Automated Interoperability Tests for PNNI: Routing Part

November 30, 1998

Table of contents

Introduction

This document describes the work that was done to accomplish the task of developing and automating interoperability tests for The ATM Forum's network routing specification, Private Network-Network Interface (PNNI) version 1.0. It describes what interoperability means in this context and how it is to be implemented in a test environment. It also describes in detail the programs, filters, and capturing mechanisms for recording and processing the observed information. It explains the actual tests that were developed and the results of using these tests on a sampling of ATM switches implementing the PNNI protocol.

1 Interoperability

Interoperability testing is done to measure to what extent and under which conditions two Systems Under Test (SUTs) that are connected to each other will inter-operate in a real operating environment and produce the expected behavior. Mostly the testing involves a certain protocol that is part of the SUT, which is then called the Implementation Under Test (IUT).

For the implemented test suite, the IUT is PNNI V1.0 Routing as specified in [ATM Forum, 1996] including the Errata [ATM Forum, 1997]. The test suite that is the basis for the developed program is described in "Interoperability Tests for PNNI" [ATM Forum, 1998] that was presented at the October 1998 meeting of the ATM Forum.

The generic test setup of the interoperability test of two ATM switches or implementations running PNNI, as given in [ATM Forum, 1994], is shown in Figure 1.

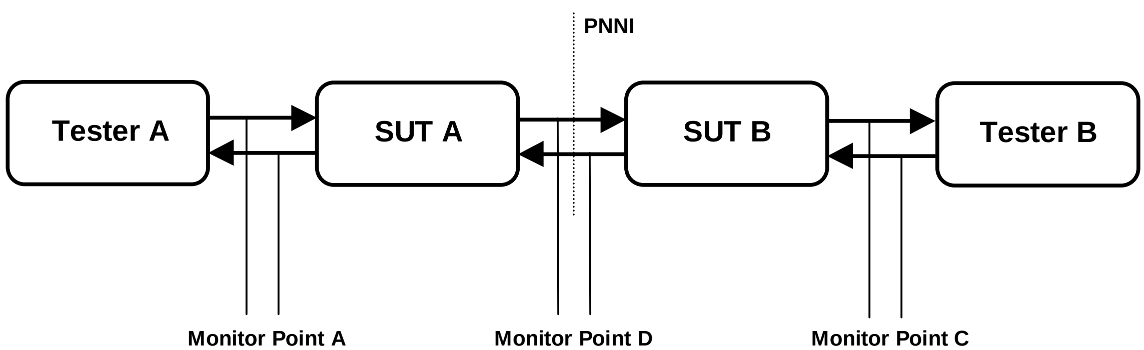

Figure 1: Generic PNNI Test Configuration

In this configuration, the interoperability is tested at monitor point D. Tester A and B are added to cover the complete PNNI behavior by generating extra (outside) events that result in a certain behavior between SUT A and B. Monitor points A and C are necessary for observing the generated events, so that this information can be used in the tests at point D.

For interoperability of PNNI Routing the complete generic configuration is not necessary. When analyzing the Routing protocols this becomes clear. The Hello and Database Synchronization (DBS) protocols only run between two nodes, so a configuration of two nodes is sufficient. The Flooding protocol runs over all the nodes in a peergroup, but only three nodes are necessary to test the complete behavior. The Peer Group Leader Election (PGLE) also runs over a complete peergroup, but a configuration of two nodes combined with a tester that fakes another node, is enough to test the complete behavior of this protocol. Therefor, two configurations are used in the interoperability tests, one mainly for testing the Hello and DBS protocols (Figure 2) and one that is mainly for testing the PGLE and Flooding protocols (Figure 3).

