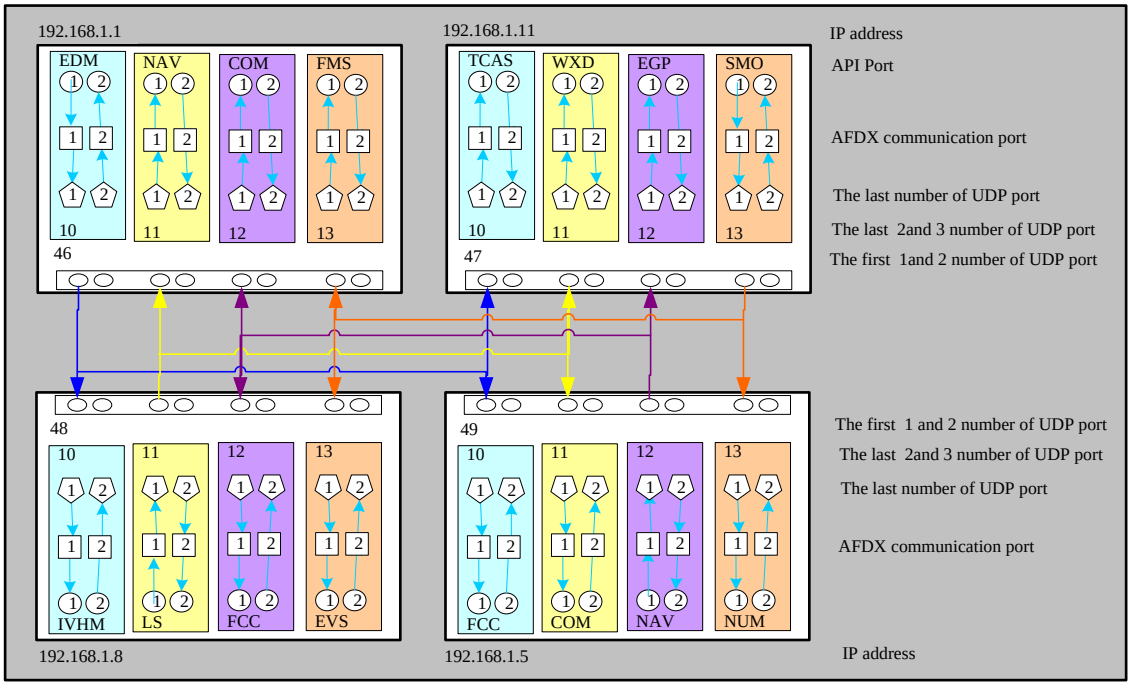

Figure 3-6 Example of avionics architecture topology

Figure3-6 illustrates an example of the avionics architecture topology. There are four IMA End system simulation computers, which IP address ranges from 192.168.1.1 to 192.168.1.11. Each computer also has four avionics subsystem applications with two UDP ports. The pentagon in the figure represents the UDP port, and the number in the pentagon is the last number of UDP port. For example, the top-left End system computer with IP address 192.168.1.1 has four simulation applications, including EDM,NAV,COM and FMS. In the EDM application, two UDP ports have been defined in 49101 and 49102. This figure also shows four VLs. Each VL has one source port and three destination ports. For example, the blue line VL transmits from one source UDP port 46101 in the

computer with IP address 192.168.1.1, to three destination UDP ports 48101 in 192.168.1.8, 49101 in 192.168.1.5 and 47101 in 192.168.1.11.

In order to analyse clearly and distinctly, the introduction for AFDX framework static and dynamic entities will be based on this concrete example. Figure3-7 shows the static entities existing in the end system computer with the IP address 192.168.1.1, and one source UDP port 46101 and three destination ports 46111, 46121 and 46131.

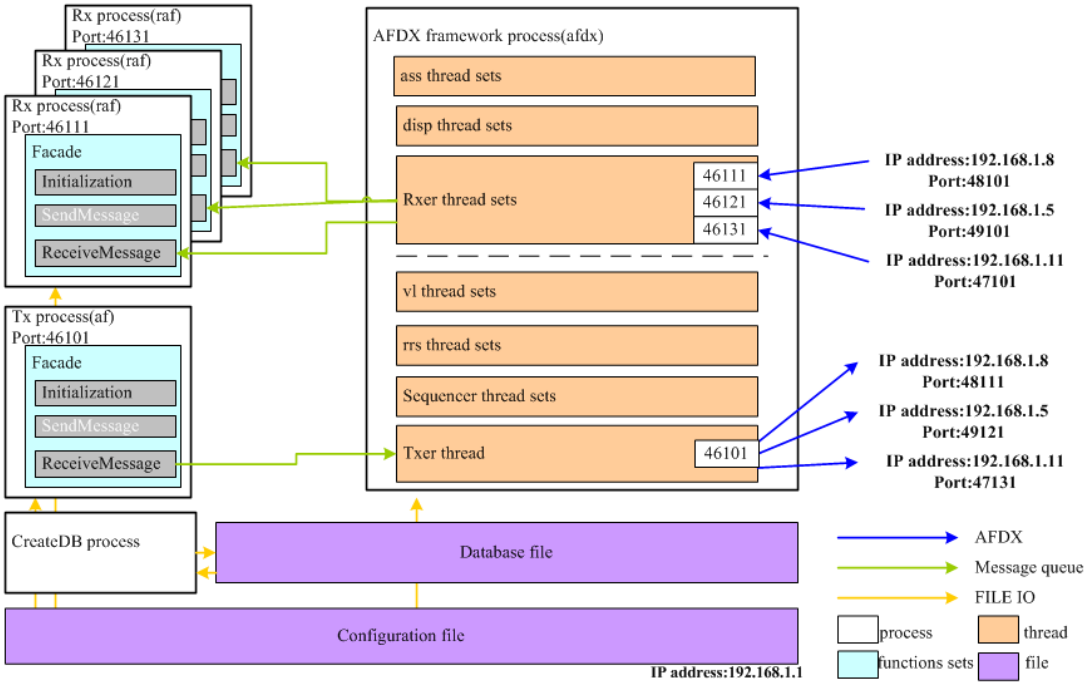

Figure 3-7 Framework static entities

In Figure 3-7, the *CreateDB* process creates the *Database* file in the preparing phase, and all the process and modules could load the records from a *database* file and *configuration* file at the beginning of the simulation. In this example, it has one *Tx* process and three *Rx* processes. The *Tx* processes simulate the avionics subsystem sending process, and it can send the *mailbox* or *queue message* into the AFDX process. The framework provides the façade for *Tx* and *Rx* process in order to implement the transmission functions with an *AFDX* process. The *AFDX* process is the main simulation process. It has two main parts- the sending parts and the receiving parts, and each part has so many sub threads. The sending part can send the AFDX socket message to the switches

network according to the AFDX behaviours, and the receiving part can receive the message from the network according to the AFDX behaviours. In this example, the *Txer* thread sends a message from the UDP port 46101 to the three destination IP address 192.168.1.5 ,192.168.1.8 and 192.168.1.11, and the *Rxer* threads receive the message from the three source IP address 192.168.1.5,192.168.1.8 and 192.168.1.11 into the three UDP ports 46121,46131,46111 respectively.

The AFDX simulation framework has several software static entities:

Tx process(af)

Although the process named **af** is not a main part of simulation framework, it still acts as an important role in simulation. It is developed for testing AFDX communication function purpose – sending messages to the AFDX framework process, in the form of the Message Queue(in Linux standard user space) or Mail Box(in RTAI), through calling functions located in the function set of the AFDX façade.

In this example, the end-system computer has one Tx process with the Message Queue port number equal to 46101.

Façade

The façade module can not be directly compiled into an executable application, and it is a static function unit, which is incorporated into Avionics Tx and Rx process software. The main function is to successfully support the messages transmission between an avionics Tx or Rx process with AFDX process, using Message Queue(in Linux) or Mail Box(in RTAI).

The façade includes the initial function, sending and receiving message functions, including other supporting basic functions, such as the queue management. Configuration data for all simulation process are loaded using the configuration module in the façade, and the database module read information records in the same database file.

Rx process(raf)

Rx process has the same architecture and characteristics with the Tx process. The difference between them is that a Rx process is developed for testing reception of message functions. This process is named by **raf**.

In this example (showed in Figure3-7), the end-system computer has three Rx processes with the Message Queue port numbers 46111, 46121 and 46131.

AFDX process(afdx)

This is the main simulation framework process. It runs two major threads – Txer thread and Rxer thread.

Txer thread

Txer thread is the main thread in the AFDX process, including two major modules: the Initialization module and receiving module.

Initialization module main functions are listed here:

- To load configuration parameters from a configuration file, using the configuration module.
- To read records from the network topology database, using the database module.
- To initialize the Message queue, Mail box, Semaphores and Mutex.
- To allocate the buffer for environment parameters.
- To initialize a RTAI handler in the RTAI circumstance.
- To run and manage the sub threads.

The sub threads include:

- Sequencer thread: It fragments input data according to Lmax. It adds sequencer number into the right fragmented units. It queues units into a correct sub virtual link buffer.

- RRS(Round robin scheduler) thread: it queues the data units into the right virtual link buffers according to Round Robin algorithm.

- VLS(Virtual links) thread: it transmits VL packets according to the correct network behaviour ,such as BAG and jitter.

Rxer thread

Rxer thread provides AFDX socket message reception functionality. It is waked by a Txer thread. This thread communicates with the other computers through socket API, and transmits the AFDX messages which have been received from the other computers to the right buffers, then supports the transmission to the corresponding Rx process message ports. The connection with Rx processes could be done by its sub thread – ASS thread.

The Rxer thread is another main thread in AFDX process, including two major phases: Initialization phase and receiving phase.

The initialization phase includes:

- To load the configuration parameters from the configuration file, using the configuration module.

- To read the records in the network topology Database, using the database module.

- To initialize the Message queue, Mail box, Semaphores and Mutexs .

- To allocate buffers for environment parameters.

- To initialize the RTAI handle in the RTAI circumstance.

- To run and manage the sub threads.

The sub threads include:

- Dispatcher thread (DIS): it analyses the coming network frames from Rxer threads and sends them to the right assembler thread queue buffers.

- Assembler thread (ASS): it assembles the input frames to construct the whole message. After this, the thread queues them in the correct Mailbox or Message queue port.

In the receiving phase it receives other computer's socket messages.

Logger module

The logger module is function units, which records a *log* file for the whole software entities in the platform. All the simulation running information, such as warning, successful, error information, could be recorded in a *log* file in each particular end-system computer.

Configuration file

The framework allows all process modules to set and read configuration parameters form the same configuration file. The data in this file are the software internal configuration and the network configuration.

The configuration file parameters are listed in Listing3-1.

```
// min number port.
unsigned int i_min_port_number ;
// max port number.
unsigned int i_max_port_number ;
// default value port number.
unsigned int i_default_port_number ;
// max amount of bytes in port.
unsigned int i_max_messages_per_port ;
max number of dataset  in units.
unsigned int i_max_datasets_in_a_message ;
// max bytes size in a dataset.
```

```
unsigned int i_max_data_size ;
// the reception buffer.
unsigned int i_reception_buffer_size ;
// max  txer buffer.
#define d_max_txer_buffer_size 2000

// the server id number.
unsigned int server_id ;
// the client id number.
unsigned int client_id ;
#define AFDX_FACADE_PRIO 5
#define AFDX_TXER_PRIO 4
#define AFDX_SEQUENCER_PRIO 3
#define AFDX_RRS_PRIO 2
#define AFDX_VLS_PRIO 0
// max number of sub vl.
#define d_MAX_NUMBER_OF_SUB_VL 4
// min number of sub vl.
#define d_MIN_BAG 1
// max BAG.
#define d_MAX_BAG 128
// default BAG.
unsigned int u_default_bag ;
// default Lmax
unsigned int u_ default_l_max ;
```

```
// max number of sub vl.
unsigned int i_sub_vl_size ;
// max number of a vl buffer size.
unsigned int u_vl_size ;
// log file name.
extern char c_log_file_name[LOG_FILE_NAME_SIZE];
// max number of message queue.
#define d_MQ_NAME_SIZE 30
// the mq name in Txer.
char c_txer_mq[MQ_NAME_SIZE];
// the number of Sequencer  threads.
unsigned int i_Sequencer_threads;
// the number of dispatcher threads.
unsigned int i_dispatcher_threads;
// max number of the dispatcher thread buffer size.
unsigned int i_dispatcher_buffer_size;
//load configuration file function .
bool loadConfiguration(char *fileName);
```

Listing 3-1 A quick overview of the configuration file parameters

Database file

The name of data file includes information of an end-system ID number and Database version number. It saves the records of the IP address, VL parameters for every AFDX ports.

Database management module (createDB)

This module creates a Database file in the prepare phase of simulation, at the same time, writes the records in a database file.

A quick overview of the record structure in a database file is included in Listing3-2.

```
//brief This structure keeps the data about the virtual link of the General
struct g_virtualLink{
        unsigned int bandwidthAllocationGap;
        unsigned int lMax;
};
//brief This structure keeps the data about the source ES of the database.
struct g_source{
        unsigned int subVirtualLink;
        keyvl virtualLink;
        unsigned int bufferSize;
};
//brief This structure keeps the data about the destination ES of the database.
struct g_dest{
        bool isSampling;
        unsigned int bufferSize;
        keysource key_source;
};
//brief This structure keeps the data about the source ES of a particular txer ES
//database.
struct txer_source{
        unsigned int subVirtualLink;
        keyvl virtualLink;
        unsigned int bufferSize;
```

```
};
//brief This structure keeps the data about the destination ES of a particular txer
//ES database.
struct txer_dest{
        keysource key_source;
};
//brief the General database and the particular txer ES database are the same.
typedef struct g_virtualLink txer_vl;
//brief This structure keeps the data about the destination ES of a particular rxer
//ES database.
struct rxer_dest{
        bool isSampling;
        unsigned int bufferSize;
        in_addr_t source_ip;
        in_port_t source_port;
        keyrxerdest key_next;
};
```

Listing 3-2 A quick overview of the records structure in database file

3.5.2 Dynamic analysis

The framework could be compiled into many threads and processes before executing them. These running entities functions have been specified in the front section.

The simulation process includes two main phases:

Prepare phase

Before the simulation run time phase, some prepare works need to be done at first, such as configuration, database management and environment settings. There are three main steps in this phase.

- Operation system environment settings: this framework has two versions, one is running on the Linux user space system calling the Linux socket API to achieve communication functions, and the other is on the RTAI using the RTnet socket API. Due to these targets, different versions require different operation system environments. Before running the programs, an important mission must be done to set the environment, especially when running on the RTAI/RTnet system. The detail information has been elaborated in section 2.2.

- Configuration parameters settings: setting parameters, listed in Listing3-1, in configuration files, which will be used in the run time period.

- Create database file: using the database management process module to create a database file, this module is a set of software with user-friendly interface. It is easy to create, read and write database records into a file. The records parameters can be set in the main5.c file(showed in List3-2). After debugging and compiling the program, the *createDB* process appears. When this process executes, the database file creates. At the same time, the records we settled in the main5.c will be listed in a print screen, checking all the parameters on the screen to make sure that it is corresponded with the avionics architecture topology.

Run time phase

The framework has one main process - AFDX process, includes with two main threads Rxer and Txer. The Txer thread can be considered the master thread in the whole process.

In the framework, the parallel threads execute at the same time, aiming to maintain the framework real-time performance. On the sending part, RRS, Sequencer and VLs threads are launched and depend on a Txer thread. On

the receiving part, Assembler and Dispatcher threads are launched and depend on a Rxer thread. The framework dynamic modules and their name and source files are showed in Table3-4.

Table 3-4 Summary of processes and threads

Name	Process/Thread	Source file	Module
rxer	T	afdx.c, afdx.h,rxer.c, rxer.h, dbm.c, loadconf.c,util.c	Rxer
txer	P	afdx.c, afdx.h,txer.h txer.h, dbm.c, loadconf.c,util.c	Txer
afdx	P	main6.c,txer.c,loadconf.c,dbm.c message.c,util.c,rxer.c,sequencer.c,vls.c,dispatcher.c,assembler.c,petition.h,environment.h	
sequencer	T	txer.c, txer.h, sequencer.h, sequencer.c,afdx.h, dbm.c, loadconf.c,util.c	Txer
rrs	T	txer.c, txer.h, rrs.h,rrs.c, afdx.h, dbm.c, loadconf.c,util.c	Txer
vls	T	txer.c, txer.h, vls.h, vls.c ,afdx.h dbm.c, loadconf.c,util.c	Txer
dis	T	rxer.c, rxer.h, dispatch.c, dbm.c, loadconf.c,util.c, dispatch.h, afdx.h	Rxer
ass	T	rxer.c, rxer.h, assembler.c, assembler.h, afdx.h	Rxer
rxer	T	rxer.c, rxer.h, dbm.c, loadconf.c,util.c,afdx.c,afdx.h	Rxer
af	P	af.h,af.c,afdx.o,loadconf.c,dbm.c,ut	Af

		il.c	
raf	P	raf.h,raf.c,afdx.o,loadconf.c,dbm.c, util.c	Raf
createDB	P	main5.c,dbm.h,dbm.c	createDB

Here is an example, which illustrates the data transmitting relationship between these threads in the run time period. In order to discuss clearly and distinctly, this dynamic entities figure uses the same example showed in Figure3-6. It also has one input port and three output ports. The run time behaviours in the computer are illustrated in Figure3-8. The façade module is executed by the Tx or Rx process as static linking library.

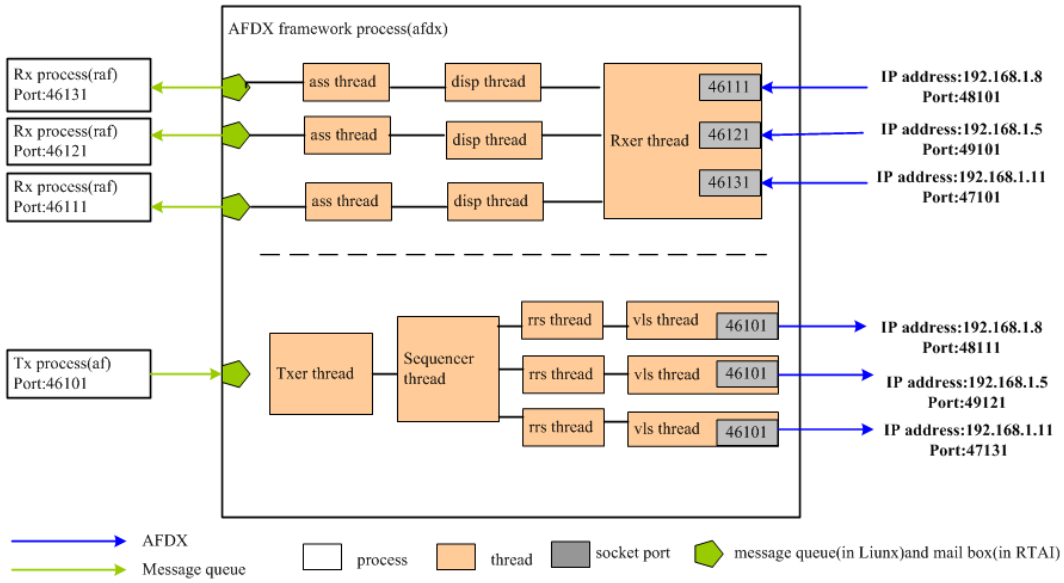

Figure 3-8 Framework run time behaviour

Each thread buffer has Mutex protections due to avoid the data read and write collisions. After finishing their missions, each thread will return to the start point quickly waiting another messages coming into their input buffer, without waiting for the whole packet transmitting event end. So this is a paralleling framework!

3.6 AFDX simulation detail design

3.6.1 Source code structure

The main directory (/afdx) includes all scripts and software files, including configuration and database files. Statistics about the number of the files and modules have been showed in Table3-4.

Table 3-5 shows the detail of the framework software design configuration. It includes programming language, framework version, use of APIs and External supported DLLs, development tools and operation machine information.

Table 3-5 Framework software design

Item	Contents	Details
Programming language	C	
Framework version	Version1.1 Version2.1	Linux task RTAI/RTnet task
Use of APIs and External supported DLLs	kernel-level software	RTAI /RTnet or Linux
	Gdbm	Database management systems
	Dot.conf	Configuration module
Development tools	Gedit on GNU/Linux	Codes editor
	GNU Compiler version4.3.2	Application compiler
	GNU GDB debugger	Application debugger
	Data Display Debugger	A graphical user interface(GUI) for debugging software application
Operation machine	Debian Lenny i386	with GNU/Linux Kernel version2.6 with RTAI and RTnet

3.6.2 Framework analysis

The whole communication process could be simply described in this way, that avionics data have been settled in the IM message structure, which would be sent from a defined particular port in Af process to the AFDX process through the Message queue or Mail box. The Txer thread in AFDX process deals with these messages according to AFDX regulation. As the results, the packets will be sent to other computers through the switches network, continually received by Rxer thread in the AFDX process running on the destination computers. Packets will be dispatched and assembled according to the AFDX regulation. In the destination computers, the receiving messages will be sent to the corresponding *raf* process.

In this section, we list some part of previous projects software flow chart, in order to do some comparison with this project.

AFDX process

The main function launches the AFDX process. The calling is defined by *runTxer()*. It starts an AFDX process as a RTAI task in the initialization phase, before the loop running phase.

Txer process

The Txer process flow chart of the previous projects is showed in Figure 3-9a, and this project Txer process flow chart is showed in the Figure3-9b. This project made a lot of modification in the previous projects because there are so many logical errors in it.

The Txer process has two main phases.

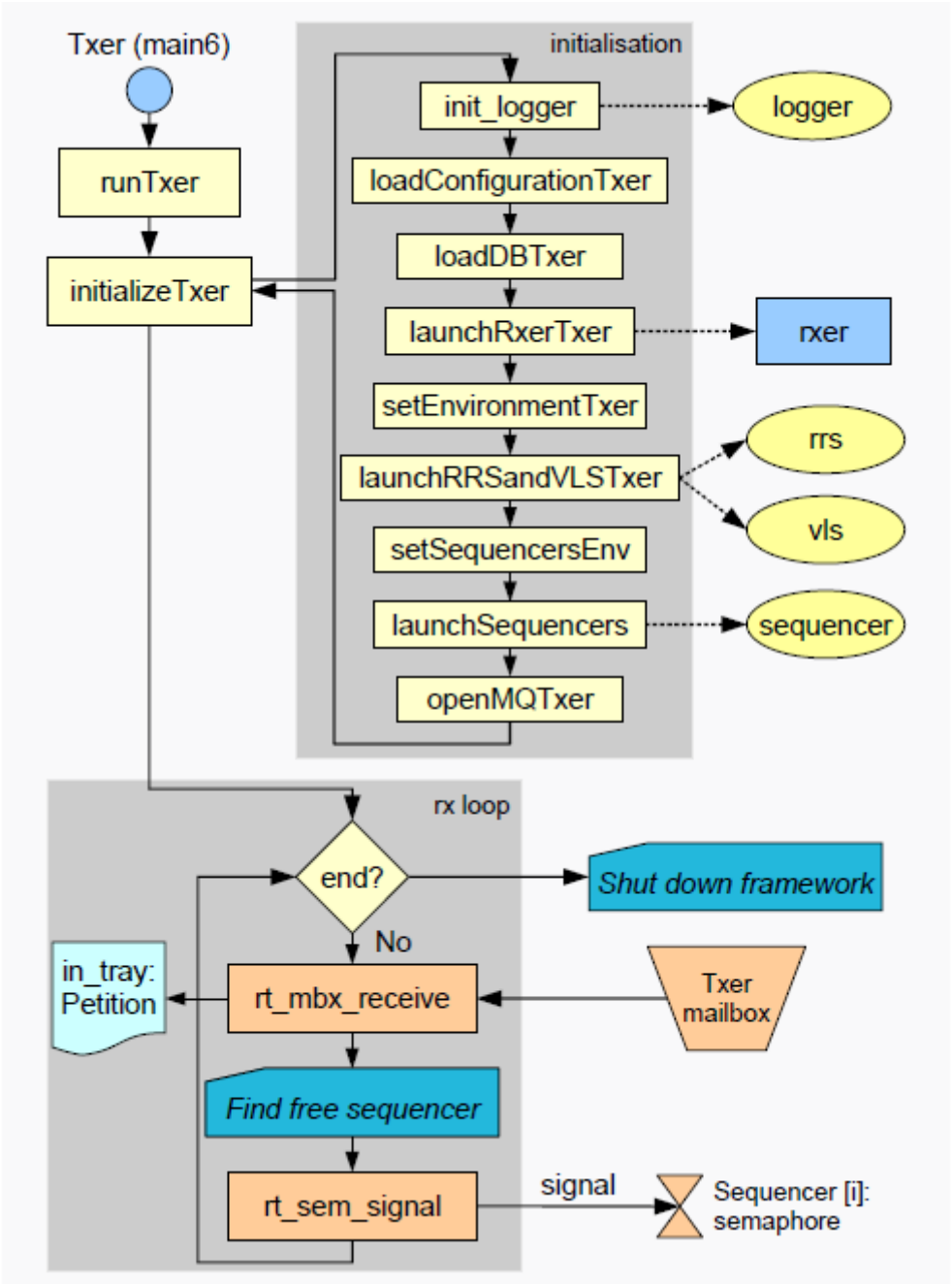

Figure 3-9a Txer process flow chart of the previous project[9]

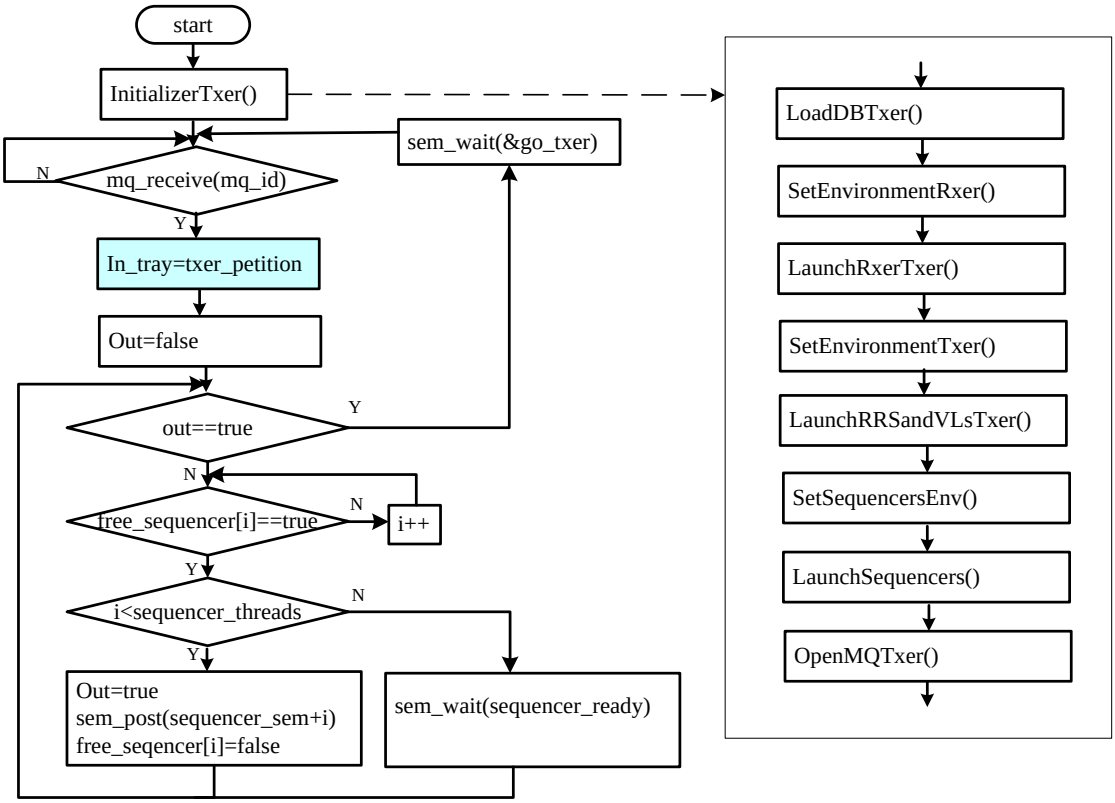

Figure 3-10b Txer process flow chart of this project

1) Initialization

In initialization (*initializerTxer*), the main functions are listed in Table3-6.

Table 3-6 Main functions of *initializerTxer*

Function	Detail
Initialization	Environment parameters, message queue, mutex, sigmapores, mail box, buffer
Load	The framework configuration and network database records
Lunch	RTAI timer; *rxer*, *rrs*, *vls*, *sequencer* thread.

The initialization function implements three main functions. At first, it allocates some buffers for environment parameters and internal data. Then it creates the message queues and threads management objects, such as the Mutex and Semaphore. After this, it loads the framework configuration and network

database records. At last, the initialization function starts a RTAI task and timer and launches the VLs, RRS, Rxer and Sequencer threads. The quantity of virtual links in the network topology effects the number of VLs and RRS threads, and the sequencer threads quantity are set in the configuration file.

2) Transmission

After the initialization phase, the process goes into a transmission running loop, showed in the figure3-9b.

The *rt_mbx_recieve*() call waits for the messages coming from the Mailbox or Message queue in the *Af* progress. When units arrive, it will be saved from *txer_petition* to the internal *in_tray* buffer, which is defined as a *structure* object of *sPetition*. Listing3-3 illustrates the *sPetition* defination.

```
// This is the sPetition structure.
struct sPetition{
        AFDXport a_port;
        size_t s_length;
        unsigned char u_ data[i_max_txer_buffer_size];
};
```

Listing 3-3 The structure of sPetition

The Txer thread searches for a free Sequencer thread. When one of the Sequencer threads is idle, the Txer thread will post a *sequencer_sem* semaphore to wake it up. A *go_txer* semaphore indicates the message in *in_tray* buffer has been read by a working Sequencer thread, and it is ready for another messages saved in it and continues the running loop.

The Txer thread starts several sub threads. The messages are translated between them through already defined buffers allocated in the initial phase. Figurate3-10 shows the data relationship between threads in the light of the blue line VL in Figurate3-6. This is the example which has been used in the static entities introduction. This VL has one input port and three output ports, and messages are sent form a *petition* buffer in the *Af* process to the environment parameter *env[i]->vl_m* in VLs thread and continually transmitted

to the AFDX switches network. The message transmission between these threads will be described in the following section.

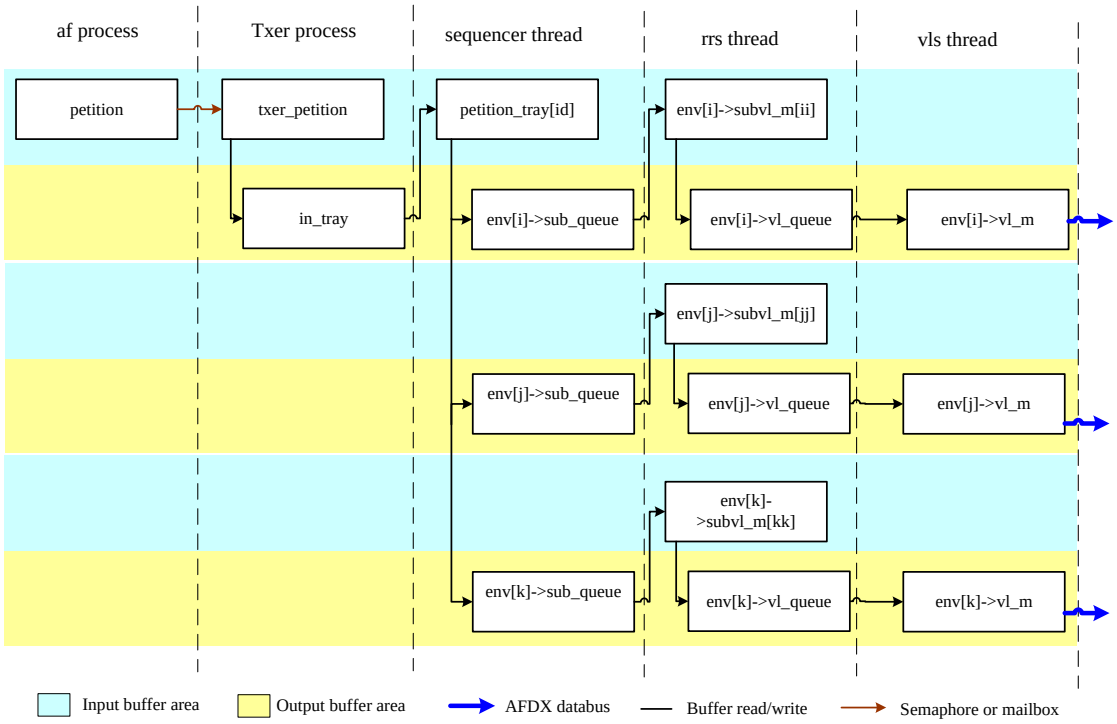

Figure 3-11 Data relationship between Txer sub modules

Sequencer

The *sequencer* thread fragments a coming message using its Lmax, and transmits a fragmented frame to a correct sub VL queue.

The *sequencer* thread flow chart of the previous projects is showed in Figure 3-11a, and this project *sequencer* thread flow chart is showed in Figure 3-11b. This project made a lot of modification in the previous projects because there are so many logical errors in it.

The *sequencer* thread is launched by a Txer thread with the *sequencer_sem* semaphore. The sequencer thread waits for the exclusive right to copy the *intray* buffer into internal *petition_tray*. If the *out_seq* flag equal to *true*, the thread will be terminated.

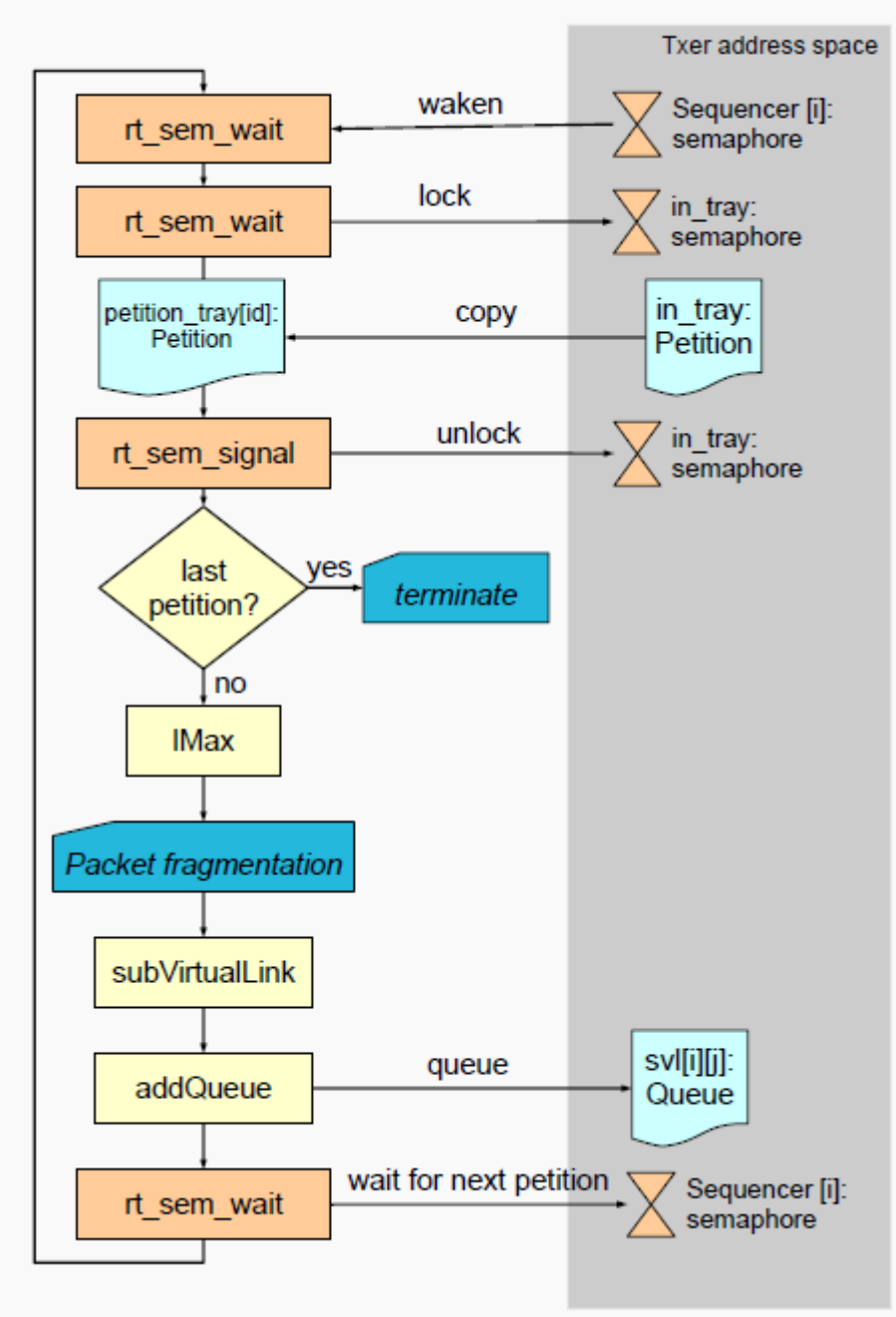

Figure 3-12a Flowchart of sequencer thread of the previous project

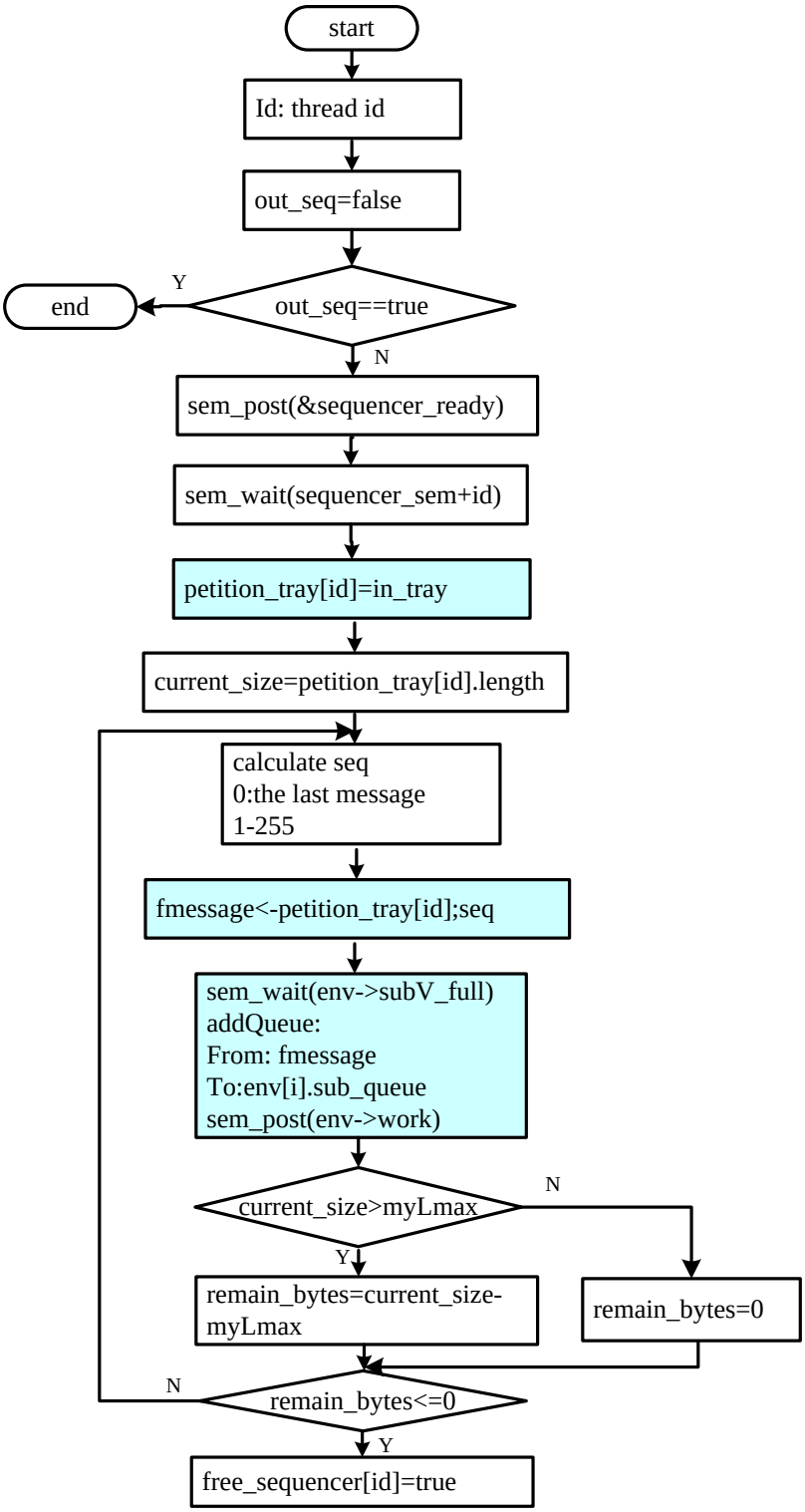

Figure 3-13b Flowchart of sequencer thread of this project

After fragmenting a message according to the Lmax, the thread calculates the *seq* number for each fragmented unit, and queues it into the *fmessage* queue,

tagged with the *seq* value. The function *subVirtualLink*() tries to find a right sub virtual link queue. Then the thread will record the *fmessage* buffer into this sub queue. The semaphore *env[i]->subvl_full* will protect the record process in order to avoid the read and write collisions, and a semaphore *env[i]->vl_work* tries to wake a free RRS thread.

The *myLmax* value is Lmax of this sub virtual link. The fragment loop can be finished when the value of *remain_bytes* is smaller than the *myLmax*. That is the logical for fragmenting message by Lmax.

Round Robin Scheduler (RRS)

Each VL has a corresponding RRS thread. It is waked by a semaphore *env->vl_work* in a Sequencer thread. The main function of this thread is to deliver a sub virtual link fragmented units into a correct virtual link queue, according to the round robin scheduling.

The flowchart of the RRS thread is showed in Figure3-12.

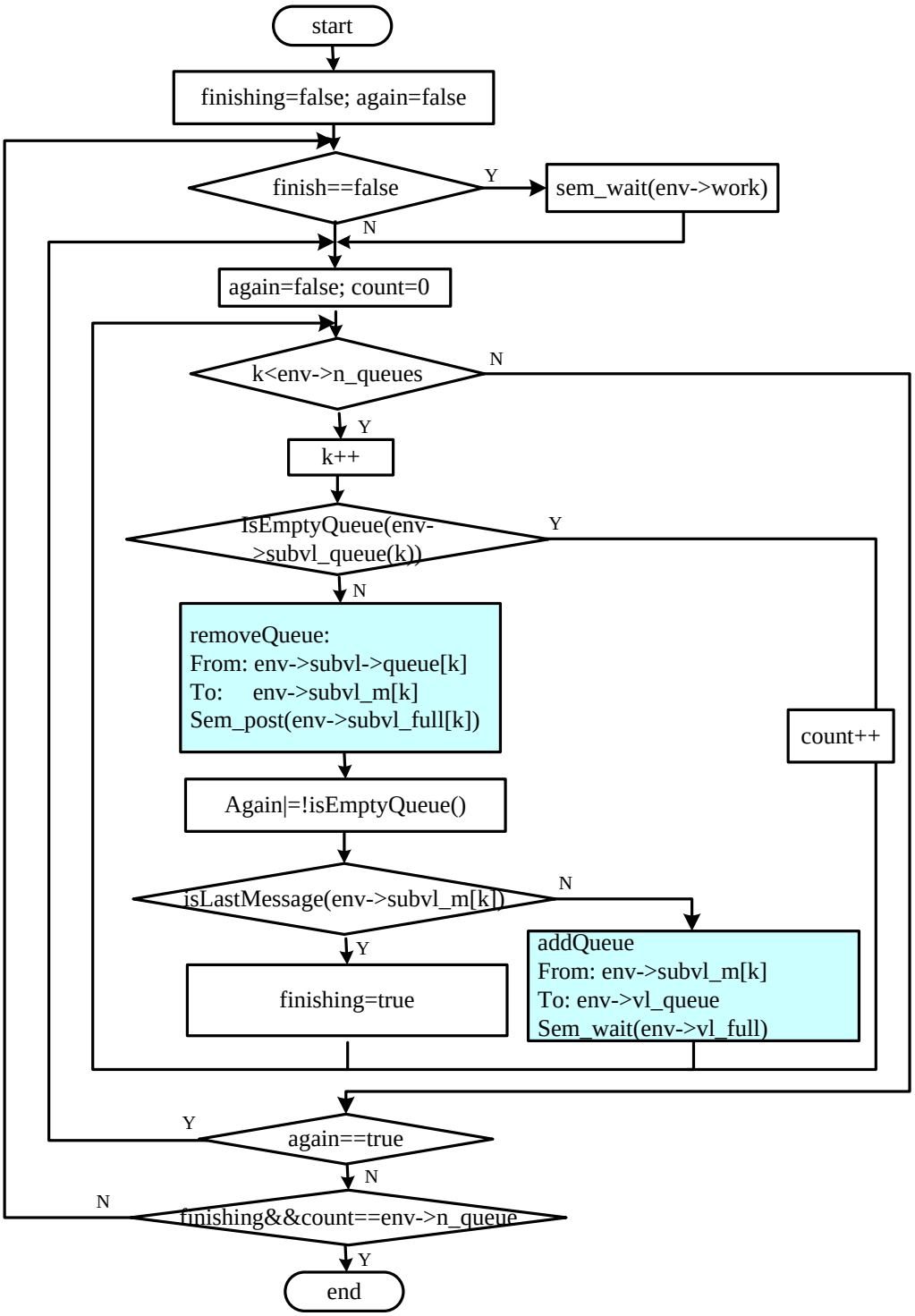

Figure 3-14 Flowchart for *rrs* thread

Virtual Link Scheduler (VLs)

At first, each virtual link associates with a corresponding VLs thread. It transmits the virtual link packets onto the AFDX switches network within the maximum BAG limitation.

At first, in real-time version, the thread registers as a periodic RTAI task by *rt_task_make_periodic*() function. The period time is equal to the BAG of this virtual link. The calling *rt_dev_socket*() initializes the socket handler and binds it to the AFDX UDP port(*rt_dev_bind*()). A message will be removed from the *env[i]->vl->queue* into the *env[i]->vl_m* buffer. Then UDP packets will be sent into the correct ports by calling *rt_dev_sendto*().

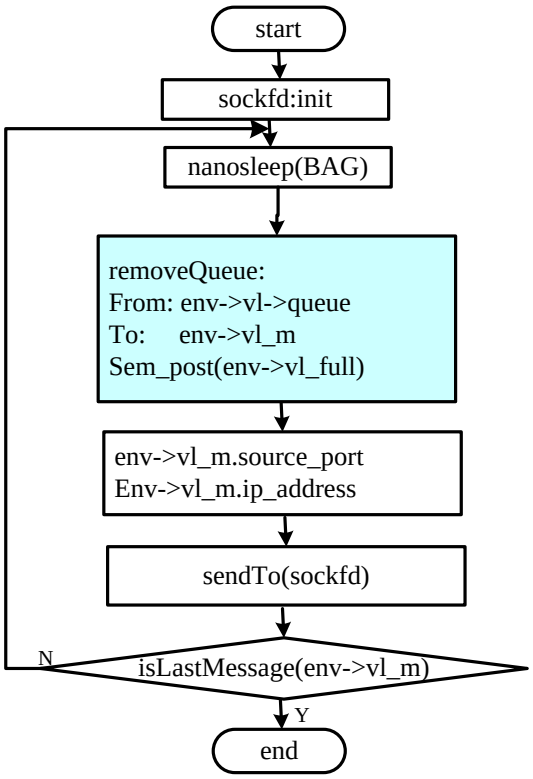

Figure 3-15 Vls threads flowchart

Rxer thread

The Rxer thread flow chart shows in Figure3-14.