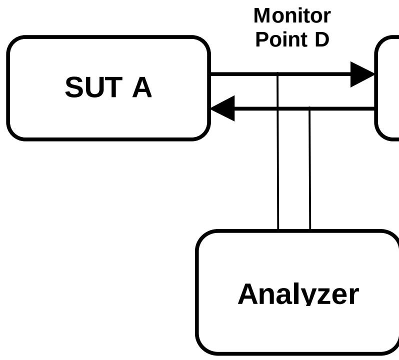

Figure 2: Configuration of analyzer observing packet exchange between switch A and B at monitor point D.

In the tests the analyzer has a passive role, in that it only observes the packets exchanged between SUT A and B and depending on the configuration, also between SUT A and Tester A. The analyzer has to decide if the two SUTs inter-operate in a correct way by means of applying a series of test cases. Each test case checks that a specific action or a sequence of actions occurs during the exchange of packets.

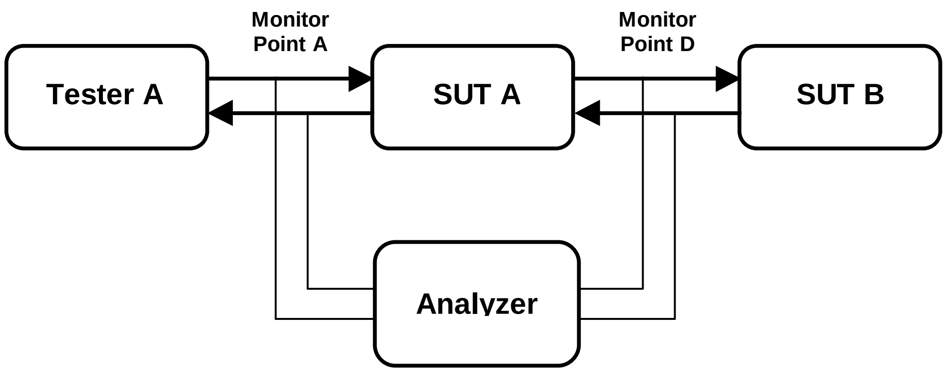

Figure 3: Configuration of analyzer observing packet exchange between switch A and B (monitor point D) and switch A and tester A (monitor point A).

A general assumption is that conformance testing of each SUT has already been done before starting the interoperability testing. Interoperability testing is being used to determine and help rectify any incompatibilities in conforming systems in real working environments.

2 General description

The analyzer that is used for running the Interoperability tests is the interWATCH 95000 ATM/LAN/WAN analyzer, a product of GN Nettest. The current analyzer has two ATM cards each capable of capturing ATM packets from one bi-directional link. The analyzer provides the ability to capture packets that are transmitted over a certain VPI/VCI pair and is also able to filter these packets. In the used setup, only PNNI Routing packets are captured and decoded. The software versions that are used are listed below:

Atak : v2.100
PNNI : v2.000
Application manager : v2.1.1

The standard interfacing language for the analyzer is TCL. Additional TCL commands are provided to operate the analyzer.

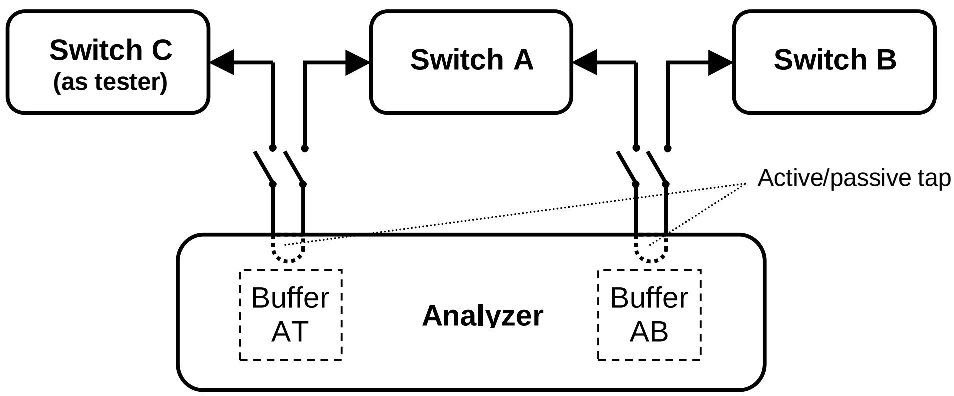

Figure 4: Test setup for interoperability testing.