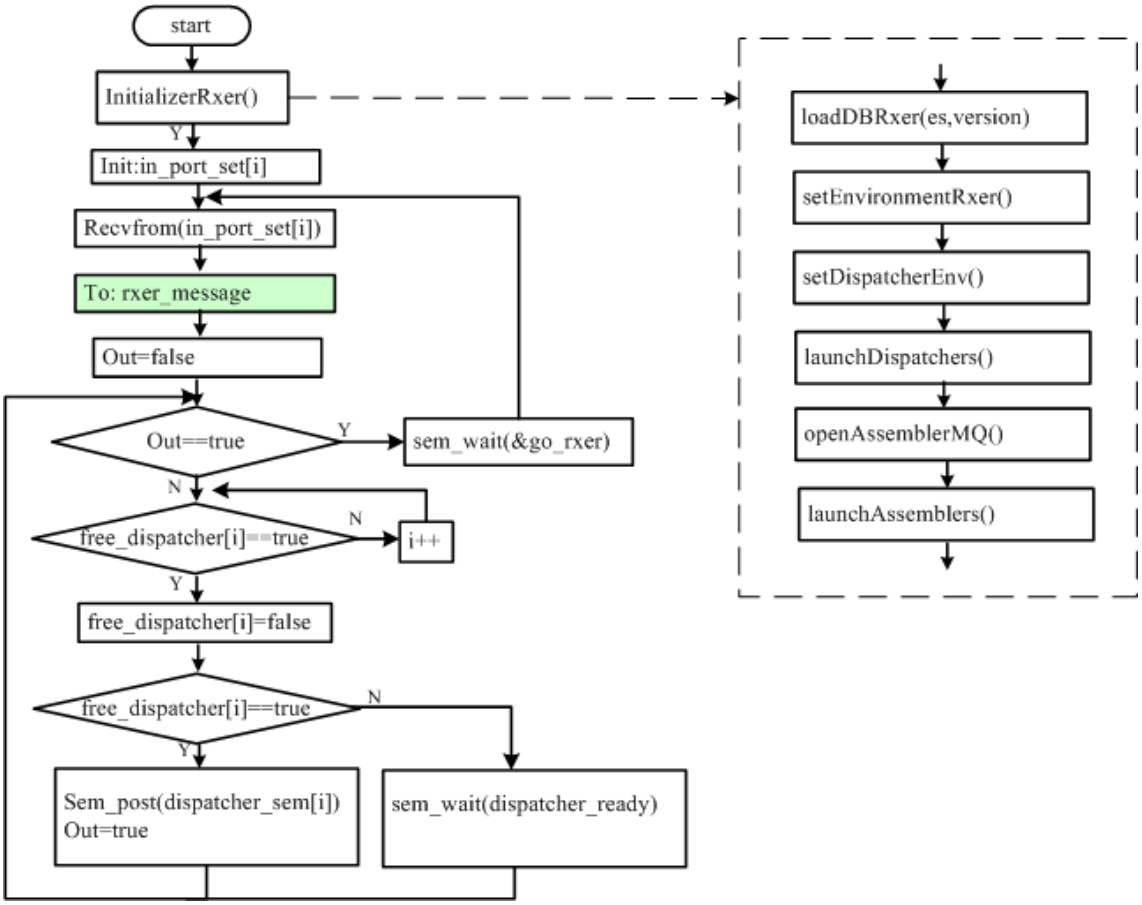

Figure 3-16 Rxer thread flowchart

The Rxer thread is launched by a TXer thread by calling *fork()*. It starts from the *runRxer*() function. This thread has two main phases- initialisation and reception.

1) Initialisation

In initialization (*initializerRxer*), the main functions are listed in Table3-7.

Table 3-7 Main functions of *initializerRxer*

Function	Detail
Initialization	Environment parameters, message queue, mutex, sigmapores, mail box, buffers for its sub threads.
Load	The framework configuration and network database records
Lunch	RTAI timer; *ass* thread; *dis* thread.

The initialization function implements three main functions. At first, it allocates the buffers for environment parameters and the Rxer thread internal data. Then it creates the message queues and threads management objects, such as mutexs and semaphores. After this, it loads the framework configuration and network database records. At last, the initialization function starts the RTAI task and the timer and launches the Dispatcher and Assembler threads. The quantity of VLs in the network topology effects the number of the working Dispatcher and Assembler threads, and the number of the Dispatcher and Assembler threads is defined in a configuration file.

2) Reception

After the initialization phase, the Rxer thread goes into the reception running loop, showed in Figure3-14.

At first, in real-time version, the Rxer thread registers itself as a oneshot RTAI task by *rt_set_oneshot_mode*() function. The calling *rt_dev_socket*() initializes a socket handler and binds it to AFDX UDP ports(*rt_dev_bind*()). All the sockets listen to the incoming packets by calling *rt_dev_select*() using a file data set handler(*fd_set*). The AFDX message will be received into the *rxer_message* buffer by calling *rt_dev_recvfrom*().

The Rxer thread searches for a free Dispatcher thread. When one of the Dispatcher threads is idle, the Txer thread will post a *dispatch_sem* semaphore to wake it up. The *go_rxer* semaphore indicates the message in *rxer_message*

buffer has been read by a working Dispatcher thread, and it is ready for another messages saved in it and continues the reception running loop.

The Rxer thread starts several sub threads. The messages translate between them through the buffers allocated in the initial phase. Figure 3-15 shows the data relationship between threads in the light of the blue line VL in Figurate 3-6. This is the example which has been used in the static entities introduction. The messages are sent form the *rxer_message* buffer to the destination *buffer* in assemble threads and continually transmitted to the Message queue or Mailbox port in the *Raf* process. The message transmission between these threads will be illustrated in the bellow section.

The Rxer thread messages transmission between sub threads is showed in Figure 3-15.

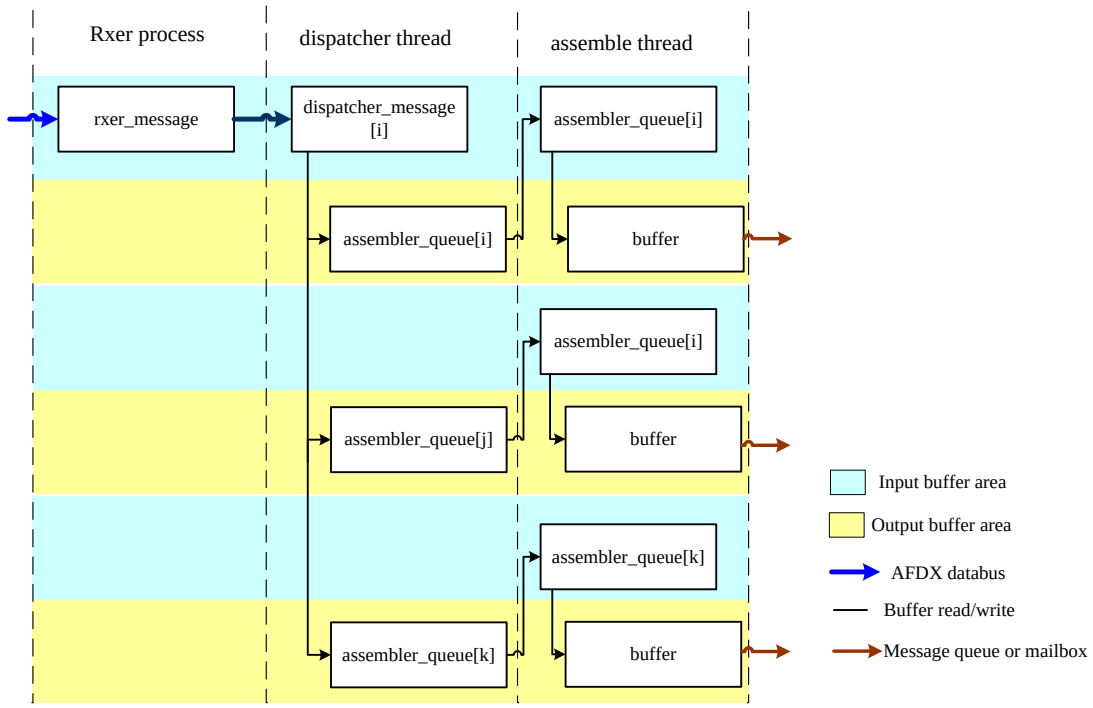

Figure 3-17 Rxer thread messages transmission between sub modules

Dispatcher

Each virtual link has a corresponding Dispatcher thread. It is waked by a semaphore *dispatcher_sem* in the Rxer thread. The dispatcher thread analyses

every message head to find the correct destination port. Then it sends the message to a corresponding assembler thread.

It copies the data from a *rxer_message* buffer to a *dispatcher_message* buffer in order to let the *rxer_message* free for another incoming message. After this, the data in *dispatcher_message* buffer will be queued into the *assembler_queue[i].* The semaphore *assembler_full[i]* will protect the queue in order to avoid the read and write collisions. The *free_dispatcher[id]* represents this *id* dispatcher thread is idle now.

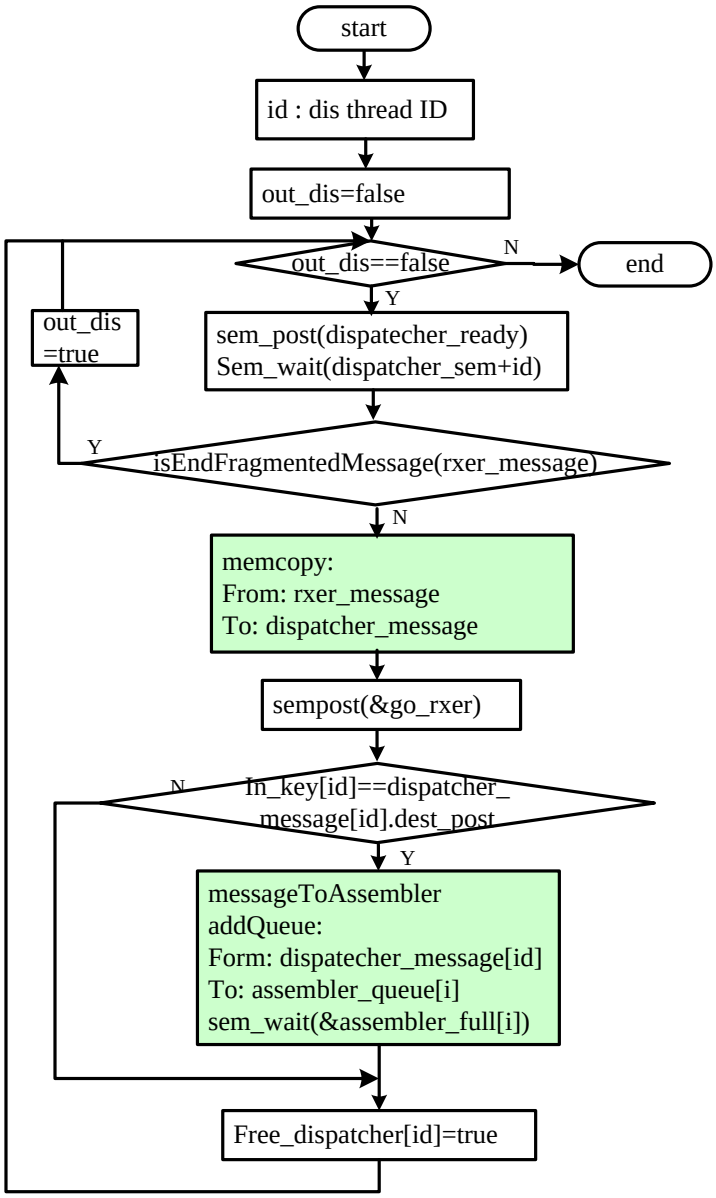

Figure 3-18 Dispatcher thread flowchart

Assembler

The assembler thread builds an original AFDX frame from incoming fragmented messages. It is a reverse process of the sequencer. The results will be sent to correct AFDX ports(*AFDX_id[i]*) associated with af processes, by calling the function *mq_send()*. The semaphore *AFDXport_sem* protects the sending process in order to avoid the read and write collisions.

Figure 3-17 illustrates the assembler thread flowchart.

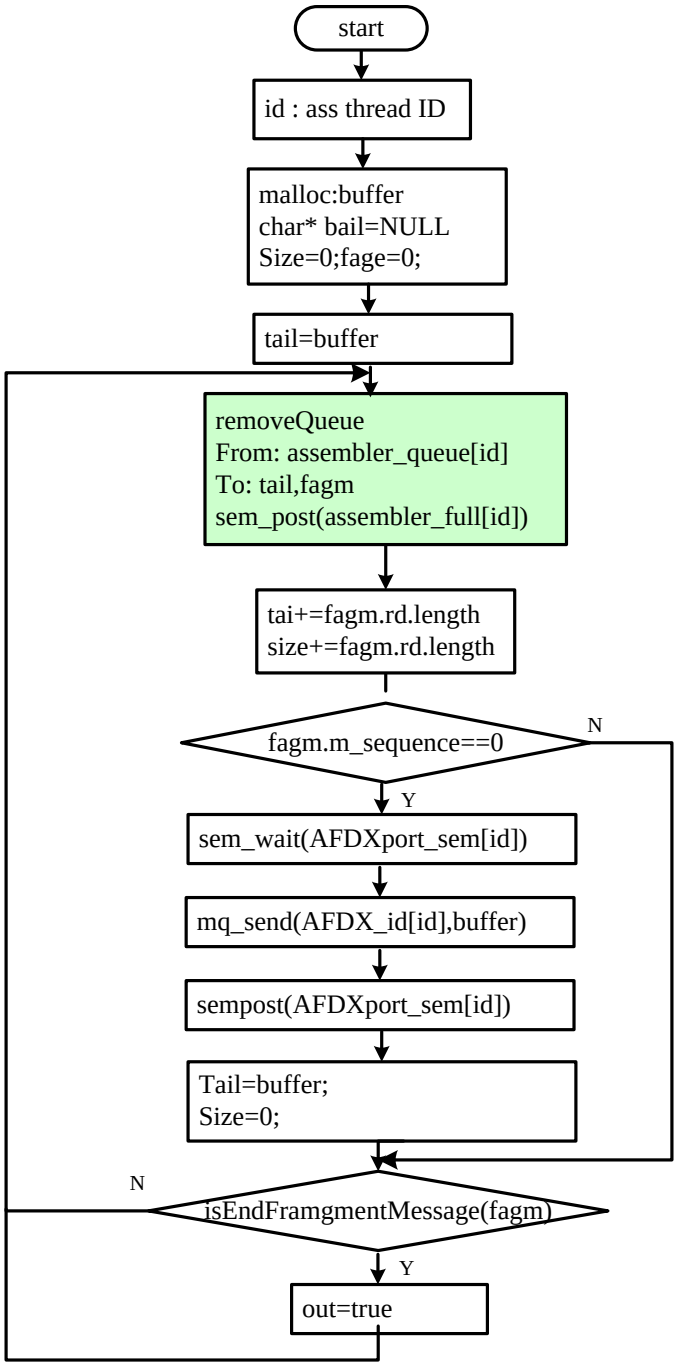

Figure 3-19 Assembler thread flowchart

Af and raf process

The af and raf process initialize the framework and Dataset by calling the function *initializeFramework*() and *initializeIM*() at the beginning, and destroy them in the end. *Af* process calls sendMessage() to send the data, and raf

process calls the receiveMessage() to get the data. The flowchart is showed in Figure3-18.

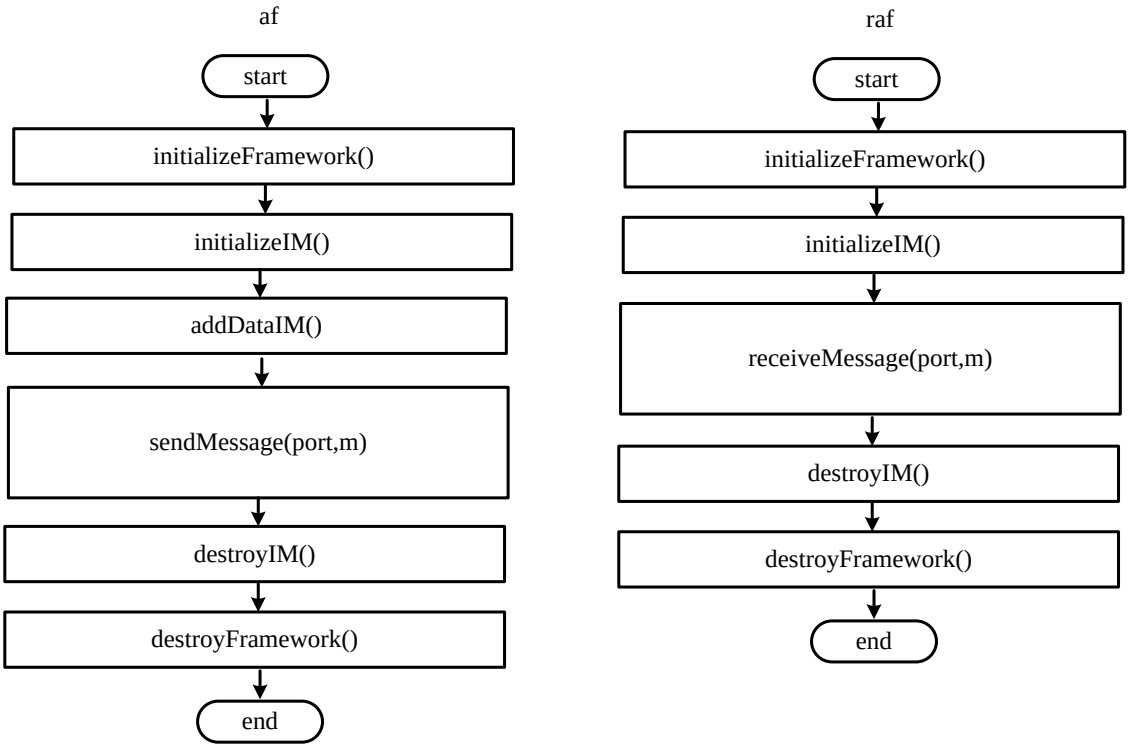

Figure 3-20 Af and raf process flowchart

Queue management

The simulation framework manages queues effectively. It uses the FIFO principle. The management logic depends on six scenarios that are showed in Figure3-19.

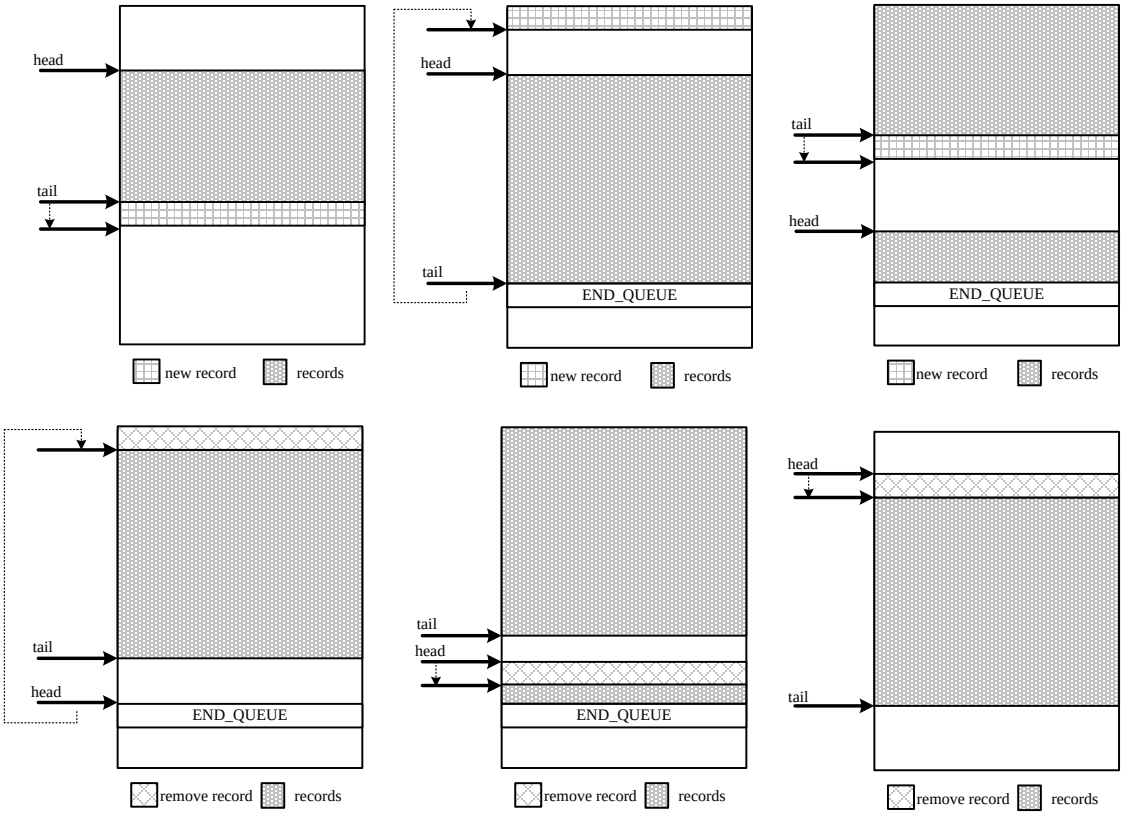

Figure 3-21 Queue management

The top three sub figures show the process of adding data into a queue, and the lower three sub figures show the process of removing data from a queue. The *END_QUEUE* flag indicates the end of queues, which means queues are full and it can not record any data. Each queue has two pointers: *tail* and *head*. A new data is recorded into the address space beginning with the *tail*. On the contrary, a record will be removed from the buffer beginning with the *head* pointer. Before adding a new data into buffer, the queue management module will check whether the *head* pointer points the *END_QUEUE* flag. If it is true, the *head* will point to the beginning buffer address and add data in here (showed in left bottom sub figure in Figure3-19).

3.7 Running and testing

3.7.1 Building system

The software adopts the GNU make utility as a program tool. *Makefile* files are used to compile the framework.

The first section of the *makefile* includes some symbols and flags which the framework circumstances (hard real-time or soft) are needed.

The command *make* compiles modules into the executable entities, which include:

- afdx.
- af.
- raf.
- createDB.

3.7.2 Framework execution

Before launching the AFDX framework, an installation simulation circumstance process runs at first. The details RTAI install introduction can be found in [9] and [10]. The brief running steps are listed here:

1) To Install and configure RTAI/net, the details are illustrated in [9]
2) To run the command *Make* in order to make the executable program.
3) To execute the *./createDB*, which can create the database file.
4) To execute *./afdx* process to run the framework.
5) To execute *./af* process to run avionics sending test functions.
6) To execute *. /raf* process to run avionics receiving test functions.
7) Check the results and progress records during the simulation.

3.7.3 Test phase

During the test process, the framework has three ways to test if the software behaviours are good or not.

- Standard console output (*stderr ,stdout*): In debug phase, it is used to print the information and state of the simulation process on the screen. The problem is that this method needs time. Printing process will effect the real time performance.
- Kernel buffer: Program calls the function *printk*() to record the output information of simulation behaviours. After running process, the kernel log file will be showed by the command *dmesg*.
- Log file: The log module records the process output information.

4 Performance Analysis

4.1 Network Calculus theory

Network Calculus is an Internet theory of the queuing system in the deterministic way. Network Calculus is developed for the deep research into network flow problems. The foundation is the mathematical theory of dioids, and it is called the Min-Plus dioids or Min-Plus algebra. Using network calculus, a better understanding form some fundamental properties can be got on the integrated services network, such as the backlog buffer or delay dimensioning. It has been used in the A380 design process.

4.2 Basic Min-plus Calculus

The basic Min-plus calculus is a concept when addition is replaced by infimum, and multiplication is replaced by addition. In conventional algebra, the two most common operations on elements of $\mathbb{Z}$ and $\mathbb{R}$ are their addition and their multiplication. Here consider about another algebra, where the operations are changed as follows: addition becomes computation of the minimum, multiplication becomes addition. This is the Min-plus calculus.

The Min-plus calculus includes so many definitions and theorems, such as the concave, convex and start-shaped functions, min-plus convolution, sub-additive functions, sub-additive closure and Min-plus deconvolution. The deconvolution operator allows to easily express two very important quantities in the network calculus, which are the maximal vertical and the horizontal deviation between two curves. That is the fundamental method of the NC three bound calculation. The main definitions and theorems in Min-plus calculus are introduced in Appendix A.

4.3 AFDX Network Performance

4.3.1 Introduction

In the avionics field, flight data must be punctual and reliable delivered. It is necessary to estimate whether the AFDX network can achieve real-time requirements or not. A maximum network transmission limitation must be guaranteed, so an effective method requires to be deduced to evaluate the network performance, including three bounds(end-to-end latency, backlog and output flow), based on the NC.

In general, the actual industrial AFDX is very complex, for example in A380, it has at least 100 end systems with two same networks, and each of them includes 8 switches, with over 1000 virtual links. The A380 avionics architecture design figure is illustrated in Figure4-1.

A380 avionics systems have five areas, including Fuel&LG area, Cabin area, Energy area, Flight control area and Cockpit area. It uses a symmetrical structure and there are a large number of Virtual Links in it. The questions is whether this AFDX data network architecture design could afford the heave transmission load so that the maximum length messages crisscross and transmit between so many end systems in the network. AFDX VL Schedulers deliver data through Virtual Link according to AFDX behaviours, such as Lmax, BAG. It fragments a message into so many units which length is lower than Lmax, and delivers the fragmented units to the destination end systems within a BAG time interval. So a VL scheduler controls each virtual link within its assigned bandwidth, BAG and Lmax Limitation. However, if the assigned bandwidth, BAG and Lmax are not appropriate, it is possible to cause the unacceptable delay and loss data.

For example, in Figure4-1, if the FM1 want to send data to the IOM in the Cockpit area, it would probably through the two ways: FM1-SWITCH1-SWITCH2-IOM or FM1-SWITCH1-SWITCH3-SWITCH2-IOM. No matter which ways, the data from the end system FM1 maybe wait in switch1 input buffer due to multi flow congestion from other end systems, such as SFCC1,FCSC1,

FCGC1. So the messages from FM1 could deliver with the unacceptable delay. The maximum end-to-end delay would be over the upper limitation of AFDX network configurations. In the meantime, the data need to be saved in the switch1 input buffer waiting for their transmission due to the multi flow congestion. If the buffer size is not bigger enough to hold these data, it would cause the loss data. The maximum switches buffer size depends on the backlog. It is very important for avionics designers to find a way to evaluate the network performance, especially maximum end-to-end latency and switch backlog according to BAG and Lmax and assigned bandwidth. They can use this method to guide the design process.

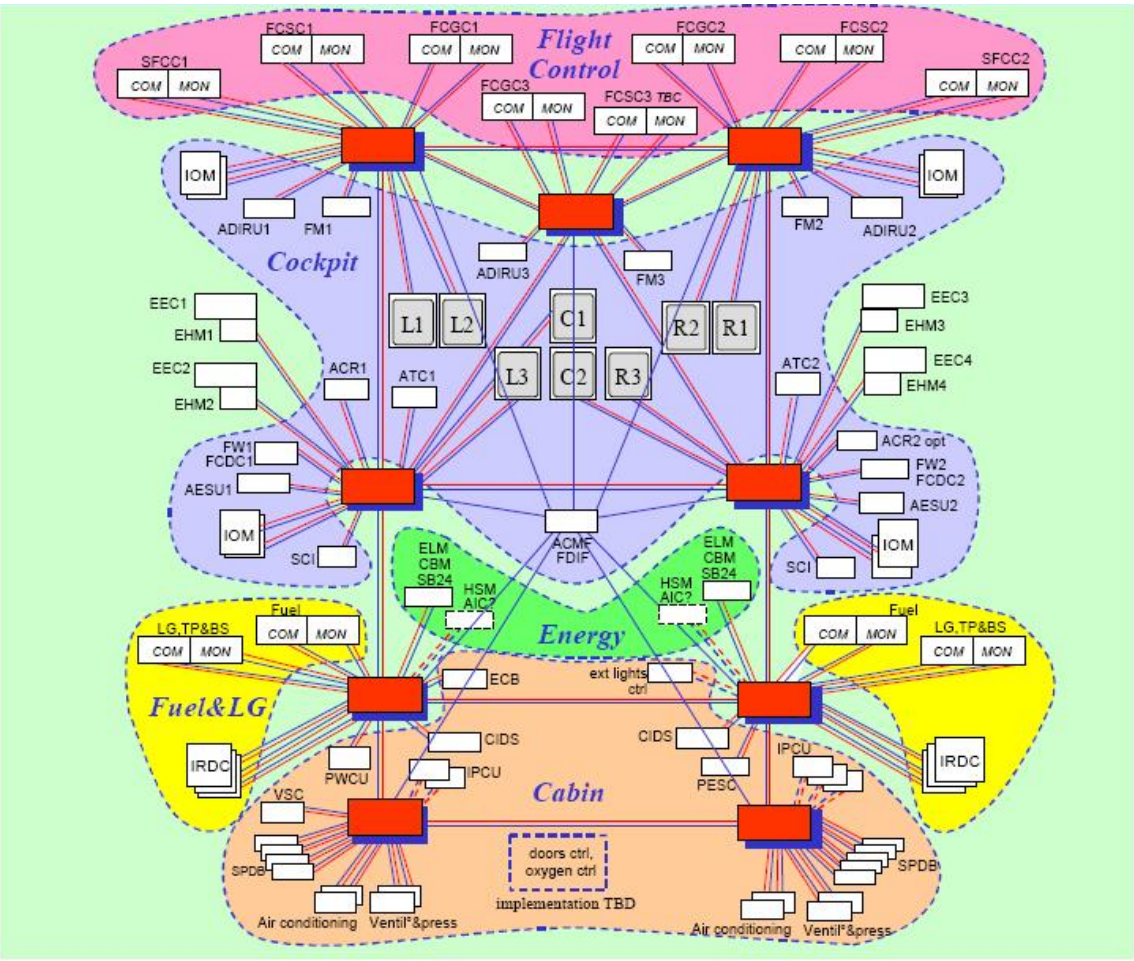

Figure 4-1 A380 avionics architecture topology [28]

Network performance can be assessed the simulation method for a given set of scenarios, but it cannot consider every cases. So results from the simulation method are not complete. Network Calculus can establish a sophisticated network performance verification method for the worst-case scenario, through which, the pessimistic upper bound of the network delay and maximum backlog can be calculated.

All in all, in this section, the network calculus is used to calculate of the maximum end-to-end delay and the backlog buffer size and the minimum output flow in AFDX.

4.3.2 Calculation Approach for three important parameters of network performance

In this thesis, the calculation approach is deduced on the foundation of NC theory. As we have discussed above, NC theory has been successfully used in network analysis, of which Min-plus Calculus theory is the basis, and it has been used in the A380 network design process.

Cumulative Function

$R(t)$, $R^*(t)$ refers as the input and output cumulative function. It means the cumulative function $R(t)$ and $R^*(t)$describes the input and output data flows and it is always wide-sense increasing. Figure4-2 illustrates these definitions.

According to NC theory, a flow which is described by a wide-sense increasing function $R(t)$, unless otherwise specified, is consider as the following three types of models:

- discrete time: $t \in \mathbb{N} = \{0,1,2,3, \dots\}$
- fluid model: $t \in \mathbb{R} = [0, +\infty)$ and R is a continuous function
- continuous time model: $t \in \mathbb{R}^+$ and R is a left- or right-continuous function.

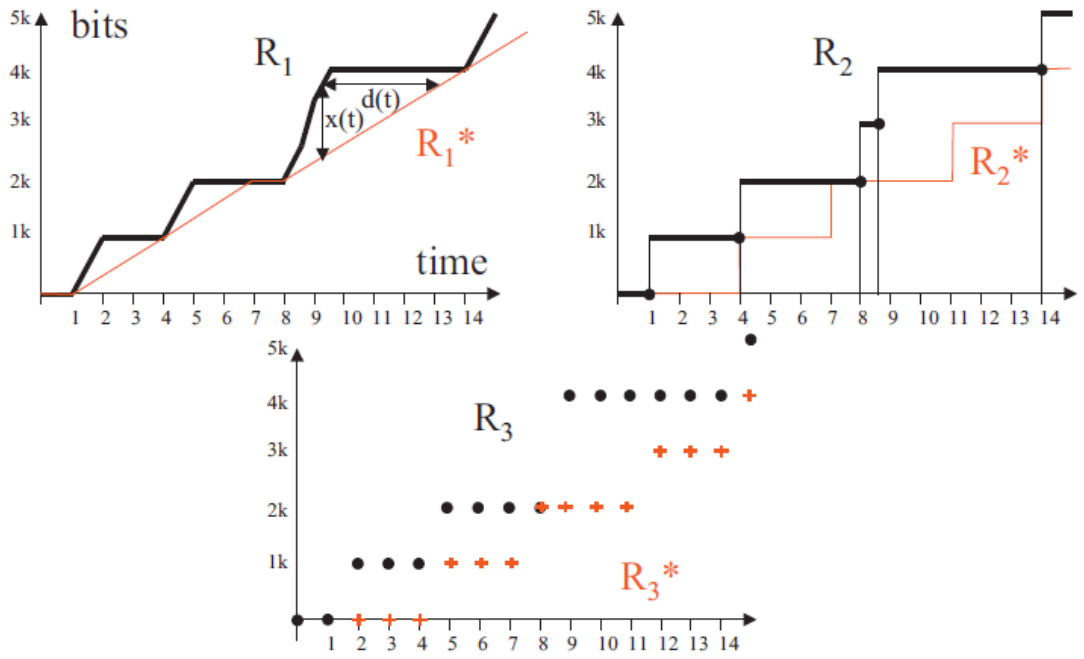

Figure 4-2 Examples of Input function[17].

In Figure4-2, R_1 and R_1^* are input and output function of continuous time (fluid model); R_2 and R_2^* are continuous time with discontinuities; R_3 and R_3^* are discrete time model.

Arrival Curves

The $R(t)$ meet certain constrains known as the "arrival curve", which represents the flow properties. The concept of the arrival curve is used to limit the traffic sent by sources in order to provide the limitation to data flows.

DEFINITION 4.1 (ARRIVAL CURVE) [17] Given a wide-sense increasing function α defined for $t \geq 0$ we say that a flow R is constrained by α if and only if for all $s \leq t$:

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

We say that R has α as an arrival curve, or also that R is α-smooth[17].

This is an example for arrival curve, as Figure4.3 illustrates.

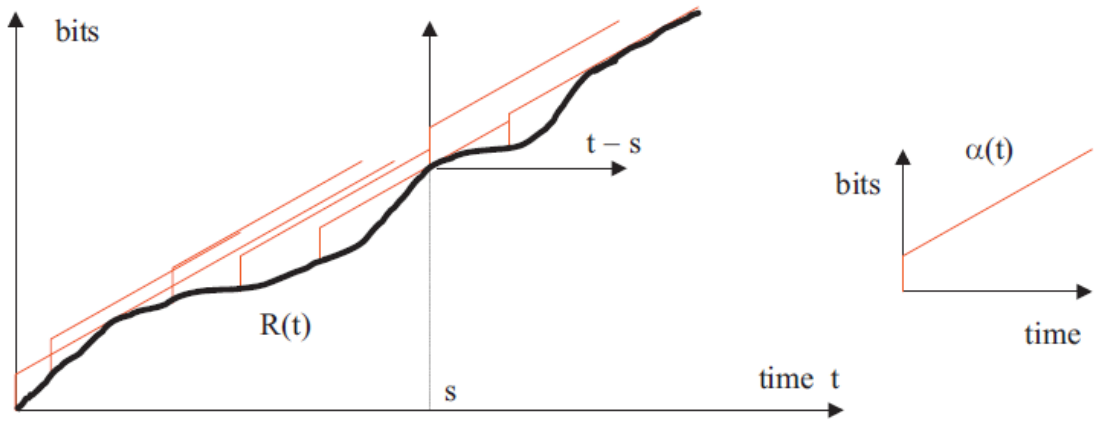

Figure 4-3 Example of the arrival curve $R(t)$ [17]

The arrival curve has associated with the concept of the leaky bucket, which will be described below. According to the ARINC664, the leaky bucket concept could be used in AFDX data network analysis.

A Leaky Bucket controller is a device that analyzes the data on a flow $R(t)$ as follows. There is a pool (bucket) of fluid of size b. The bucket is initially empty. The bucket has a hole and leaks at a rate of r units of fluid per second when it is not empty.

Data from the flow $R(t)$ has to pour into the bucket an amount of fluids equal to the amount of data. Data that would cause the bucket to overflow is declared non-conformant, otherwise the data is declared conformant.

Figure 4-3 illustrates the leaky bucket definition.

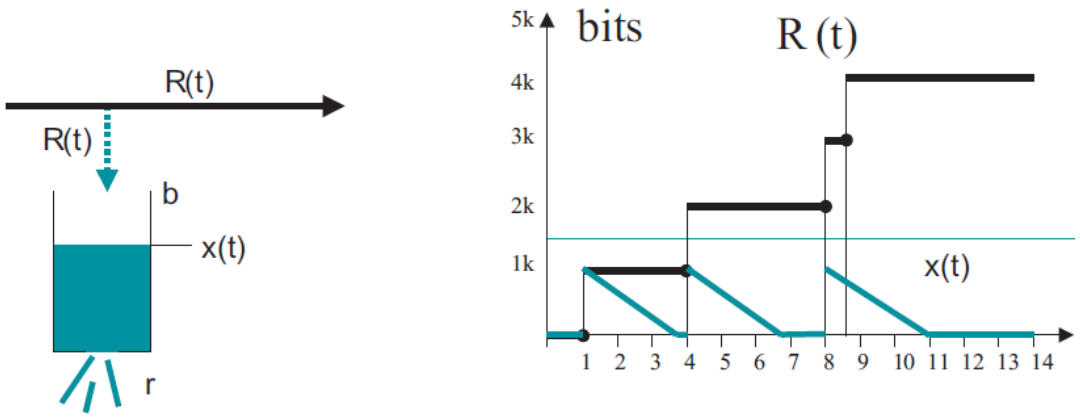

Figure 4-4 Example of leaky bucket definition[17].

Considering a buffer served at a constant rate r, and assuming that the buffer is empty at time 0, and the input is described by a cumulative function $R(t)$, if there is no overflow during $[0, t]$, the buffer content at time t is given by

$$x(t) = \sup_{s:s\le t}\{R(t) - R(s) - r(t-s)\}^{[17]}$$

According to the NC, A leaky bucket controller with leak rate r and bucket size b forces a flow to be constrained by the arrival curve $\gamma_{r,b}$, namely:

- the flow of conformant data has $\gamma_{r,b}$ as an arrival curve[17];
- if the input already has $\gamma_{r,b}$ as an arrival curve, then all data is conformant[17].

Service Curve

The switches service the input flows with "service curve". The output R^* must be above $R \otimes \beta$, which is the lower envelope of all output curves. β is defined by service curve. It represents the limitation of output flows. The service curve is defined as bellow.

DEFINITION 4.2 (SERVICE CURVE) [17] Consider a system S and a flow through S with input and output function R and R^* . We say that S offers to the flow a service curve β if and only if β is wide sense increasing, $\beta(0) = 0$ and $R^* \geq R \otimes \beta$ [17].

Figure4-4 illustrates the definition.

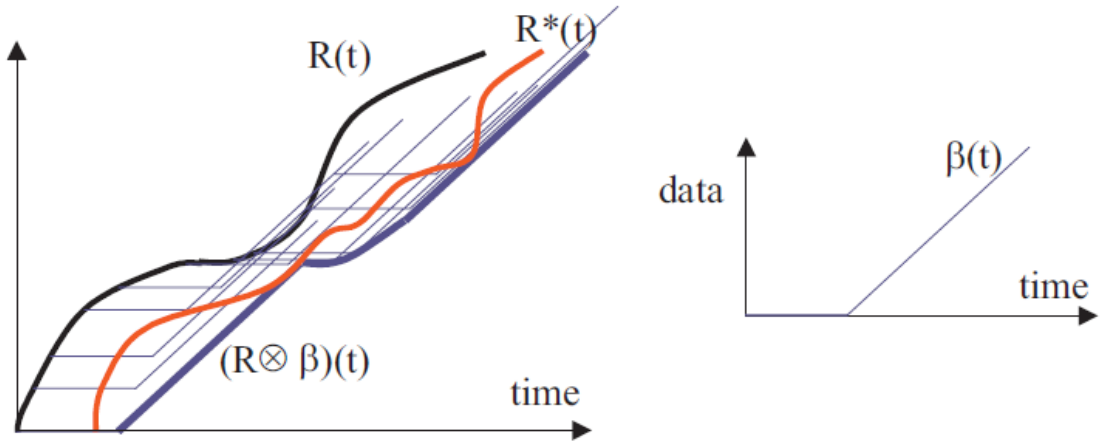

Figure 4-5 Definition of service curve[17].

Here list two classical service curve examples.

(1) Guaranteed delay node

For a lossless bit processing system, saying that the delay for any bit is bounded by some fixed T is equivalent to saying that the system offers to the flow a service curve equal to δ_T .

(2) Premptive priority node

A constant bit rate server, with rate C, serves two flows, H and L, with non-preemptive priority given to flow H. Then the high priority flow is guaranteed a rate-latency service curve with rate C and latency $\frac{l^L_{max}}{C}$ where l^L_{max} is the maximum packet size for the low priority flow.

If in addition the high priority flow is $\gamma_{r,b}$-smooth, with $r < C$, then the low priority flow is guaranteed a rate-latency service curve with rate $C - r$ and latency $\frac{b}{C-r}$.

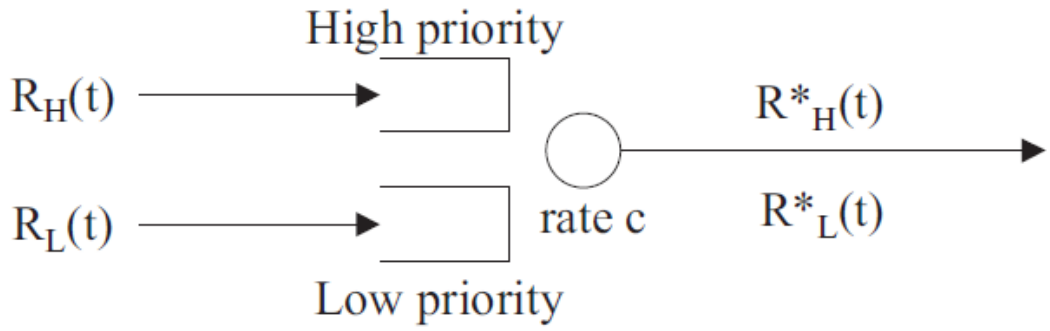

Figure 4-6 Two priority flows (R and L) [17] .

(3) Non premptive priority node

Consider a node serving two flows, H and L, with non-preemptive priority given to flow H. Assume that the node guarantees a strict service curve β to the aggregate of the two flows. Then the high priority flow is guaranteed a service curve $\beta_H(t) = [\beta(t) - l^L_{max}]^+$ where l^L_{max} is the maximum packet size for the low priority flow.

If in addition the high priority flow is α_H-smooth, then define $\beta_L(t)$ by $\beta_L(t) = [\beta(t) - \alpha_H(t)]^+$. If β_L is wide-sense increasing, it is a service curve for the low priority flow.

Three bounds

The three bounds are for lossless systems with service guarantees.

According to definition3.1.14 in appendix B,

- Backlog Bound: is the maximum vertical deviation between two curves: the arrival and service curves. We Assume a flow, constrained by arrival curve α, which traverses a system that offers a service curve β. The backlog $R(t) - R^*(t)$ for all t satisfies:

$$R(t) - R^*(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\}^{[17]}$$

- Delay Bound: is the maximum horizontal deviation between two curves: the arrival and service curves. The mathematical definition is showed as follows. We assume a flow, which is constrained by arrival curve α, traverses a system that offers a service curve of β. The virtual delay $d(t)$ for all t satisfies:

$$d(t) \leq h(\alpha, \beta)^{[17]}$$

- Output curve: is the output flow limitation. We assume a flow, constrained by arrival curve α, traverses a system that offers a service curve of β. The output flow is constrained by the arrival curve:

$$\alpha^* = \alpha \oslash \beta^{[17]}$$

AFDX NC model analysis

(1) Switch service model

As standardized by ARINC 664, the transmission and reception could be managed according to the FIFO discipline. Each switch serves the aggregate of VLs with a constant rate $R = 100Mbps$ transmission capacity. Therefore,

according to the rate-latency function definition, the switch will be modelled by rate-latency curve:

$$\beta_{R,T}(t) = R[t-T]^{+} \quad (1)$$

Where R is the maximum service capacity for aggregate input flows;

T is the maximum time in switches form the input to output buffer.

(2) VL model

According to the ARINC 664, each VL has two parameters.

- Bandwidth Allocation Gap(BAG), represents the maximum time interval of the first bit in each network frame. It ranges from 1 to 128 ms. $BAG = 2^{K}[in\ ms]$ (K integer is in range 0 to 7).
- L_{max} is the largest frame size, in bytes. Its value ranges from 64 to1518 bytes.

Then, according to definition of the leaky bucket function, the each VL arrival curve can be defined by

$$\alpha(t) = [\sigma + \rho t]^{+} \quad (2)$$

Corresponding to the AFDX network, $\sigma = L_{max}$, $\rho = \frac{L_{max}}{BAG}$. The arrival curve of VL is

$$\alpha(t) = \left[L_{max} + \frac{L_{max}}{BAG}t\right]^{+} \quad (3)$$

(3) AFDX three bounds of two case

Two case in AFDX network for three bounds will be listed below:

- A simple example and interpretation of leaky bucket

Assumed a flow constrained by one leaky bucket with an arrival curve of the form $\alpha = \gamma_{r,b}$, the service curve is $\beta_{R,T}$, the three bounds are showed in Figure4-6.

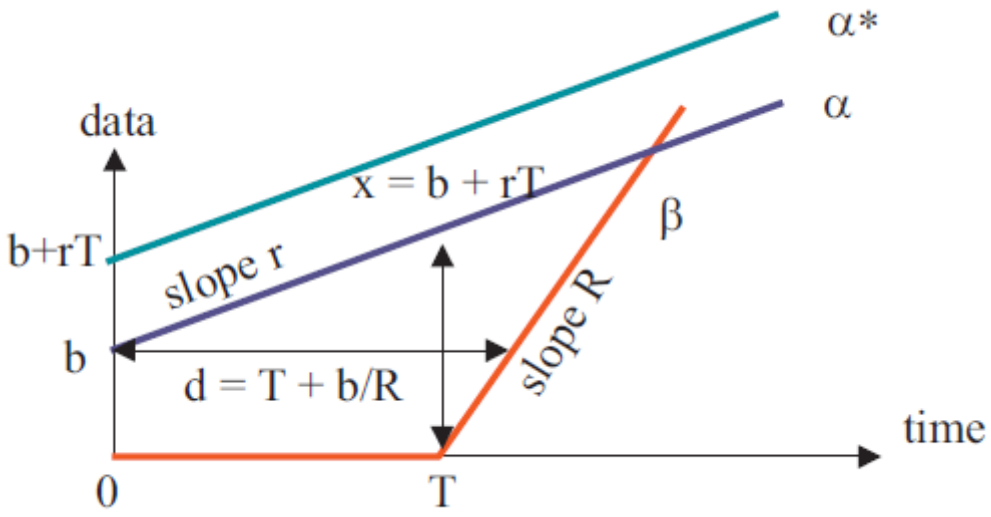

Figure 4-7 Example for three bounds[17]

.