Figure 4 shows the setup of the switches as used for the implemented test suite. It is different from the configurations in Figure 2 and Figure 3 in that Tester A is replaced by a switch. Since the current software of the interWATCH analyzer does not support the PNNI Routing protocols for simulation purposes, it can not be used to actively communicate with another switch. Although the tester is replaced by a switch most of the test cases from the test suite do not change. Some requirements regarding the switch have to be made though:

- PNNI Routing must be implemented correctly. This forms a major problem, since the implemented test suite is supposed to (partly) check this. A requirement might be that the switch has passed conformance testing, so that at least the switch conforms to the protocol and that the implementation does not contain any major errors.
- The Peer Group must be configurable. This is necessary for the tests of the Hello protocol.
- The Leadership Priority must be configurable. This is necessary for the tests of the PGLE protocol.
- At startup a PTSE must be present in the topology database. When PNNI has been implemented correctly in the switch this should happen automatically, since at startup the switch would have a PTSE containing the Nodal Information Group that belongs to the node in question.

The above requirements also apply to SUT A and B, otherwise not all the tests can be run. Figure 4 shows that the links between the switches can be connected and disconnected to obtain the desired configuration for running a group of tests.

Figure 5 shows the basic structure of the implemented test suite. The main program links the different parts of the program together. After capturing a buffer or a number of buffers, a filter is is run to reformat these buffer so that the actual tests have a less difficult task in retrieving the information from the packets that were exchanged. After filtering, the actual interoperability tests are run. Depending on the configuration of the switches, section A, B or C is run. Chapter 6 will explain under which circumstances each section must be run. Each section calls a number of test cases for each of the four protocols that are tested.

More information about each of the parts of the program is given in the following chapters.