- A VBR(variable bandwidth rate) flow with a rate-latency service curve

In a VBR flow, the flow arrival curve is $\alpha(t) = \min(M + pt, rt + b)$. Assume that the flow is served by the function $\beta = \beta_{R,T}$. Assume that $R \geq r$, the vertical deviation $v = \sup_{s \geq 0}[\alpha(s) - \beta(s)]$ is reached an angular point of either α or β. Thus

$$v = max[\alpha(T), \alpha(\theta) - \beta(\theta)]$$

With $\theta = \frac{b-M}{p-r}$, similarly, the horizontal distance is reached an angular point. Thus, the bound on delay d is given by

$$d = \max\left(\frac{\alpha(\theta)}{R} + T - \theta, \frac{M}{R} + T\right)$$

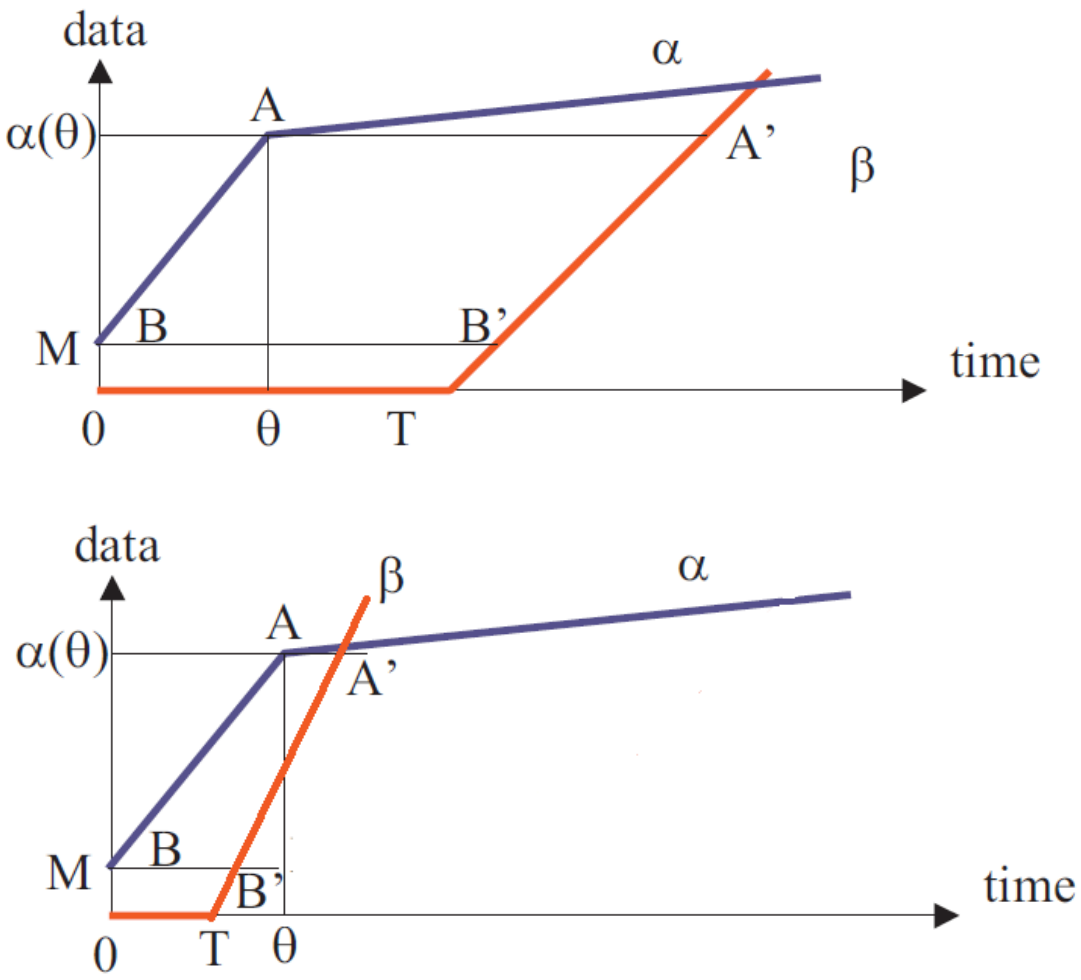

Figure 4-8 VBR flow three bounds

After considering basic network parts, the concatenation of network elements could be considered.

Integration Consideration

In general, the actual industrial AFDX is very complex, for example in A380, it has at least 100 end systems with the two same networks, and each of them includes 8 switches, with over 1000 virtual links. So in order to deduce the integration model, many VLs will be translated into a simple flow structure , and many switches network systems also need to be translated into one switch system. The calculations are described as follows.

(1) One switch service curve

In order to calculate multi-switches structure, the one switch architecture gets the first sight. Assume that each switch provides service for the aggregate input flows $R_i(t)(i = 1,2, \dots n)$, as Figure4-8 shows.

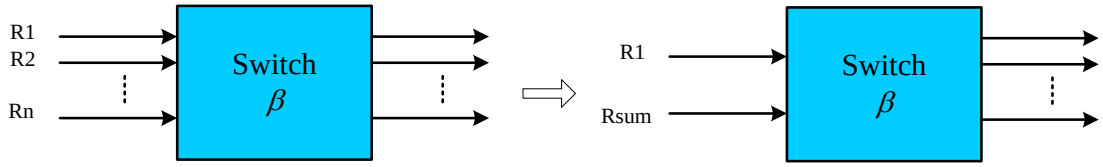

Figure 4-9 Flows go through a switch

The input flows could be divided into two parts: one is $R_i(t)$ which is the one we want to analyze, and the other is $R_{sum}(t)$, depicted in Figure4-8 . Its arrival curve is

$$\alpha_{sum}(t) = [\sigma_{n-1} + \rho_{n-1}t]^+ \tag{4}$$

Where $\sigma_{n-1} = \sum_{j=1}^{n} \sigma_j - \sigma_i$, $\rho_{n-1} = \sum_{j=1}^{n} \rho_j - \rho_i$.

$R_{sum}(t)$ represents the sum of the input flows, apart from the i^{th} input flow.

Based on the discussion above, the multi input flows could be parted into two input flows, and what we need is to calculate the serve curve for the $R_i(t)$ input flow.

If multiplexing input flows aggregate into a network nod, the service curve method can be deduced as follows.

THEOREM 4.3 (BLIND MULTIPLEXING) [17] Consider a node serving two flows, 1 and 2, with some unknown arbitration between the two flows. Assume that the node guarantees a strict service curve β to the aggregate of the two flows. Assume that flow 2 is α_2-smooth. Define $\beta_1(t) = [\beta(t) - \alpha_2(t)]^+$. If β_1 is wide-sense increasing, it is a service curve for flow 1[17].

We consider a node serving two flows in an aggregate manner, and assume the aggregate is guaranteed a strict service curve $\beta_{R,T}$. Assume also that flow i is constrained by one leaky bucket with parameters (ρ_i, σ_i) . If $\rho_1 + \rho_2 \leq R$ the output of the first flow is constrained by a leaky bucket with parameters (ρ_1, b_1^*) with

$$b_1^* = \sigma_1 + \rho_1 T + \rho_1 \frac{\sigma_2 + \rho_2 T}{R - \rho_2} \quad .$$

According to theorem4.3 (blind multiplexing theorem) and the strict serve curve and α-smooth, the switch guarantees a strict service curve β, and flow $R_{n-1}(t)$ is $\alpha_{n-1}(t)$-smooth. As the results,

$$\begin{aligned} \beta_i(t) &= \left[\beta_{R,T}(t) - \alpha_{n-1}(t)\right]^+ \\ &= [R(t-T) - (\rho_{n-1}t + \sigma_{n-1})]^+ \\ &= (R - \rho_{n-1})\left[t - (T + \frac{\rho_{n-1}T + \sigma_{n-1}}{R - \rho_{n-1}})\right]^+ \end{aligned} \qquad (5)$$

is a service curve for flow $R_i(t)$, where $i = 1,2, \ldots n$; $\sigma_{n-1} = \sum_{j=1}^{n} \sigma_j - \sigma_i$, $\rho_{n-1} = \sum_{j=1}^{n} \rho_j - \rho_i$.

(2) Many switches network system

After considering basic network parts, the concatenation of network elements could be considered. Two switches systems are considered can be translated into one switch system.

THEOREM 4.4 (CONCATENATION OF NODES) [17] Assume a flow traverses systems S_1 and S_2 in sequence. Assume that S_i offers a service curve of $\beta_i, i = 1,2$ to the flow. Then the concatenation of the two systems offers a service curve of $\beta_1 \otimes \beta_2$ to the flow[17].

In this case, at first, we think about the two AFDX nodes service curve $\beta_{R_i,T_i}, i = 1,2$. The result is

$$\beta_{R_1,T_1} \otimes \beta_{R_2,T_2} = \beta_{\min(R_1,R_2),T_1+T_2}$$

The definition of $\otimes$ is in the appendix A.1(DEFINITION 3.1.9).

The concatenation theorem has a character of “Pay Bursts Only Once”. Assume the two nodes have service curves $\beta_{R_i,T_i}, i = 1,2$ and input flow is limited by $\gamma_{r,b}$. If $r < R_1$ and $r < R_2$, the results are considered by two ways.

(1) by applying the whole network service curve;

(2) By considering of the respective bounds on each node.

In the first method, the delay bound D_0 can be calculated by using Theorem3.1.10:

$$D_0 = \frac{b}{R} + T_0$$

with $R = \min_i (R_0)$ and $T_0 = \sum_i T_i$.

Now adopt the second method. A delay bound on the delay at node 1 is (Theorem 4.3):

$$D_1 = \frac{b}{R_1} + T_1$$

The output of the first node is limited by α^* , given by :

$$\alpha^*(t) = b + r \times (t + T_1)$$

A delay bound at the second buffer is:

$$D_2 = \frac{b + rT_1}{R_2} + T_2$$

And thus

$$D_1 + D_2 = \frac{b}{R_1} + \frac{b + rT_1}{R_2} + T_0$$

It is easy to see $D_0 < D_1 + D_2$. In other words, the bounds obtained by considering the whole service curve are better than the bounds obtained by considering every buffer in isolation.

According to theorem4.4 (concatenation of nodes theorem), the concatenation theorem gives a good understanding of “Pay Bursts Only Once”. Two switches systems are considered can be translated into one switch system by$\beta_1 \otimes \beta_2$. We can also deduce that m switches can be translated into one switch system by

$$\beta = \beta_1 \otimes \beta_2 \otimes \ldots\ldots \otimes \beta_m \qquad (6)$$

The results show in Figure4-9.

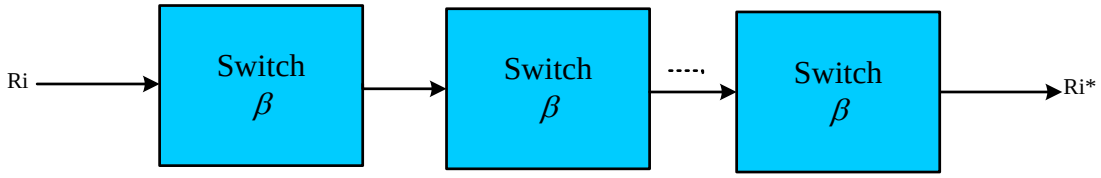

Figure 4-10 A flow go through m switches

Suppose the service curves of m switches are $\beta_{R_i,T_i}(i = 1,2,\dots m)$ respectively, then the global service curve is

$$\beta = \beta_{R_1,T_1} \otimes \beta_{R_2,T_2} \otimes \dots \otimes \beta_{R_m,T_m} \quad (7)$$

According to the convex function definition (definition3.1.7 in appendixA), the rate-latency function is convex, also according to theorem 3.1.4(General properties of ⊗) and its Rule9, If f and g are and piecewise linear, $f \otimes g$ will be generated through this way: putting the linear pieces of f and g together by the order of increasing slopes. Figure4-10 shows an example for explaining the meaning of Rule9.

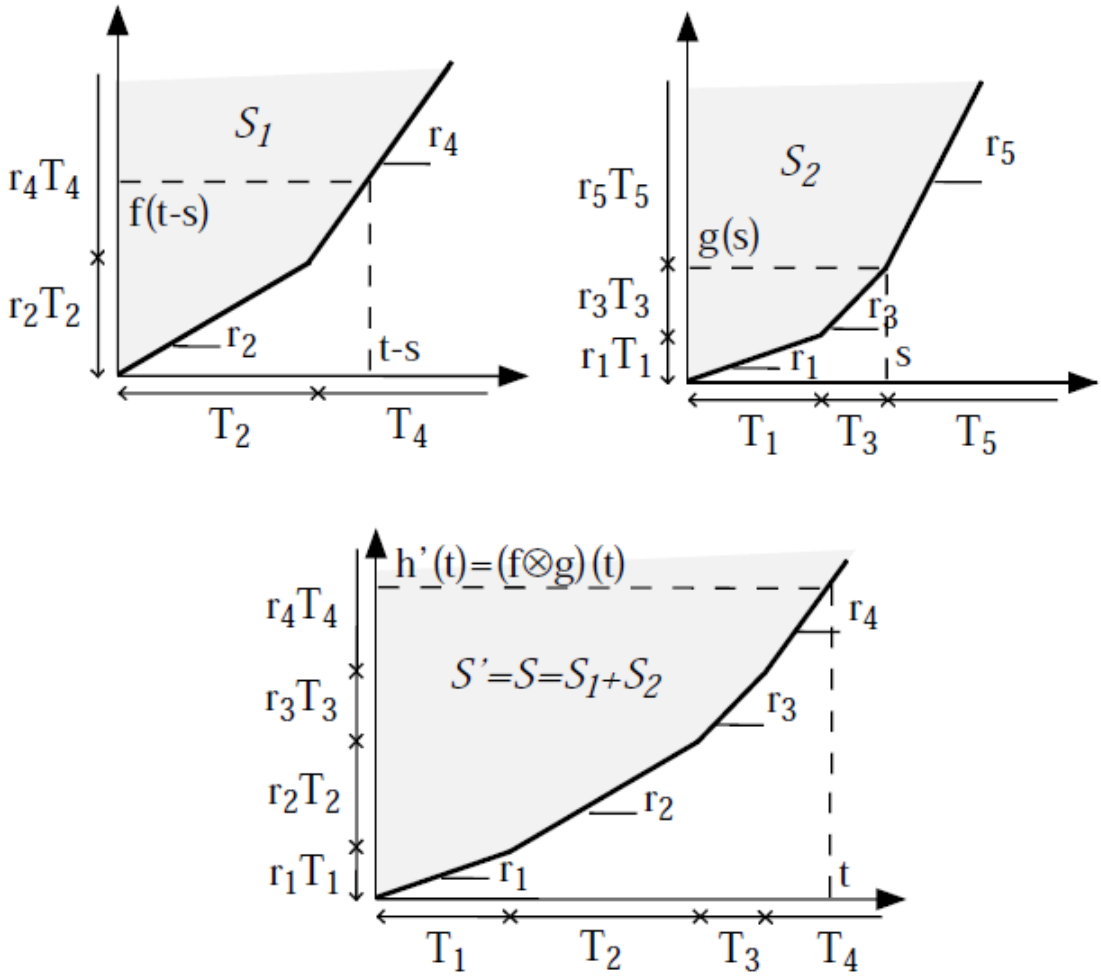

Figure 4-11 Example of $f \otimes g$ [17].

In Figure4-10, h'(t)= $f \otimes g$, the upper two figures show the curve of f and g . The bellow figure shows the results of $f \otimes g$ – the curve of h'(t). The figure

shows the regulation of this calculation. The r1 has the lowest sloping than any other lines, so it is the first line in the curve of the h'(t). In this way, the r2 is the second and r3 is the third. In r4 and r5 are the unlimited line, and one of them will be the last one. The r4 has the lower sloping, so it is the last unlimited line.

According to the calculation we discussed above, the results for $\beta_{R_1,T_1} \otimes \beta_{R_2,T_2}$ in AFDX is showed in Figure4-11.

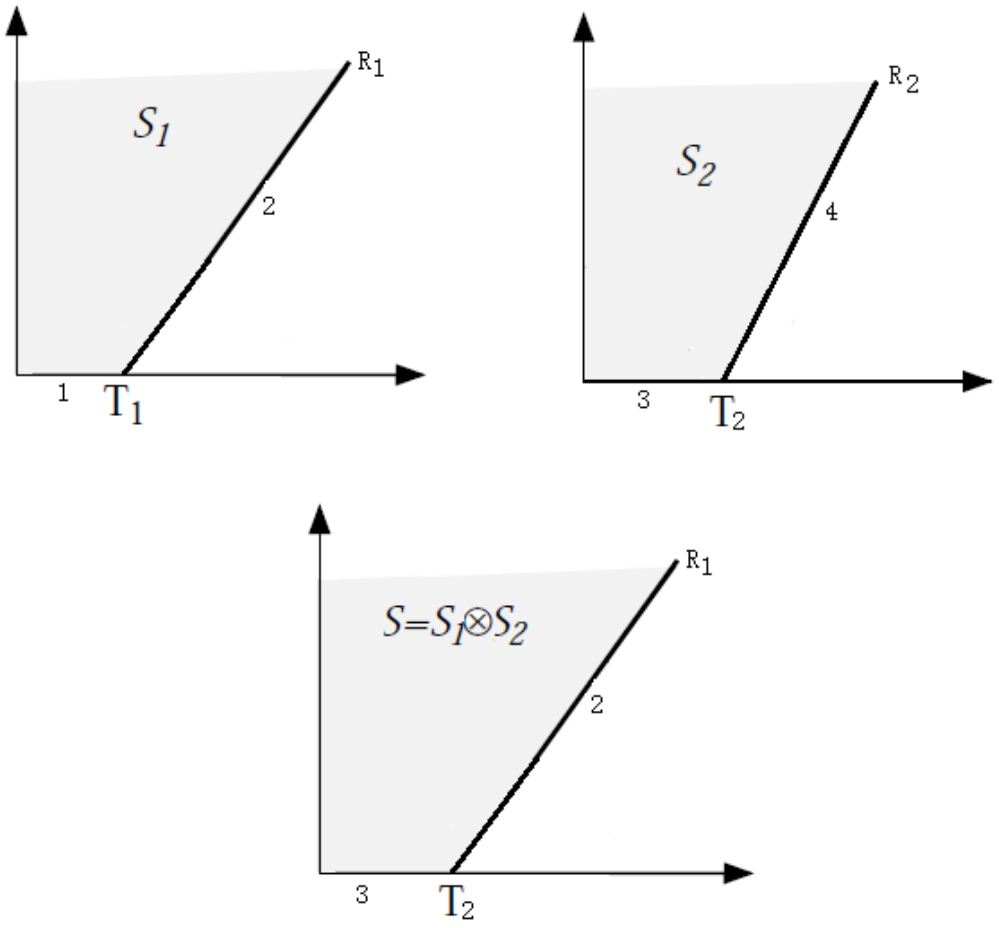

Figure 4-12 $\beta_{R_1,T_1} \otimes \beta_{R_2,T_2}$

So,

$$\beta_{R,T} = \beta_{R_1,T_1} \otimes \beta_{R_2,T_2} \otimes \ldots \otimes \beta_{R_m,T_m} \quad (8)$$

Where $R = \min\{R_1, R_2, \ldots R_m\}$, $T = T_1 + T_2 + \cdots + T_m$.

As a conclusion, for i^{th} input flow, the service curves on m switches are

$$\beta_i(t) = (R - \rho_{n-1})\left[t - (T + \frac{\rho_{n-1}T + \sigma_{n-1}}{R - \rho_{n-1}})\right]^+ \quad (9)$$

Where $R = \min\{R_1, R_2, \ldots R_m\}$, $T = T_1 + T_2 + \cdots + T_m$.

AFDX Three Bounds

According to three bounds definition in (theorem3.1.9, theorem 3.1.10, theorem 3.1.11 in appendix A), assuming the input flow is α with arrival curve $\alpha = \gamma_{r,b}$ served in switches system which the service curve is $\beta_{R,T}$, the three bounds and finding results are showed in Figure4-6.

So in this case,

- Backlog buffer bound is equal to $b + rT$;
- Delay bound is equal to $T + \frac{b}{R}$;
- Output flow is limited by the arrival curve $\gamma_{r,b} \oslash \beta_{R,T} = \gamma_{r,b+rT}$. ($\gamma_{r,b} \oslash \beta_{R,T}$ is shown in FigureB-3. When $0 < r < R$ and $t \geq 0$, $\gamma_{r,b} \oslash \beta_{R,T} = \gamma_{r,b+rT}$).

End-to-end latency

Assumes DF_{PX} is the end-to-end latency of each virtual link, and the frame F_{PX} is transmitted on the px path:

$$DF_{px} = YDF_{px} + LDF_{px} + SDF_{px} \quad (10)$$

Where:

- YDF_{px} is the pre-processing time in the ES, and assumes this value probably equal to zero in this thesis, due to the computer high performance. The applications in end system run fast, so it is enough to deal with the AFDX data sending mission in the tiny time interval.
- $LDF_{px}$is the transmission delay on data physical links. The full duplex AFDX has no collisions happened on physical links. Thus, the latency on one link is $c \times s_{F_{px}}$, where c is bandwidth and $s_{F_{px}}$ is length of F_{px}. So, $LDF_{px} = nbl_{px} \times c \times s_{F_{px}}$, where $nbl_{px}$is the number of actual links. Here consider it as 0.5μs in the worst-case scenario.

- SDF_{px} is the delay in switches, which is variable. To get frame latency, SDF_{px} is analyzed in following part on the NC theory.

According to the equation(4) and(9), the delay bound is

$$SDF_{px} \leq T + \frac{\rho_{n-1}T+\sigma_{n-1}}{R-\rho_{n-1}} + \frac{\sigma_i}{R-\rho_{n-1}} \quad (11)$$

Where

- $i = 1,2,\dots n$;
- $R = \min\{R_1, R_2, \dots R_m\}$, in AFDX, the rate R is $12.5 \times 10^6 B/S$;
- $T = T_1 + T_2 + \cdots + T_m$, according ARINC 664, Part 7, the delay in switch is limited by $16\mu s$,so $T = m \times 16\mu s$;
- $\sigma_i = L_{max}$, $\rho_i = \frac{L_{max}}{BAG}$;
- $\sigma_{n-1} = \sum_{j=1}^{n} \sigma_j - \sigma_i$;
- $\rho_{n-1} = \sum_{j=1}^{n} \rho_j - \rho_i$.

Buffer bound

According to the equation(4) and (9), the backlog buffer bound is

$$H_{max} = \sigma_i + \rho_i \times (T + \frac{\rho_{n-1}T+\sigma_{n-1}}{R-\rho_{n-1}}) \quad (12)$$

Where

- $i = 1,2,\dots n$;
- $R = \min\{R_1, R_2, \dots R_m\}$, in AFDX, the rate R is $12.5 \times 10^6 B/S$;
- $T = T_1 + T_2 + \cdots + T_m$, according ARINC 664, Part 7, the delay in switch is limited by $16\mu s$, so $T = m \times 16\mu s$;
- $\sigma_i = L_{max}$, $\rho_i = \frac{L_{max}}{BAG}$;
- $\sigma_{n-1} = \sum_{j=1}^{n} \sigma_j - \sigma_i$;
- $\rho_{n-1} = \sum_{j=1}^{n} \rho_j - \rho_i$.

The output flow

According to the equation (9) and (4),the output flow arrival curve is:

$$\alpha^* = \left[\rho_i t + (\sigma_i + \rho_i \times (T + \frac{\rho_{n-1}T+\sigma_{n-1}}{R-\rho_{n-1}}))\right]^+ \quad (13)$$

Where

- $i = 1,2, \dots n$;
- $R = \min\{R_1, R_2, \dots R_m\}$, in AFDX, the rate R is $12.5 \times 10^6 B/S$;
- $T = T_1 + T_2 + \cdots + T_m$, according ARINC 664, Part 7, the delay in switch is limited by $16\mu s$, so $T = m \times 16\mu s$;
- $\sigma_i = L_{max}$, $\rho_i = \frac{L_{max}}{BAG}$;
- $\sigma_{n-1} = \sum_{j=1}^{n} \sigma_j - \sigma_i$;
- $\rho_{n-1} = \sum_{j=1}^{n} \rho_j - \rho_i$.

5 Simulation and Comparing Results

5.1 Mathematic results analysis

This section analyses the mathematical method of the AFDX network performance in some concrete cases. These cases assume that there are 8 switches in the physical link path of VL_1 , which is the virtual link planned to be analysed. Each switch has 8 average VL input flows.

The calculation and results output for the delay and backlog buffer size in these specific cases requires a *verification* software tool, which has been developed by the author.

The input parameters are BAG and L_{max} . They are the BAG and Lmax of the 8 Virtual Links(from VL_1 to VL_8). The output parameters are SDF_{pxi} and H_{maxi} . SDF_{pxi} is the main part of the End-to-End system delay of the VL_1 . H_{maxi} is the main part of the minimum switches input buffer size of the VL_1 .

1) Case 1: we assume all of virtual links, $BAG = 1(ms)$, $L_{max} = 1518(byte)$, except VL_1 . The values of L_{max1} of VL_1 range from 64 to 1518(ms) bytes.

Table5-1 shows the value range of the input and output parameters of the case1. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-1 Value range of the input and output parameters of the case1

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1	64~1518	6.481~7.231	427~10127
VL_2	1	1518		
VL_3	1	1518		
VL_4	1	1518		
VL_5	1	1518		
VL_6	1	1518		
VL_7	1	1518		
VL_8	1	1518		

In Figure5-1, the x axis represents the value of L_{max1} , and the y axis represents the calculated results of SDF_{px1} and H_{max1} .

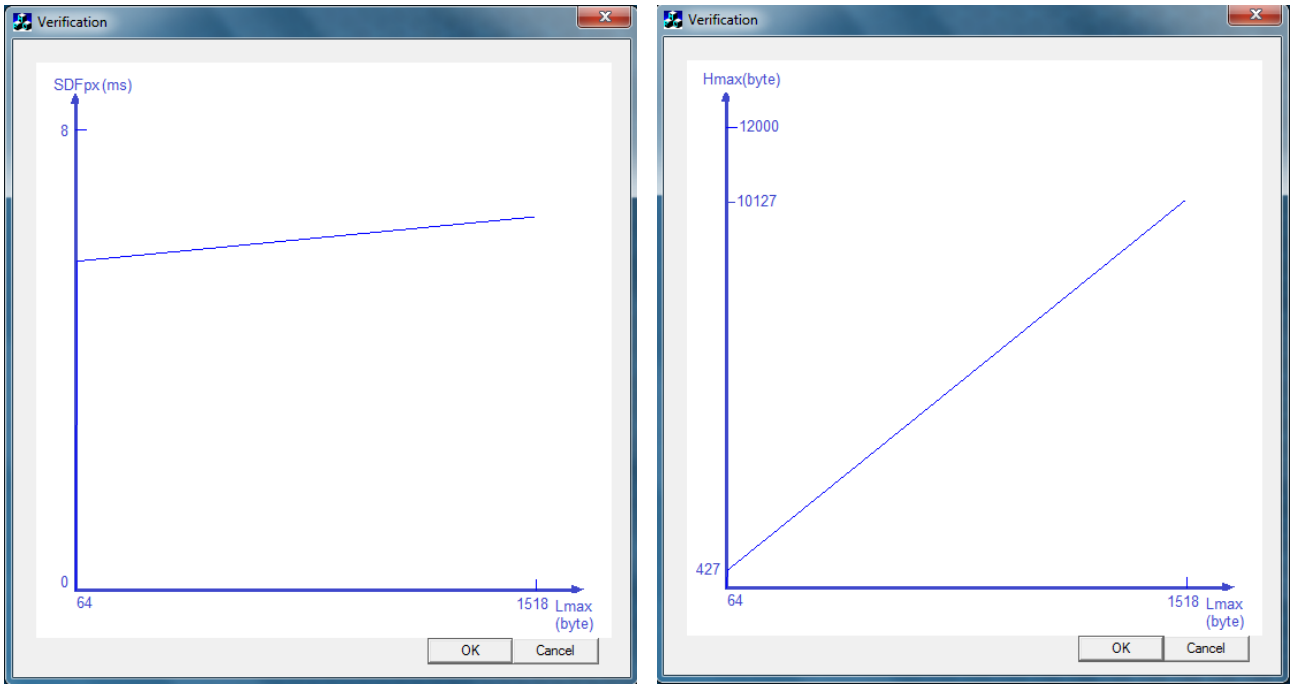

Figure 5-1 Calculated results of SDF_{px1} and H_{max1} in case1

In this case, we could see the value of the end-to-end latency have linear relationship with its Virtual Link Lmax. When the Lmax of VL_1 increases from 64 to 1518 and others are equal to 1518, the delay would increases from 6.481 to 7.231.

We could also see the value of the backlog have linear relationship with its Virtual Link Lmax. When the Lmax of VL_1 increases from 64 to 1518 and others are equal to 1518, the backlog would increases from 427 to 10127bytes.

2) Case 2: assume, about all of the virtual links, $BAG = 1(ms)$,$L_{max} = 1518(byte)$, except VL_1 . The values of BAG_1 of VL_1 range from 1 to 128(ms) (2^K: $K = 1\sim7$).

Table5-2 shows the value range of the input and output parameters of the case2. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-2 Value range of the input and output parameters of the case2

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1~128	1518	6.481	10126~1585
VL_2	1	1518		
VL_3	1	1518		
VL_4	1	1518		
VL_5	1	1518		
VL_6	1	1518		
VL_7	1	1518		
VL_8	1	1518		

In Figure5-2, the x axis represents the value of BAG_1 , and the y axis represents the calculated results of SDF_{px1} and H_{max1} .

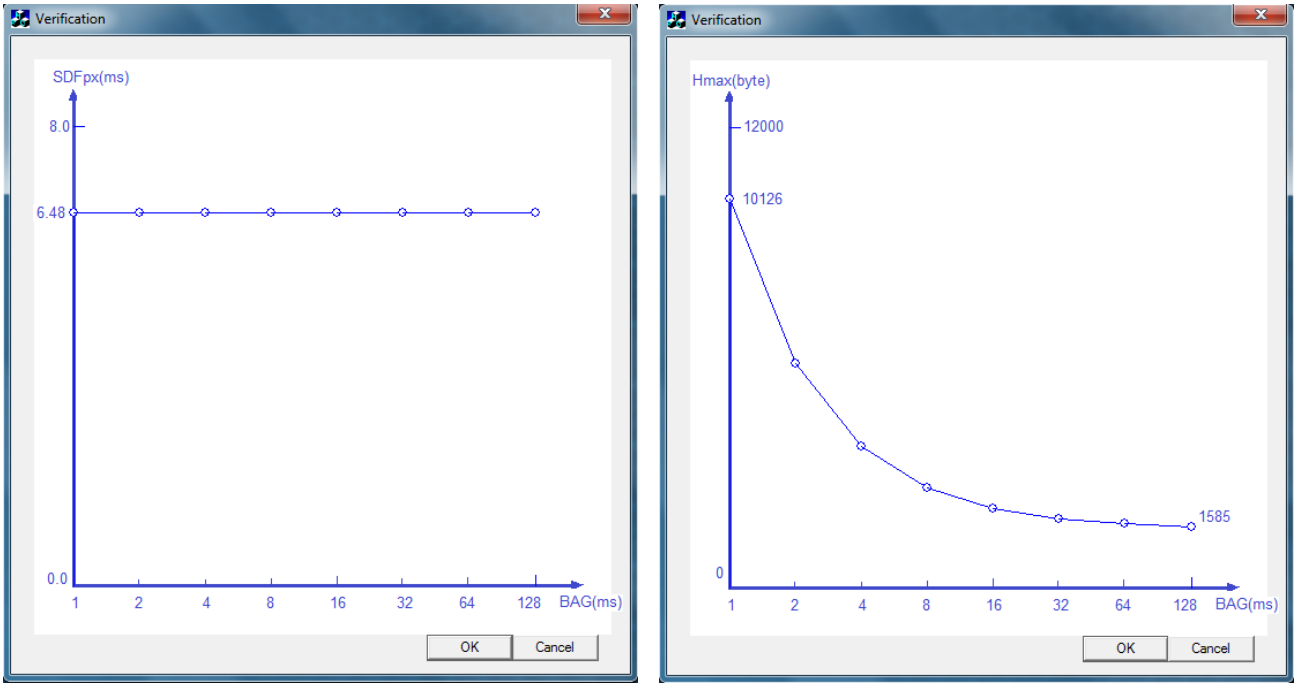

Figure 5-2 Calculated results of SDF_{pxi} and H_{maxi} in case2

In this case, we could see the value of the end-to-end latency remains stable 6.48. When the BAG of VL_1 increases from 1 to 128 and others equal to 1, the delay equals to 6.48.

We could also see the value of the backlog has relationship with its Virtual Link BAG. When the BAG of VL_1 increases from 1 to 128 and others equal to 1518, the backlog would decreases from 10126 to 1585 bytes.

3) Case3: assume, about all of the virtual links, $\mathrm{BAG} = 1(\mathrm{ms})$, $\mathrm{L_{max}} = 1518(\mathrm{byte})$, including VL_1 . But VL_m , one of the other 7 virtual links, its L_{maxm} value rangse from 64 to 1518(byte). The VL_1 delay and backlog size calculated results are showed in Figure5-3.

Table5-3 shows the value range of the input and output parameters of the case3. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-3 Value range of the input and output parameters of the case3

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1	1518	3.213~6.481	5702~10127
VL_2	1	1518		
VL_3	1	64~1518		
VL_4	1	1518		
VL_5	1	1518		
VL_6	1	1518		
VL_7	1	1518		
VL_8	1	1518		

In Figure5-3, the x axis represents the value of L_{maxm}, and the y axis represents the calculated results of SDF_{pxi} and H_{maxi} .

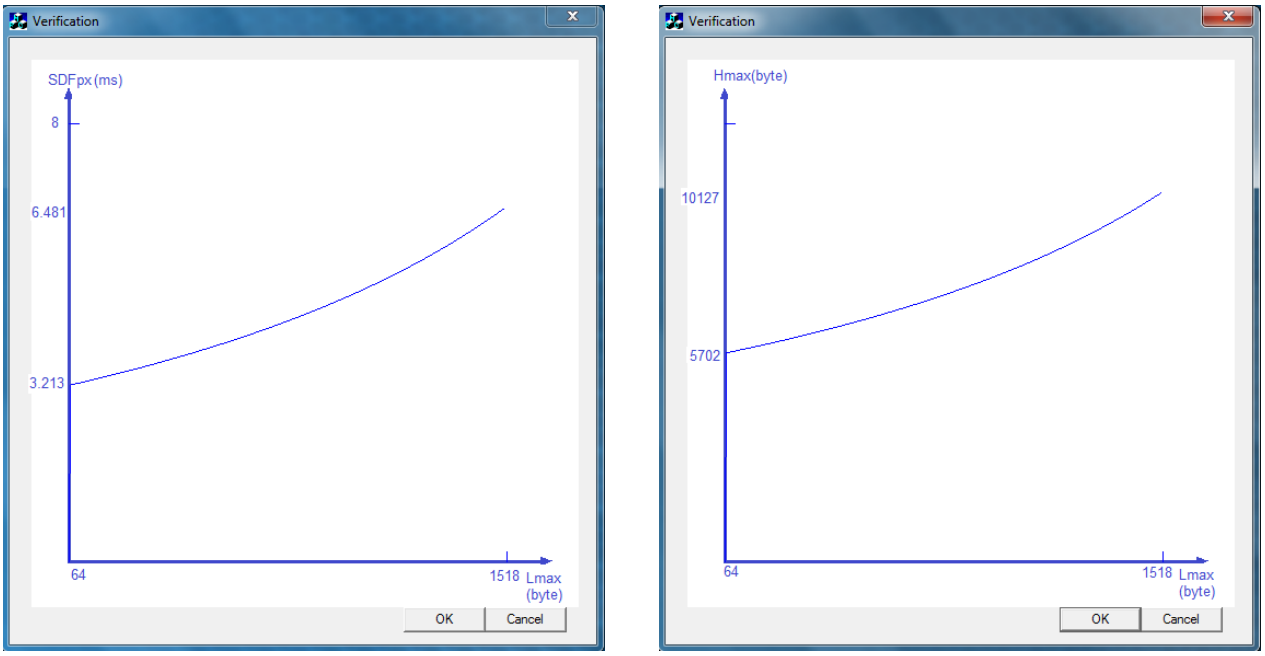

Figure 5-3 Calculated results of SDF_{pxi} and H_{maxi} in case3

In this case, we could see the value of the end-to-end latency have relationship with the other Virtual Link Lmax. When the Lmax of $\mathrm{VL_m}$ increases from 64 to 1518 and VL_1's Lmax equal to 1518, the delay would increases from 3.213 to 6.48.

We could also see the value of the backlog have relationship with its Virtual Link Lmax. When another Lmax of VL_m increases from 64 to 1518 and the Lmax of VL_1 equal to 1518, the backlog would increases from 5702 to 10127 bytes.

4) Case 4: we assume, about all of the virtual links, $BAG = 1(ms)$,$L_{max} = 1518(byte)$, including $\mathrm{VL_1}$. But, for VL_m , one of the other 7 virtual links, it's BAG_i values are range from 1 to 128(ms) ($2^K: K = 1\sim7$). The VL_i delay and backlog size calculated results are showed in Figure5-4.

Table5-4 shows the value range of the input and output parameters of the case4. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-4 Value range of the input and output parameters of the case4

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1	1518	6.481~3.593	10128~629
VL_2	1	1518		
VL_3	1~128	1518		
VL_4	1	1518		
VL_5	1	1518		
VL_6	1	1518		
VL_7	1	1518		
VL_8	1	1518		

In Figure5-4, the x axis represents the value of BAG_m, and the y axis represents the calculated results of SDF_{pxi} and H_{maxi}.

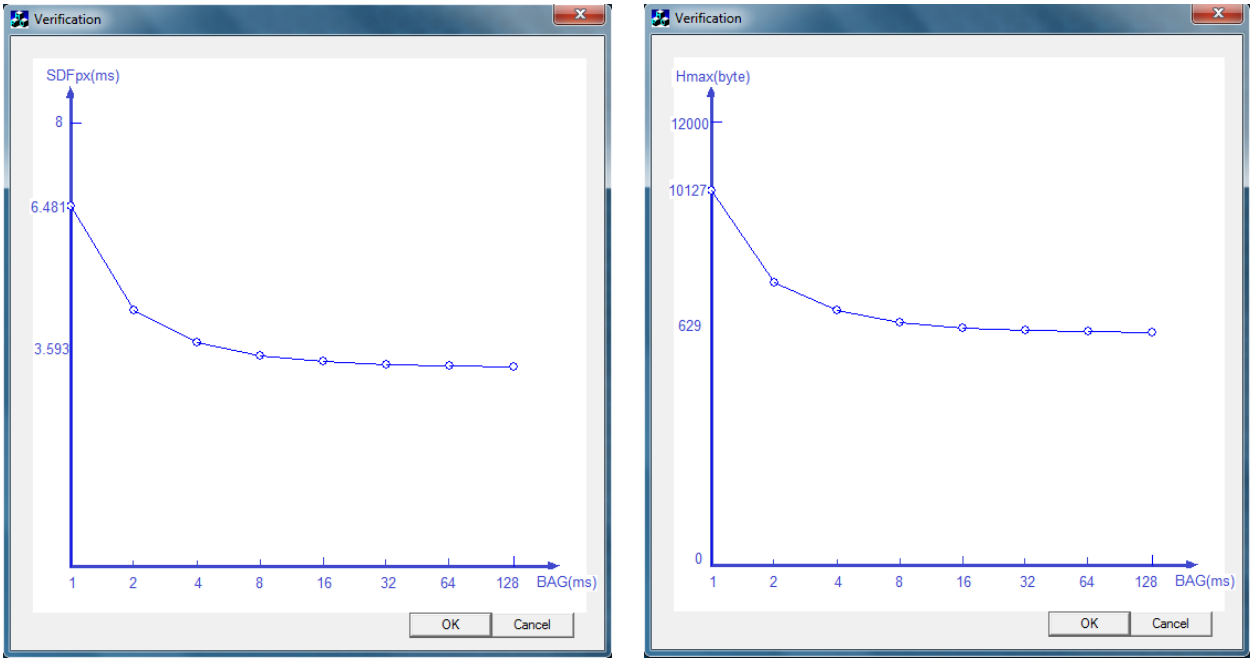

Figure 5-4 Calculated results of SDF_{pxi} and H_{maxi} in case4

In this case, we could see the value of the end-to-end latency have relationship with other Virtual Link's BAG. When BAG of VL_1 equals to 1 and another increases from 1 to 128, the delay would decreases from 6.481 to 3.593.

We could also see the value of the backlog have relationship with its Virtual Link's BAG. When the BAG of VL_1 equal to 1, and others increases from 1 to 128, the backlog would decreases from 10128 to 629 bytes.

5) Case 5: we assume, about all of the virtual links,$\mathrm{BAG} = (1\mathrm{ms})$, its $\mathrm{L}_{\max}$ values range from 64 to 1518(byte), including VL_1 . The VL_1 delay and backlog size calculated results are showed in Figure5-5.

Table5-5 shows the value range of the input and output parameters of the case5. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-5 Value range of the input and output parameters of the case5

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1	64~1518	0.043~6.379	66~9967
VL_2	1			
VL_3	1			
VL_4	1			
VL_5	1			
VL_6	1			
VL_7	1			
VL_8	1			

In Figure5-5, the x axis represents the value of L_{max} , and the y axis represents the calculated results of SDF_{pxi} and H_{maxi} .

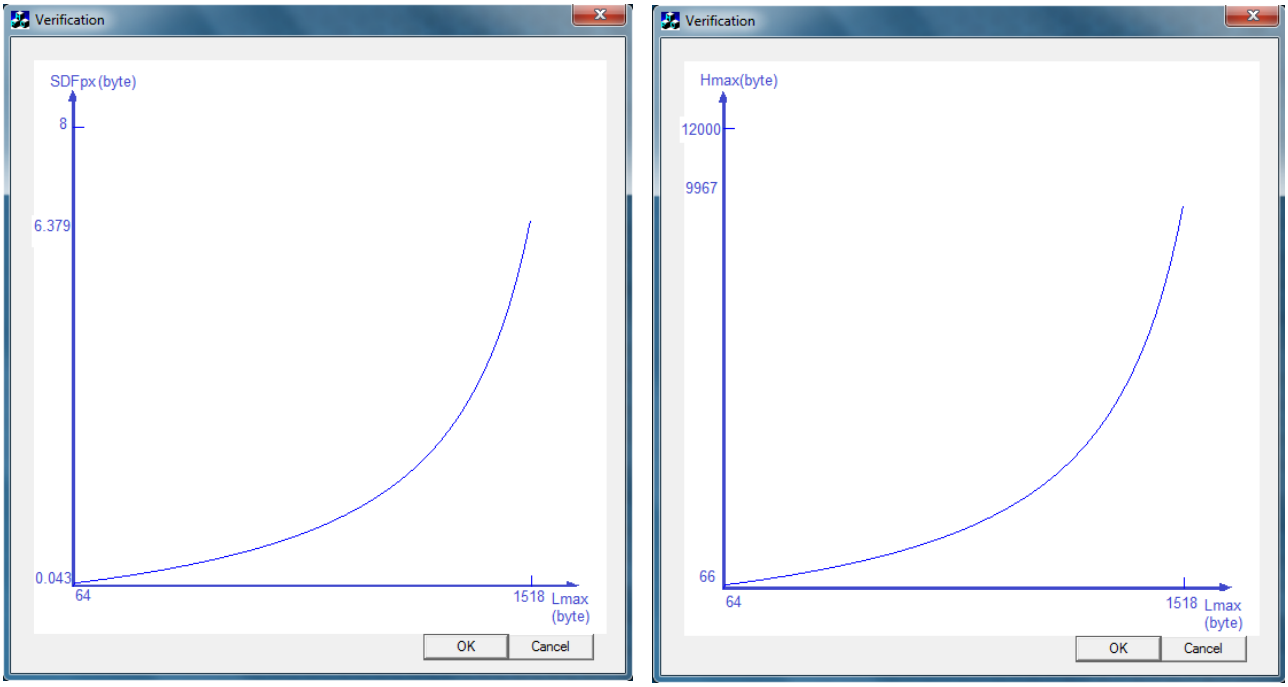

Figure 5-5 Calculated results of SDF_{pxi} and H_{maxi} in case5

In this case, we could see the value of the end-to-end latency have relationship with its Virtual Link Lmax. When the Lmax of $VL_{1\sim8}$ increases from 64 to 1518, the delay would increases from 0.043 to 6.379.

We could also see the value of the backlog have linear relationship with its Virtual Link Lmax. When the Lmax of $VL_{1\sim8}$ increases from 64 to 1518, the backlog would increases from 66 to 9967 bytes.

6) Case6: assume, about all of the virtual links, $\mathrm{L_{max}} = 1518(\mathrm{byte})$, its BAG values range from 1 to 128(ms) (2^K: $K = 1\sim7$), including VL_1 . The VL_1 delay and backlog size calculated results are showed in Figure5-6.

Table5-6 shows the value range of the input and output parameters of the case6. The value in the blue box represents the x axis, and the value in the pink box represents y axis.

Table 5-6 Value range of the input and output parameters of the case6

VL id	BAG	L_{max}	SDF_{px1}	L_{max1}
VL_1	1~128	1518	6.481~0.978	10127~1528
VL_2		1518		
VL_3		1518		
VL_4		1518		
VL_5		1518		
VL_6		1518		
VL_7		1518		
VL_8		1518		

In Figure5-6, the x axis represents the value of BAG , and the y axis represents the calculated results of SDF_{pxi} and H_{maxi} .

In this case, we could see the value of the end-to-end latency have relationship with its Virtual Link's BAG. When the BAG of $\mathrm{VL}_{1\sim8}$ increases from 1 to 128, the delay would decreases from 6.48 to 0.978.

We could also see the value of the backlog have relationship with its Virtual Link's BAG. When the BAG of $\mathrm{VL}_{1\sim8}$ increases from 1 to128, the backlog would decreases from 10127 to 1528 bytes.

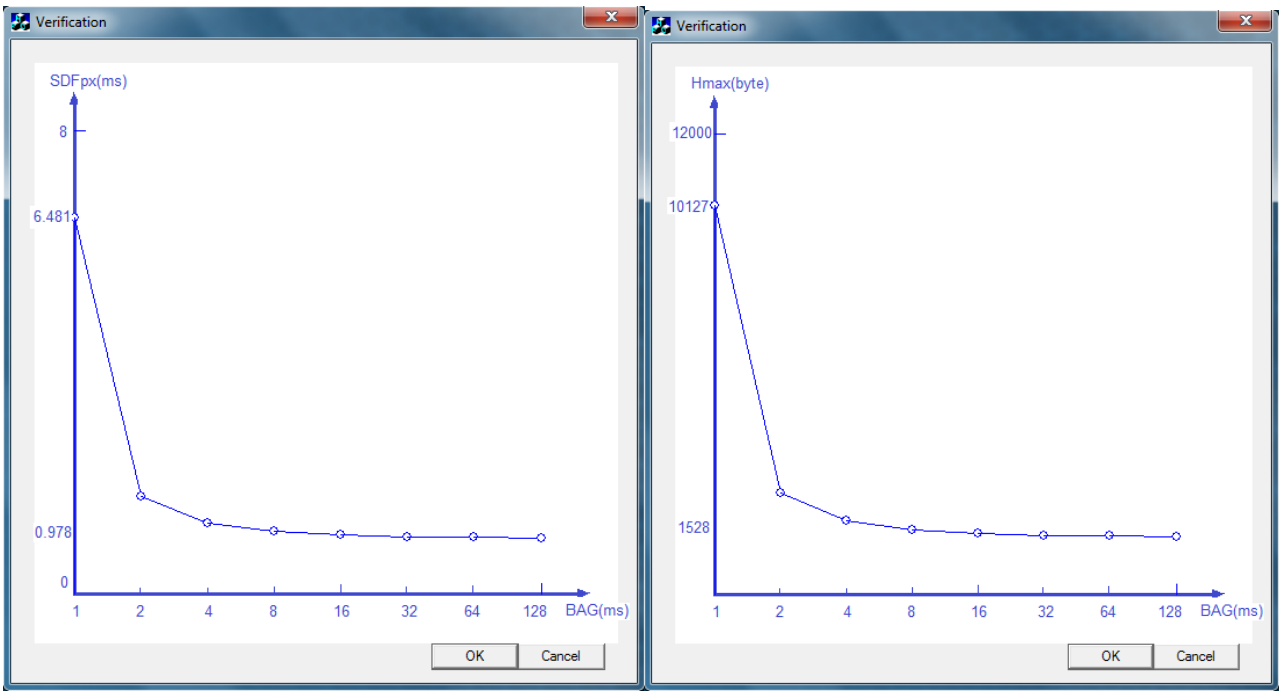

Figure 5-6 Calculated results of SDF_{pxi} and H_{maxi} in case6

5.2 Simulation test

Test plan

Before the simulation process, we plan to build the test plan, according to the demand of the network performance test and simulation acquirements. Table 5-7 shows us a test plan and final results.

Table 5-7 Simulation framework test plan

ID	BAG	Lmax	Size	PC	Port	Time	result	content
1	2	200	2000	1	2	1	problem	It executes at most 1 hour.
2	16	200	2000	1	2	5	good	It can execute at least 5 hour.
3	128	200	2000	1	2	48	good	It can execute at least 48 hour.
4	128	800	2000	1	4	48	good	It can execute at least 48 hour.
5	128	600	2000	1	8	48	good	It can execute at least 48 hour.
6	128	400	1000	2	8	48	good	It can execute at least 48 hour.
7	128	400	1000	4	20	48	good	It can execute at least 48 hour.
8	128	64	1000	4	20	48	good	It can execute at least 48 hour.
9	128	1518	2000	4	20	48	good	It can execute at least 48 hour.
10	32	1518	2000	4	20	24	good	It can execute at most 48 hour.
9	64	1518	2000	4	20	48	good	It can execute at least 48 hour.
9	128	1518	2000	4	20	48	good	It can execute at least 48 hour.

In Table5-7, *Id* means the test item number, *BAG* and *Lmax* are the VL parameters, and *size* means the VL maximum frame size. *PC* number represents how many computers there are in the network, and *Port* number represents how many ports there are in the whole network. *Time* represents how long this simulation test plan to be. The result shows that this test is good or not very good. The content shows the detail.

The simulation tests has implemented successfully. Only in one situation there are a problem happened. That is, when BAG is too small, such as 2(ms), the

simulation processes can successfully execute about one hour. The reasons are the framework architecture. It is too complex. It can cause the queue management problems, more in and less out, especially when the transmission rate is to low. According to the suggestion of my superviser, we still use this architecture in this project. Because changing the architecture means changing everything. That is not what we what.

Simulation Application Man-machine Interface

The man-machine interface of simulation applications is showed as follows.

In the Linux user space version application, simulation tests have executed continuously over 48 hours, including 3 switches and 4 computers in the network test process. The framework transmits 1000 bytes message per 64 (ms) without any transmission mistakes.

The Linux user space simulation applications windows are showed in Figure5-7. This example uses the same topology in Figure 3-6 apart from the UDP number.

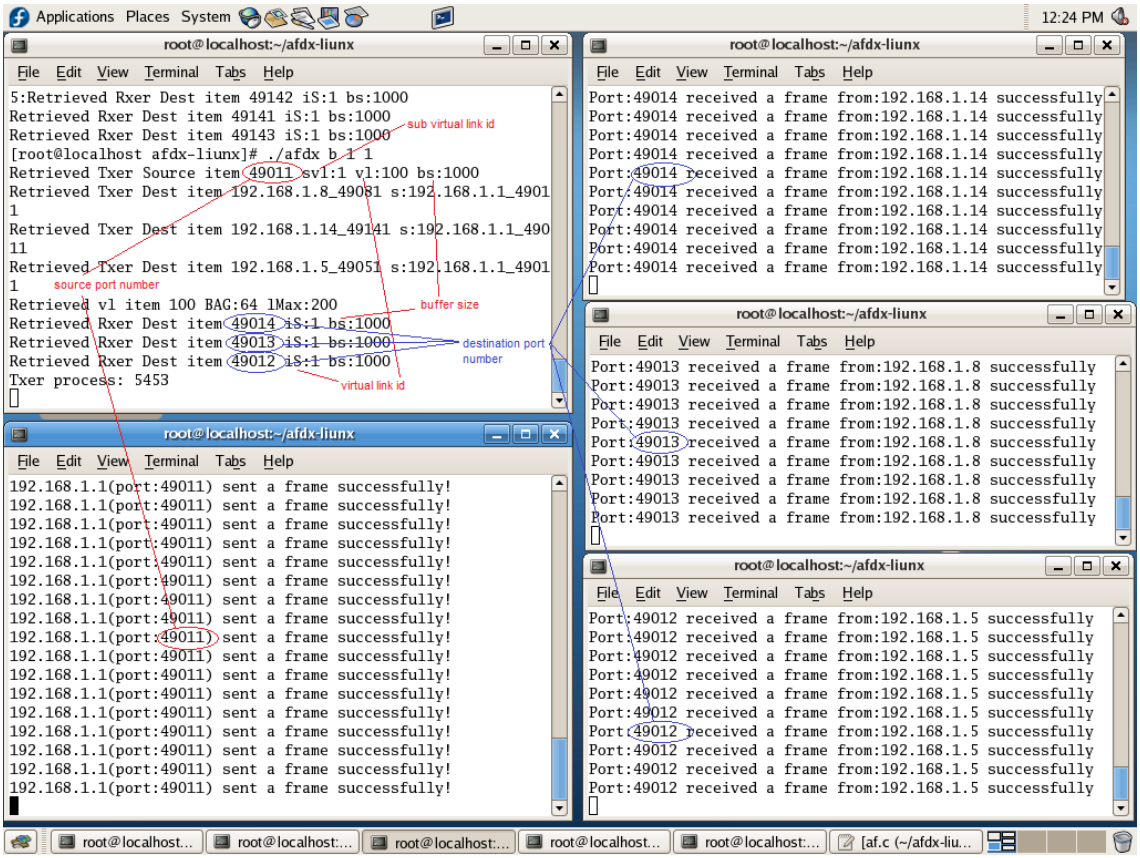

Figure 5-7 Linux user space simulation applications windows

In Figure 5-7, the left top window is the *afdx* process application interface. The indication in this window shows the *afdx* process has one item in the Txer source data base file and three items in the Txer destination data base file. That means the computer has one virtual link, this virtual link delivers data from port 49011 in this computer to the three destination computer with IP address equal to 192.168.1.5, 192.168.1.5 and 192.168.1.14, and three destination ports are 49051,49081 and 49141. The left down window is the *af* process application interface. It indicates the UDP port 49011 has been sent frames successfully to the three destination computers.

In Figure 5-7, the right three windows are the *raf* process application interface. For example, the right top window shows the port 49013 has been successfully received frames from the computer with IP address equal to 192.168.1.8.

In the RTAI version application, simulation tests have executed continuously over 48 hours, but the problem is the lab computers. We cannot find more than one computers in the lab to run the RTAI/RTnet tasks successfully, so the RTAI real time version software application only has been tested on one computer with multi AFDX ports in the whole network. The RTAI realtime version application could pass through over 48 hours running test in one computer with 1000bpm transmission rate between at least 7 AFDX send and receive ports without any transmission mistakes. The real time simulation application windows are showed in Figure5-8.

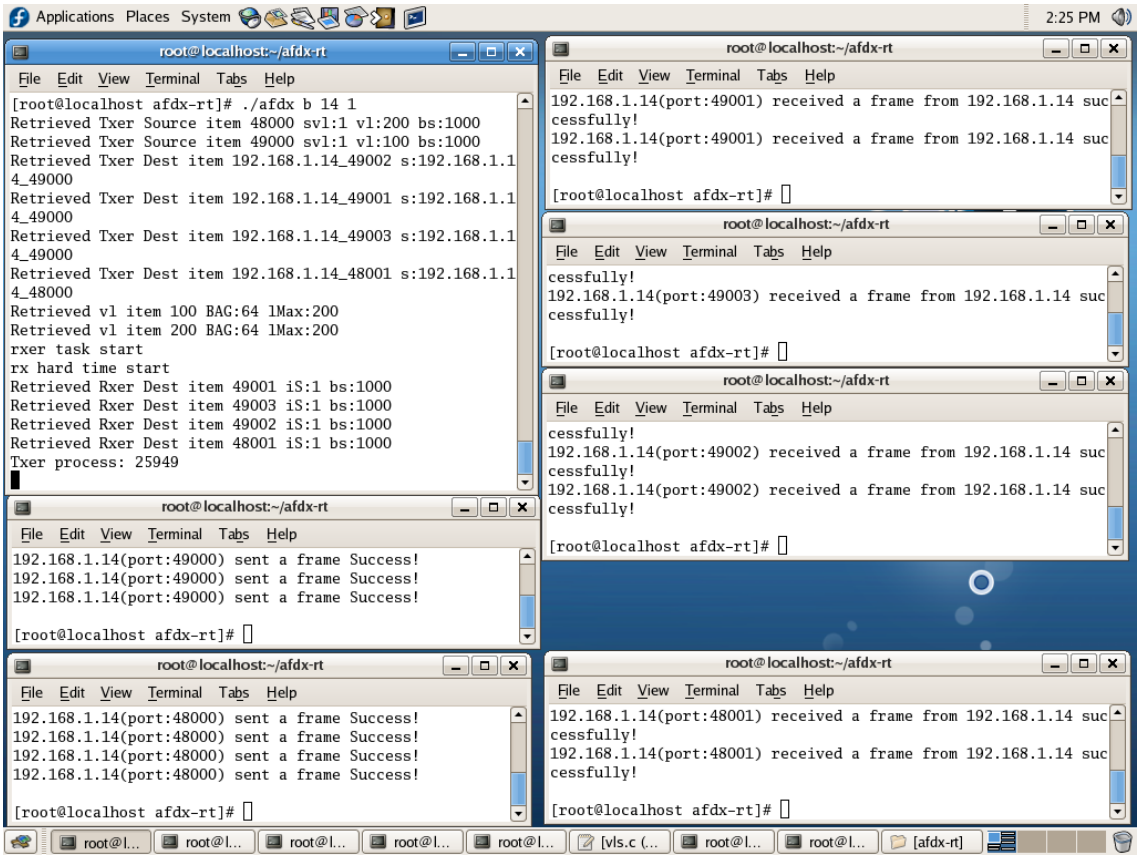

Figure 5-8 Real time simulation application windows

Simulation Verification

Aims for verifying the mathematic method validity, some real time simulation tests require to be done to get the simulation results. After comparison with the method results, mathematic method availability could be verified.

The two Linux user space and RTAI real-time version software have been successfully developed. But, due to lab computers actual situation, it is difficult to find over one computer to do the hard real time communication simulation task, even if its program logic and architecture have been verified in the Linux version. The real time version software only has been tested in one computer, and the end-to-end delay between two computers in the simulation process cannot find the way to get form the one computer test. Because the method verification needs the simulation process running on the hard real time circumstance in order to keep its results correctness. It is regretful that the verification work did not go into operation in hard real-time circumstance, and

we only did some verification in Linux user space circumstance. Each computer's situation has been listed in Table5-8.

Table 5-8 Lab computer's situation

Item	IP Address	Problems
1	192.168.1.8	It cannot run over one RTAI task at the same time.
2	192.168.1.14	Ok
3	192.168.1.5	It has version conflict between the RTAI/RTnet with Linux operation systems. Problem is that the software cannot be compiled successfully. The compilation tool do not find the the RTnet socket function's call in the kernel lib in *makefile* period.
4	192.168.1.1	
5	192.168.1.7	
6	192.168.1.3	It can go through the *rtping* test, but it cannot receive messages for applications, even the application is the typical example download from the RTnet internet resource or in the RTnet *example* directory.

The probable solutions for these computers problems and the reason why the author did not finished are listed as follows.

- To try to install the Linux and RTAI/RTnet again. Because the time is tight and the installation RTAI/RTnet process is very complex, and it maybe cost several days, especially without the installation experience. The author cannot find enough time to try it.
- To buy new computers. Because the computers in the lab have been used for a long period of time, and the configuration maybe too low to support RTAI real time applications.

Table 5-9 lists the results of the Linux user space version simulation framework test, comparing with the mathematical data.

Table 5-9 Results of the Linux simulation test comparing with the mathematical results in the same condition

BAG	Lmax	Size	PC	switch	Simulation delay	NC delay
1	1518	2000	8	8	5.0050	6.4810
2	1528	2000	8	8	3.2765	4.6087
4	1528	2000	8	8	3.4565	4.1552
8	1518	2000	8	8	2.8681	3.9345
16	1518	2000	8	8	2.6107	3.7874
32	1528	2000	8	8	2.1327	3.6404
64	1518	2000	8	8	2.2431	3.6036
128	1518	2000	8	8	1.8723	3.5931

In section5.1, there are six cases. Here we do some simulation test under the same situation with case 4 in section5.1, in order to get the end-to-end latency. The simulation results are showed in the left picture of Figure5-9. The mathematical method results are showed in the right picture of Figure5-9. We could see these two pictures have the same changing regulation. And the simulation delay is lower than the mathematical results. In conclusion, we could use the mathematical results as the upper limitation of the end-to-end delay, because it calculates the delay of the worst situation in the specific AFDX data network.

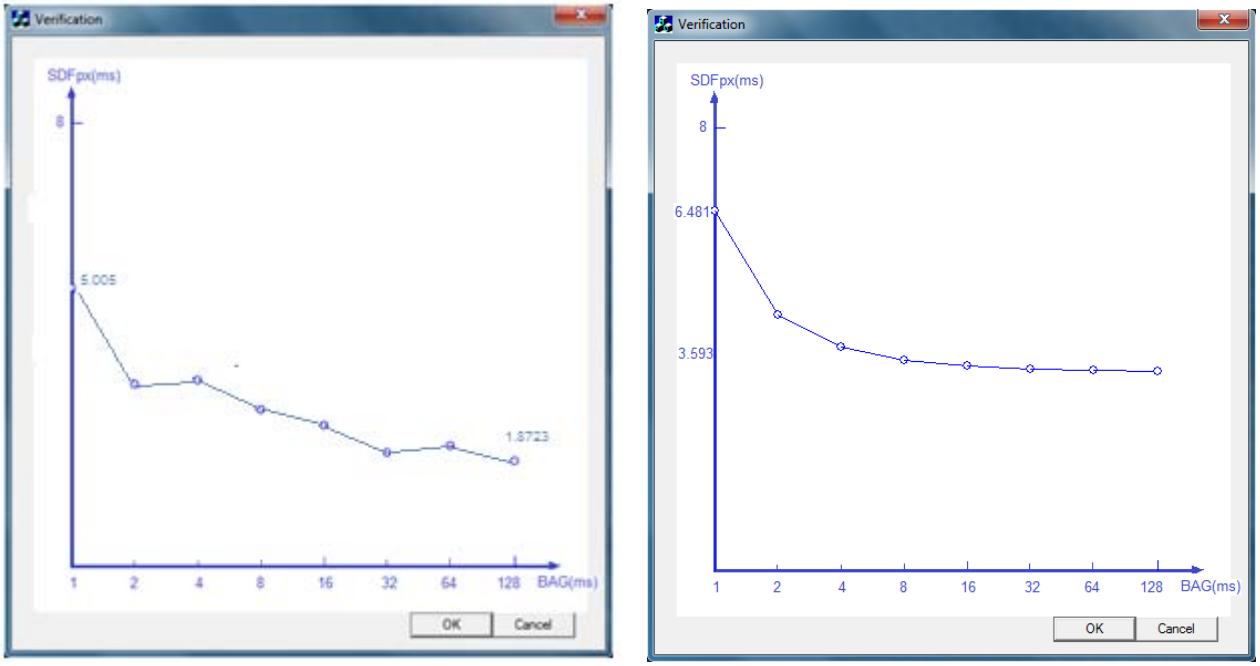

Figure 5-9 Real time simulation application windows

6 Conclusion and future work

6.1 Work Summary

In this project, the executable AFDX simulation framework is successful developed, based on the previous program final status. The previous students have built a good simulation framework before this project began, and the program is flexible, applicative and modularized.

However, what we need is two version programs, which could run on the Linux user space and RTAI real-time circumstance. The previous students did the Linux user space AFDX simulation software with so many logic errors. They did not build the hard real-time simulation software. They did nothing about the research on network performance.

All of these objectives have been achieved in this project. The major contribution by author was to successfully achieve the frameworks as the two-version software running on the RTAI/RTnet and Linux, and successfully find a new way to evaluate the network performance ---- three bounds. This method is based on the new theory ---- NC. It is very useful in the avionics design process.

The adoptive methods and resolving problems in the different developing stage have been concluded as follows.

In the early phase, it is very difficult to getting a better understanding in the complex source code made by the previous students. The thesis which previous students wrote helped a lot to understand the framework clearly, but it does no use for getting into details of the software and its sub layer logic. Apart from these, mastering and applying the technology of RTAI/net is another difficulty, because the documentary is lack and it is a complex programming technology, which need to spend so much time to hold. Under the superviser's suggestions and help, at the end, the software detail design has been totally apprehended and the lab RTAI/RTnet developing environment has been built.

During the development simulation software phase, the main research has been done on developing new features and resolving bugs. During this phase, the author modify the source code so many times, in order to understand RTAI behaviours or find out whether the logic is correct or not. Some simple examples have been worked out to test a particular feature. Because equipments and computers in the lab have been used for a long term, maybe over ten years, so the RTAI/net circumstance on these computers is not very good. Some of them are not capable of running two RTAI task applications. Some of them have the version collisions between the RTAI/RTnet and the Linux. It is proved to be too time consuming. In this phase, the two-version software has finally been achieved, and simulation test executed at last over 48 hours, and no errors happened.

In the network performance research phase, the NC theory has been completely comprehended. The methods for estimating three bounds of performance parameters have been deduced clearly and accurately. The methods may be not very perfect, but it has its own industrial application value for the network performance.

As a conclusion, the implement of the AFDX simulation platform and evaluation of network performance is a sophisticated and challenging work. The method in this thesis has an existential and application value. This project also did some simulation test and analytic comparison, and tries to verify the theories deduction and application value.

6.2 Further work

Not all the objects in the beginning have been finished because the time is tight. It is impossible to achieve them all in the end. Some suggestions for the future research are listed as follows.

- The framework of AFDX simulation seems to be a good solution for achieving the requirements. However, in the author option, it needs to be changed totally, because of the lack consideration for the most important performance of the simulation system - real time performance. The frame work

is very complex. It includes so many threads and processes. The communication between them seems to consume much time, and it will reduce the real-time performance. The simulation test verified the opinions. The simplest solution is the best!

- In this thesis, the normal standard AFDX functions were achieved. However other major features still need to be complemented in the future, such as the redundancy management, the subVL functions, etc. That would increase the AFDX simulation framework performance.

- The research for the network performance proved to be done, but it is just a beginning. The application value needs verification.

- In simulation testing phase, there is no synchronization mechanism applies in the integrated systems. So, as the results, the simulation results for testing the end-to-end latency lose its creditability. Definitely some methods which have been developed compensated this lost. The fundamental research need to be launched in employing the synchronization system.

REFERENCES

[1] RTnet-A Flexible Hard Real-time Networking Framework: https://www.rtnet.org/doc.html (accessed 01 March 2011).

[2] RTnet-Application Programming Interface: https://www.rtnet.org/doc.html (accessed 01 March 2011).

[3] RTAI3.4 User Manual rev0.3: https://www.rtai.org/index.php?module=documents&JAS_DocumentManager_op=categories&category=4&MMN_position=8:4 (accessed 01 March 2011).

[4] RTAI- A beginner's Guide: https://www.rtai.org/index.php?module=documents&JAS_DocumentManager_op=categories&category=4&MMN_position=8:4 (accessed 01 March 2011).

[5] RTAI- API Documentation magma: https://www.rtai.org/documentation/magma/html/api/ (accessed 01 March 2011).

[6] ARINC, 651-1 Design Guidance for integrated modular avionics. Aeronautical Radio Inc.,Annapolis,MD,1997.

[7] ARINC, 653 Avionics Application Software Standard Interface, Part 1 –Required Service. Aeronautical Radio Inc.,Annapolis,MD,2005.

[8] ARINC, 664 Aircraft data network part7: avionics full duplex switched Ethernet(AFDX). Aeronautical Radio Inc.,Annapolis,MD,2005.

[9] Diez Barrero,D.Distributed avionics databus simulation. Individual research project MSc , Cranfield University, Cranfield,UK,September 2009.

[10] Tommaso Falchi Delitala , Simulatioin of Switched Avionic Databus, A real-time software AFDX simulation. Individual research project MSc , School of Engineering,Cranfield University, Cranfield,UK,September 2009.

[11] Shaojun Yang ;Design and Realization of MAC IP Core on AFDX Switching Chip. Master Degree thesis , School of Communication and Information ,Chinese XIDAN University, XIAN,CHINA,January 2009.

[12] Haihua Wang ;The design and implementation of real-time Ethernet image transmission system based on RTAI/RTNET. Master degree thesis , School of Crytology, Southwest jiaotong University, XIAN,CHINA,January 2009.

[13] Jie Zhang;Study on the data transmission time and QoS based on the RTnet real-time. Master degree thesis , School of Physics, Lanzhou University, CHINA, June 2008.

[14] Li Minghui; Research on Network Modeling Based on Network Calculus. Master degree thesis , School of Computer Application, Southwest jiaotong University, CHINA, May 2007.

[15] Fan Baohua; Network performance Analysis Model Research Based on Network Calculus. Doctor degree dissertation , School of Computer Science and Technology, National University of Defense Technology, Changsha,Hunan,P.R.CHINA, October 2009.

[16] Stephen A Rago; " UNIX System V Network Programming". ADDISON-WESLEY PUBLISHING COMPANY. 1993.

[17]Jean-YVES LE BOUDEC PATRICK THIRAN; NETWORK CALCULUS, A Theory of Deterministic Queuing Systems for the

Internet.Online Version of the Book Springer Verlag-LNCS 2050.Version May 10.2004.

[18]Jean-luc Scharbarg,Christian Fraboul; "Simulation for end-to-end delays distribution on a switched Ethernet". IRIT-ENSEEIHT， Toulouse University， 2, rue Camichel 31000 Toulouse-France.1-4244-0826-1/07, ©2007 IEEE.

[19]Marc Boyer,Christian Fraboul; "Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus". IRIT/ENSEEIHT-University of Toulouse，France978-1-4244-2350-7/08,©2008 IEEE .

[20] Jean-Luc Scharbarg, Frédéric Ridouard, and Christian Fraboul; "Probabilistic Analysis of End-To-End Delays on an AFDX Avionic Network". 1551-3203, © 2009 IEEE.

[21] Hussein Charara, Jean-Luc Scharbarg, Jerome Ermont, Christian Fraboul ; "Methods for bounding end-to-end delays on an AFDX network". Proceedings of the 18th Euromicro Conference on Real-Time Systems(ECRTS'06),0-7695-2619-5 /06, © 2006 IEEE.

[22] Xiaoting Li, Jean-Luc Scharbarg, Christian Fraboul; "Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis". Universite de Toulouse - IRIT/ENSEEIHT/INPT - Toulouse, France 978-1-4244-6850-8/10/. ©2010 IEEE.

[23] IEEE Computing Society; 802.3 Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Speci_cations. IEEE, New York, December 2008.

[24] Hong Guo, A Pu Zhang;"Johnson 算法的极大极小代数证明", 1998-01-15;FuJian province 福建省自然科学基金资助项目, 厦门大学自动化系, 厦门, 中国.361005 ???E

[25] 须文波,张星烨，欧爱辉."基于 RTAI-Linux 的实时操作系统的分析与研究", 现代计算机（总第 163 期）;江南大学信息工程学院，无锡，江南大学通信与控制工程学院，无锡，中国. 214036 .

[26] FAN Bao-hua, Dou Qing,Zhang He-Ying. “A Matrix Interpretation of Network Calculus”.School Of Computer Science. National University of Defense echnology,Changsha,410073,Chian Academic Journal Electronic Publishing House.2009-10-25. TP323.10.3724/SP.J.1016.2009.02411.

[27] IEEE Computing Society. 802.3 Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Speci_cations. IEEE, New York, December 2008.

[28] Ian Moir and Allan Seabridge. ”Civil Avionics Systems”. ISBN:978-0-4700-2929-9, 2003

APPENDICES

Appendix A Network Calculus Theory

A.1 Basic Min-plus Calculus

Infimum and minimum

Let S be a nonempty subset of R. S is bounded from below if there is a number Msuch that $s \geq M$ for all $s \in S$. The completeness axiom states that every nonempty subset S of R that is bounded from below has a greatest lower bound. We will call it *infimum* of S, and denote it by $infS$. we will often use the notation $\wedge$ to denote *infimum.*

A catalog of wide-sense increasing functions

A function f is wide-sense increasing, if and only if *f(s)* $\leq$ *f(t)* for all *s* $\leq$ *t*.

DEFINITION 3.1.1 (PEAK RATE FUNCTIONS $\lambda_R(t)$).

$$\lambda_R(t) = \begin{cases} Rt & if\ t > 0 \\ 0 & otherwise \end{cases}$$

For some $R \geq 0$ *(the 'rate')*

DEFINITION 3.1.2 (BURST DELAY FUNCTIONS δ_T).

$$\delta_T(t) = \begin{cases} +\infty & if\ t > T \\ 0 & otherwise \end{cases}$$

For some $T \geq 0$ *(the 'delay').*

DEFINITION 3.1.3 (RATE-LATENCY FUNCTIONS $\beta_{R,T}$).

$$\beta_{R,T}(t) = R[t-T]^+ = \begin{cases} R[t-T] & if\ t > T \\ 0 & otherwise \end{cases}$$

For some $R \geq 0$ *(the 'rate') and* $T \geq 0$ *(the 'delay').*

DEFINITION 3.1.4 ($\gamma_{r,b}$).

$$\gamma_{r,b}(t) = \begin{cases} rt + b & if\ t > 0 \\ 0 & otherwise \end{cases}$$

For some $r \geq 0$ *(the 'rate') and* $b \geq 0$ *(the 'burst').*"

DEFINITION 3.1.5 "(STEP FUNCTIONS v_T).

$$v_T(t) = 1_{\{t<T\}} = \begin{cases} 1 & if\ t > T \\ 0 & otherwise \end{cases}$$

For some $T \geq 0$.

DEFINITION 3.1.6 (STAIRCASE FUNCTIONS $u_{T,r}$).

$$u_{T,r}(t) = \begin{cases} \left\lceil \frac{t+r}{T} \right\rceil & if\ t > 0 \\ 0 & otherwise \end{cases}$$

For some $T \geq 0$ *(the 'interval') and* $0 \leq r \leq T$ *(the 'tolerance').*

These functions are also showed in the figure B-1.

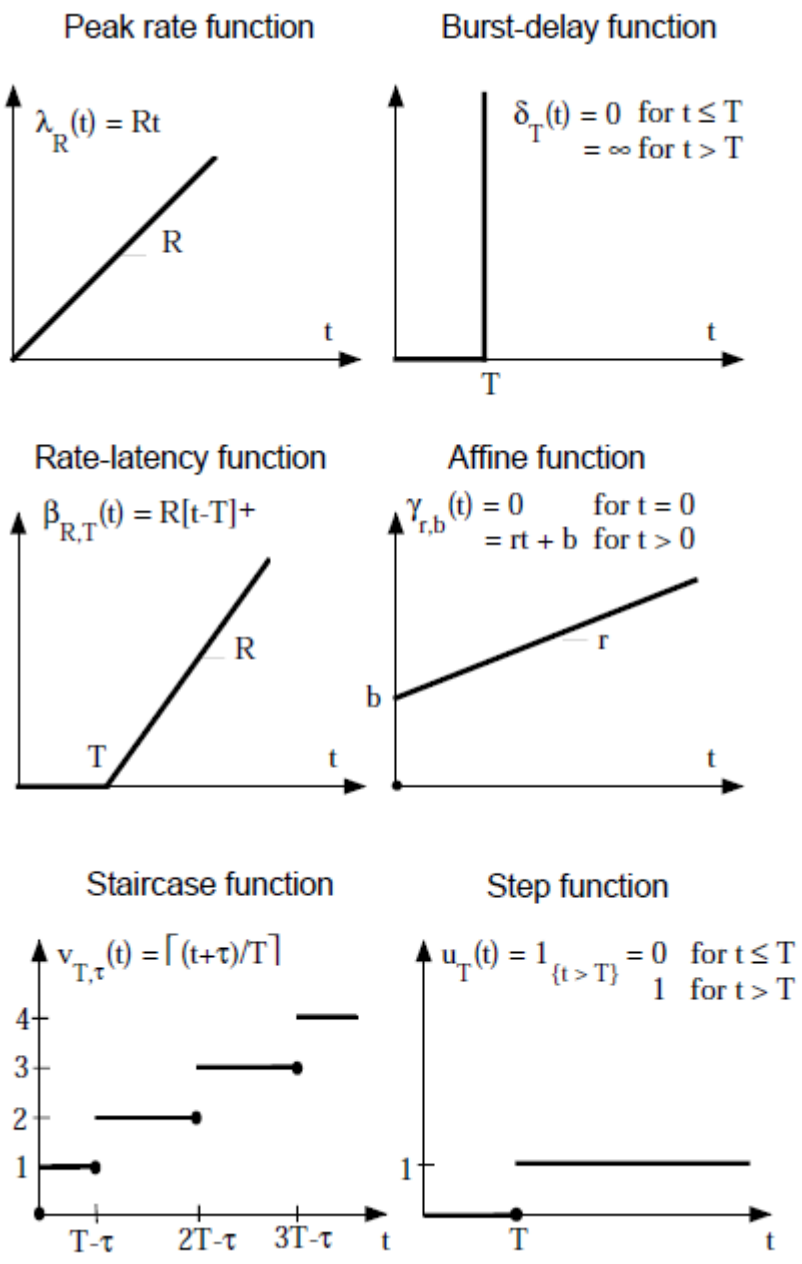

Figure B-1 Catalog of wide-sense increasing functions

Concave, convex and star-shaped functions

DEFINITION 3.1.7 (CONVEXITY IN $\mathbb{R}^n$). Let $0 \le u \le 1$ be any real such that.

- Subset $\mathbb{S} \subseteq \mathbb{R}^n$ is convex if and only $ux + (1-u)y \in \mathbb{S}$ for all $x, y \in \mathbb{S}$.
- Function f from a subset $\mathbb{D} \subseteq \mathbb{R}^n$ to $\mathbb{R}$ is convex if and only if $f(ux + (1-u)y) \leq uf(x) + (1-u)f(y)$ for all $x, y \subseteq \mathbb{D}$.
- Function f from a subset $\mathbb{D} \subseteq \mathbb{R}^n$to is concave if and only if $-f$ is convex.

DEFINITION 3.1.8 (STAR-SHAPED FUNCTION). Function $f \in \mathcal{F}$ is star-shaped if and only if $f(t)/t$ is wide-sense decreasing for all $t > 0$.

Star-shaped enjoy the following property:

THEOREM 3.1.1 (MINIMUM OF STAR-SHAPED FUNCTIONS). Let f, g be two star-shaped functions. Then $h = f \wedge g$ is also star-shaped.

THEOREM 3.1.2. Concave functions are star-shaped.

Min-plus connvolution

DEFINITION 3.1.9 (MIN-PLUS CONVOLUTION). Let f and g be two functions or sequences of $\mathcal{F}$. The min-plus convolution of f and g is the function

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}$$

(If $t < 0$, $(f \otimes g)(t) = 0$).

An example of convolution result is shown in the figureB-2

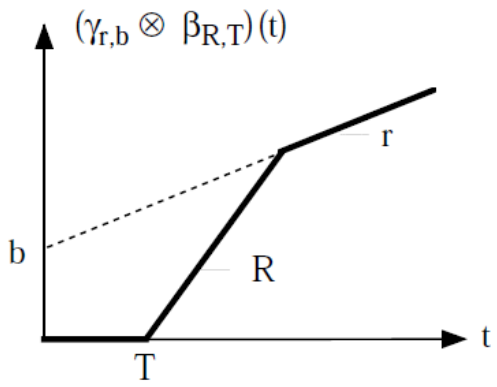

Figure B-2 Function $\gamma_{r,b} \otimes \beta_{R,T}$, when $0 < r < R$

Some properties for ⊗are listed.

THEOREM 3.1.3 (GENERAL PROPERTIES OF⊗). Let $f, g, h \in \mathcal{F}$.

- Rule 1 (Closure of ⊗) $(f \otimes g)(t) \in \mathcal{F}$.
- Rule 2 (Associativity of ⊗) $(f \otimes g) \otimes h = f \otimes (g \otimes h)$.
- Rule 3 (The zero element for ∧ is absorbing for ⊗) The zero element for ∧ belonging to $\mathcal{F}$ is the function ε, defined as $\varepsilon(t) = +\infty$ for all $t \geq 0$ and $\varepsilon(t) = 0$ for all $t < 0$. One has $f \otimes \varepsilon = \varepsilon$.
- Rule 4 (Existence of a neutral element for ⊗) The neutral element is δ_0 , as $f \otimes \delta_0 = f$.
- Rule 5 (Commutativity of ⊗) $f \otimes g = g \otimes f$.
- Rule 6 (Distributivity of ⊗ with respect to ∧) $(f \wedge g) \otimes h = (f \otimes h) \wedge (g \otimes h)$.
- Rule 7 (Addition of a constant) For any ∈ $\mathbb{R}^+$ $(f + K) \otimes g = (f \otimes g) + K$.

THEOREM 3.1.4 (PROPERTIES OF ⊗FOR CONCAVE/CONVEX FUNCTIONS). Let $f, g, h \in \mathcal{F}$.

- Rule 8 (Functions passing through the origin) If $f(0) = g(0) = 0$ then $f \otimes g \leq f \wedge g$. Moreover, if f and g are star-shaped, then $f \otimes g = f \wedge g$.

- Rule 9 (Convex functions) If f and g are convex then $f \otimes g$ is convex. In particular if f, g are convex and piecewise linear, $f \otimes g$ is obtained by putting end-to-end the different linear pieces of f and g, sorted by increasing slopes.

Sub-additive functions

DEFINITION 3.1.10 (SUB-ADDITIVE FUNCTION). Let f be a function or a sequence of $\mathcal{F}$. Then f is sub-additive if and only if $f(t+s) = f(t) + f(s)$ for all $s, t \geq 0$.

THEOREM 3.1.5 (PROPERTIES OF SUB-ADDITIVE FUNCTIONS). Let $f, g \in \mathcal{F}$.

- (Star-shaped functions passing through the origin) If f is star-shaped with $(0) = 0$, then f is sub-additive.

- (Sum of sub-additive functions) If f and g are sub-additive, so is $(f + g)$.

- (Min-plus convolution of sub-additive functions) If f and g are sub-additive, so is $(f \otimes g)$.

Sub-additive closure

DEFINITION 3.1.11 (SUB-ADDITIVE CLOSURE). Let f be a function or a sequence of $\mathcal{F}$. Denote $f^{(n)}$ the function obtained by repeating $(n-1)$ convolutions of f with itself. By convention, $f^{(n)} = \delta_0$, so that $f^{(1)} = f$, $f^{(2)} = f \otimes f$ etc. Then the sub-additive closure of f, denoted by $\bar{f}$, is defined by

$$\bar{f} = \delta_0 \wedge f \wedge (f \otimes f) \wedge (f \otimes f \otimes f) \wedge \ldots = \mathrm{Inf}_{n \geq 0} f^{(n)}.$$

COROLLARY 3.1.1 (SUB-ADDITIVE CLOSURE OF A SUB-ADDITIVE FUNCTION). Let , $g \in \mathcal{F}$. Then the three following statements are equivalent: $(i) f(0) = 0$ and f is sub-additive $(ii) f \otimes f = f$ $(iii) \bar{f} = f$.

THEOREM 3.1.6 (OTHER PROPERTIES OF SUB-ADDITIVE CLOSURE). Let $f, g \in \mathcal{F}$

• (Isotonicity) If $f \leq g$ then $\bar{f} \leq \bar{g}$.

• (Sub-additive closure of a minimum) $\overline{f \wedge g} = \bar{f} \wedge \bar{g}$.

• (Sub-additive closure of a convolution) $\overline{f \otimes g} = \bar{f} \otimes \bar{g}$ .If $f(0) = g(0) = 0$ then $\overline{f \otimes g} = \bar{f} \otimes \bar{g}$.

Min-plus deconvolution

The min-plus deconvolution is reverse defination of the min-plus convolution. Notation ∨ represents max: $a \vee b = max\{a, b\}$.

DEFINITION 3.1.12 (MIN-PLUS DECONVOLUTION). Let f and g be two functions or sequences of $\mathcal{F}$. The min-plus deconvolution of f by g is the function

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

An example is showed in the figureB-3.

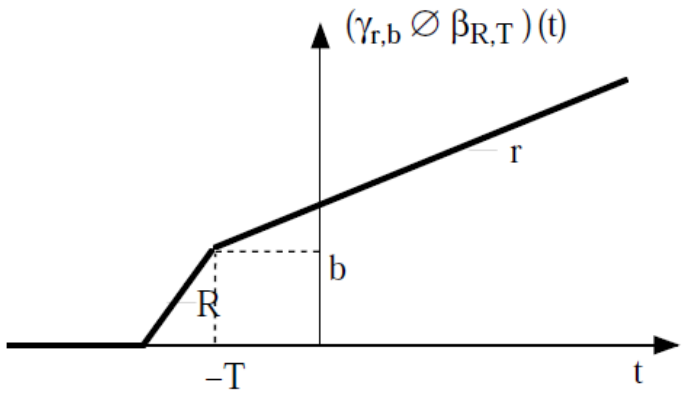

Figure B-3 Function $\gamma_{r,b} \oslash \beta_{R,T}$ when $0 < r < R$

Representation of min-plus deconvolution by time inversion

LEMMA 3.1.1. Let $f \in \mathcal{F}$ be such that $\lim_{t\to+\infty} f(t) = +\infty$. For any $g \in \hat{\mathcal{G}}$, $g \oslash f$ is also in $\hat{\mathcal{G}}$ and $(g \oslash f)(+\infty) = g(+\infty)$).

DEFINITION 3.1.13 (TIME INVERSION). For a fixed $T \in [0, +\infty]$,the inversion operator Φ_T is defined on $\hat{\mathcal{G}}$ by:

$$\Phi(f)(g) = g(+\infty) - g(T - t)$$

THEOREM 3.1.7 (REPRESENTATION OF DECONVOLUTION BY TIME INVERSION). Let $\in \hat{\mathcal{G}}$, and let T be such that $g(T) = g(+\infty)$. Let $f \in \mathcal{F}$ be such that $\lim_{t\to+\infty} f(t) = +\infty$. Then

$$\mathrm{g} \oslash \mathrm{f} = \Phi_{\mathrm{T}}(\Phi_T(g) \otimes f)$$

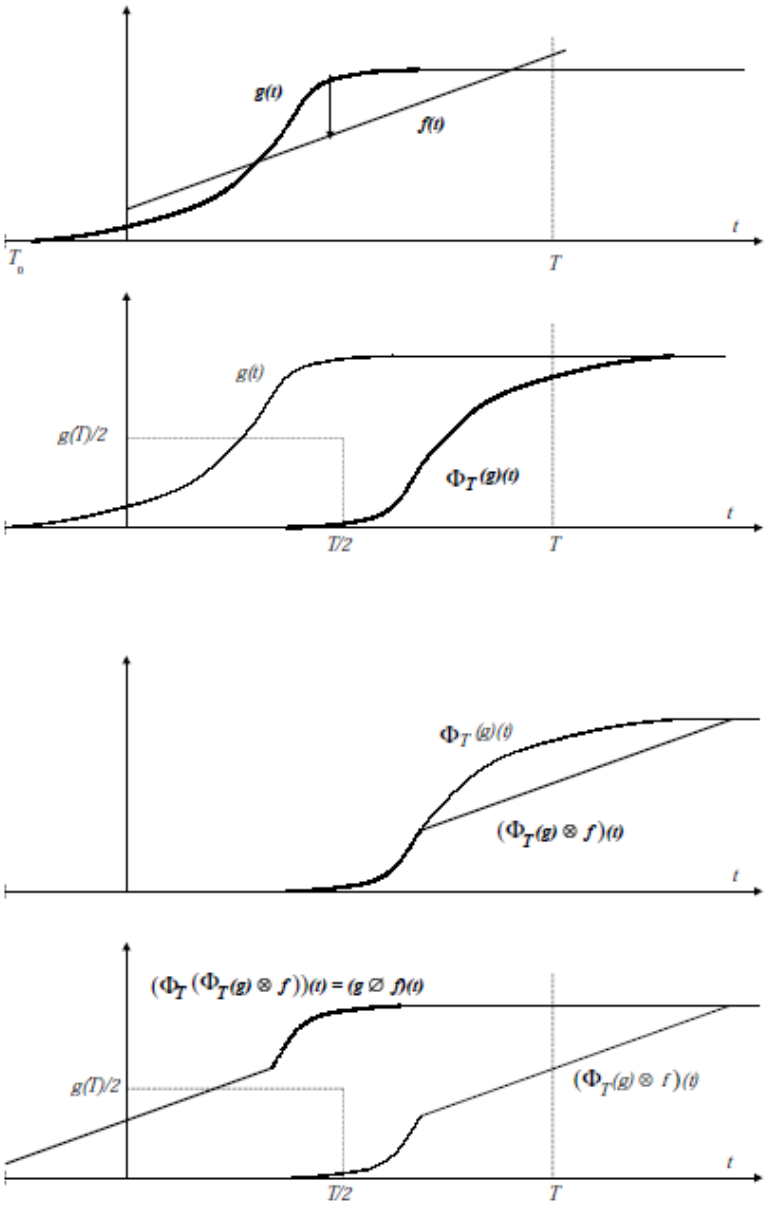

Figure B-4 Examples for $g \oslash f = \Phi_T(\Phi_T(g) \otimes f)$

Vertical and horizontal deviations

The maximum vertical and horizontal deviation between two curves f and g in $\mathcal{F}$ is illustrated in the figurateB-5. The mathematical definition is showed as follows.

DEFINITION 3.1.14 (VERTICAL AND HORIZONTAL DEVIATIONS). Let f and g be two functions or sequences of $\mathcal{F}$. The vertical deviation $v(f,g)$ and horizontal deviation $h(f,g)$ are defined as

$$v(f,g) = \sup_{t\geq 0}\{f(t) - g(t)\}$$

$$h(f,g) = \sup_{t\geq 0}\{\inf\{d \geq 0 \; such \; that \; f(t) \leq g(t+d)\}\}$$

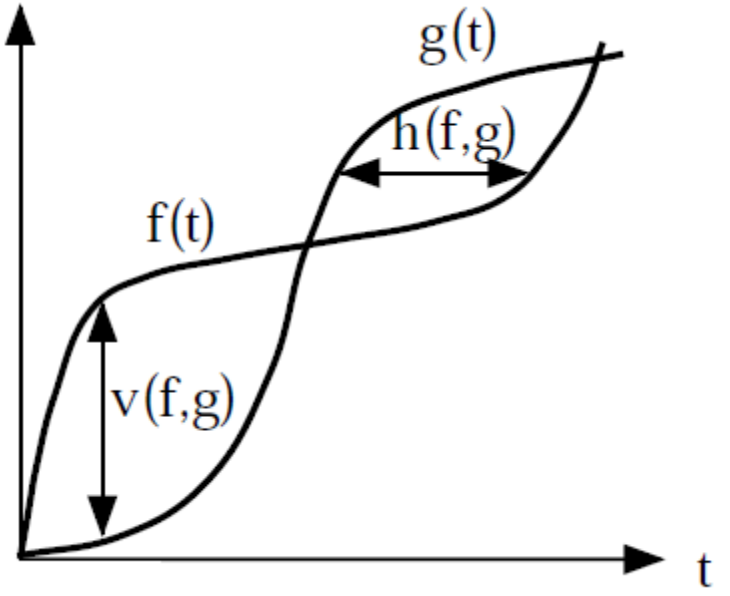

Figure B-5 Horizontal and vertical deviations between functions f and g.

Note that $v(f,g)$ can be recast as

$$v(f,g) = (f \oslash g)(0)$$

So that $h(f,g)$ can be expressed as

$$(f \oslash g)(t) = \sup_{t\geq 0}\{g^{-1}(f(t)) - t\} = (g^{-1}(f) \oslash \lambda_1)(0)$$

Appendix B Main software functions instruction

B.1 Afdx.c and afdx.h

```
// initialization the framework
bool initializeFramework(char *c_confFileName,unsigned int u_es,unsigned int u_version);
// load the database files records
bool loadDBFRM(unsigned int i_es, unsigned int u_version);
// frees the database handler.
void freeDB(void);
// malloc and set the records//variables.
void setRecords(unsigned int u_es, unsigned int u_version);
// add the message m into the queue q.
void addToRecord(Queue q_q, Message m_m);
// sends the message m through the port p
ResultCode sendMessage(AFDXport p_p, Message m_m);
// retrieves the last senting messages form the queue.
ResultCode retrieveLastSentMessages(AFDXport p, Queue **q);
ResultCode receiveMessage(AFDXport p, Message *m);
// free all the allocated memory.
void destroyFramework();
```

B.2 Txer.c and txer.h

```
//the txer thread main function.
void runTxer(char *c_confFileName,unsigned int u_es,unsigned int u_version );
// initialization the txer thread environment.
void initializeTxer(char *c_confFileName,unsigned int u_es,unsigned int i_version);
//wait for the rxer thread end.
void waitForRxer(void);
// free all environment parameters
void freeEnvironments(void);
// load the conf file data into the txer thread
void loadConfigurationTxer(char *c_confFileName);
```

```
// launches the Rxer process.
void launchRxerTxer(int i_es, int i_version);
// allocate environment parameters memory in txer thread.
void setEnvironmentTxer(void);
// launche all the rrs threads and vls threads.
void launchRRSandVLSTxer(void);
// allocate environment memory in the Sequencer threads.
void setSequencersEnv(void);
// launche the Sequencer threads .
void launchSequencers(void);
```

B.3 Rxer.c and rxer.h

```
//the rxer thread main function
void runRxer(int i_es, int i_version);
//the the main rxer loop function
bool server();
// allocates environment parameters in the Rxer process
void setEnvironmentRxer(void);
// free all environment parameters memory in Rxer variables.
void freeEnvironmentRxer(void);
// launch dispatcher threads.
void launchDispatchers(void);
// waits for dispatcher threads end.
void waitForDispatchers(void);
// closes assembler message queue ports.
void closeAssemblerMQ(void);
// closes assemblers threads.
void launchAssemblers(void);
```

B.4 Dmb.c and dmb.h

```
// This function keeps an array of Source entries to the table filename of
//the general database.
bool writeSourceDB(char *filename,keysource *ks,struct g_source*source,
unsigned int n_source );
// This function retrieves an array of Source entries to the table filename
```

```
//of the general database.
bool readSourceDB(char *filename, keysource *ks, struct g_source *source, unsigned int *n_source);
// This function keeps an array of Dest entries to the table filename of the
//general database.
bool writeDestDB(char *filename, keydest *kd, struct g_dest *dest, unsigned int n_dest);
// This function retrieves an array of Dest entries to the table filename of
//the general database.
bool readDestDB(char *filename, keydest *kd, struct g_dest *dest, unsigned int *n_dest);
// This function keeps an array of VirtualLink entries to the table filename
//of the general database.
bool writeVLDB(char *filename, keyvl *kvl, struct g_virtualLink *vl, unsigned int n_virtualLink);
// This function retrieves an array of VirtualLink entries to the table
//filename of the general database.
bool readVLDB(char *filename, keyvl *kvl, struct g_virtualLink *vl, unsigned int *n_vl);
// This function keeps an array of Source entries to the table filename of a
//Txer ES database.
bool writeTxerSourceDB(char *filename, keytxersource *ks, struct txer_source *source, unsigned int n_source);
// This function keeps an array of Dest entries to the table filename of a
//Txer ES database.bool writeTxerDestDB(char *filename, keytxerdest *kd,
struct txer_dest *dest, unsigned int n_dest);
// This function keeps an array of Dest entries to the table filename of a
//Rxer ES database.
writeRxerDestDB(char *filename, keyrxerdest *kd, struct rxer_dest *dest, unsigned int n_dest);
// This function creates the local database to keep the array of source
//entries of a Txer ES.
bool createTxerSourceDB(keysource *ks,struct g_source *sources, unsigned int n_sources, in_addr_t source_ip, unsigned int i_version);
// creates the txer database for vl.
bool createTxerVLDB(keyvl *k_kvl, struct g_virtualLink *s_vls,unsigned int
```

```
u_n_vls, keysource *k_ks, struct g_source *s_sources, unsigned int
u_n_sources, in_addr_t i_source_ip, unsigned int u_version);
// retrieves records in source database used in txer database.
bool readTxerSourceDB(char *c_filename, keytxersource *k_ks, struct
txer_source *s_source, unsigned int *u_n_source);
```

B.5 Util.c and util.h

```
// Initialize a new Queue.
bool initializeQueue(Queue *q_q, size_t s_max_size);
// destroy a queue.
void destroyQueue(Queue *q_q);
// adds a new data into the Queue.
bool addQueue(Queue *q_q, Queueable q_data, size_t s_data_size);
// read the record, but does not remove it, from the queue.
bool peekQueue(Queue q_q, Queueable data, size_t *s_data_size);
// remove the record in queue.
bool removeQueue(Queue *q_q, Queueable q_data, size_t *s_data_size);
// see the record  in the Queue ,but not remove it.
bool viewQueue(Queue q_q, unsigned int u_index, Queueable q_data, size_t
*s_data_size);
// serialize a Queue.
RawData serializeQueue(Queue q_q);
// deserializequeue a Queue.
bool deserializeQueue(Queue *q, RawData rd);
// initiliaze data sets buffer .
bool initializeDS(struct DataSet *d_ds, unsigned char *u_data, size_t
s_size);
```