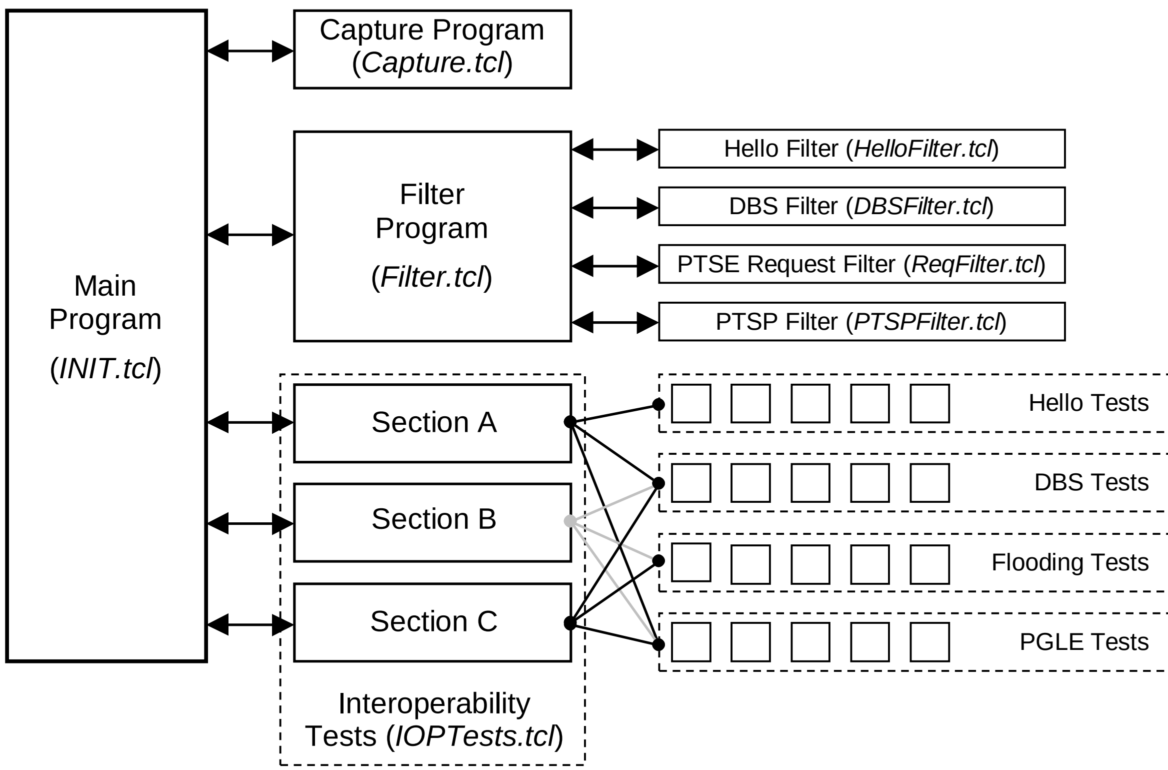

Figure 5: Structure of the interoperability test suite

3 Main Program

The Main program is the heart of the complete suite, in that it combines the intitialization of the tests, capturing of the buffer(s), filtering of the buffer(s) and of course the actual interoperability-testing. Calling the separate components of the suite is the main programs function. The main program is named *INIT.tcl*

For initialization of the test-process, a number of command-line options can be used which are provided to the program upon execution. Depending on the combination of options the various parts of the program are called. The command-line options are:

- -*save* <*Filename*> : This option expresses the intention of saving in a file the packets that are captured by the analyzer. <*Filename*> is the name of the file where these packets are going to be stored.
- -*load* <*Filename*> : Instead of capturing new data, the tests are going to be run using the captured packets stored in the file called <*Filename*>.
- -*portD* <*Interface card number*> : This option should be used when new packets are going to be captured. <*Interface card number*> specifies the interface card used to monitor the link between SUT A and SUT B (see Figure 2).
- -*portA* <*Interface card number*> : This command-line option is used when the analyzer is going to monitor two links, one between SUT B and SUT A and the other between SUT A and Tester A (see Figure 3). Here, <*Interface card number*> specifies which interface card monitors the link between Tester A and SUT A.
- -*time* <*Number of seconds*> : This command-line option specifies the period of time in seconds that the analyzer must capture new packets. The default value, when this option is not given, is five (5) minutes.
- -*linkfail* : This option is used when the test for the Hello protocol that verifies the behavior when a link failure occurs needs to be run. This requires a different capturing approach.

Depending on the given command-line options, the packet buffers can now be loaded or captured. In case the buffer is loaded, by specification of *–load* and the filename, any other command-line option is ignored. When the *–load* option is not given, the buffer must be captured, by calling the *Capture* procedure. The information that is given through the options *–portD*, -*portA*, -*time* and -*linkfail* is passed to this procedure as parameters. After capturing the exchanged packets on one or two links, each buffer is given back to the main program in the form of a keyed list. When the *–save* option is given the captured information is saved to a file. This file is merely a TCL-file that when sourced declares a number of keyed lists which contain the captured data.

Once all the packet buffers have been captured or loaded they are filtered into the format that is used by the individual interoperability tests. This filtering is done to keep the code in the interoperability tests as small as possible and to make retrieval of information as easy as possible. The procedure, *Filter*, is called for each buffer. The filtered buffer is returned as a keyed list.

Before running the tests, the program must check which configuration of the switches has been used. If one buffer (between SUT A and B) had been loaded or captured, the configuration that is used is configuration 1 (displayed in Figure 2). If an extra buffer (between SUT A and Tester A) is available configuration 2 (shown in Figure 3) was used.

In case configuration 1 was used, only section A of the tests must be run, since this is the only one that is possible. Otherwise, if configuration 2 was used, the user is asked to select the section that must be run, since this depends on the configuration of the switches and the settings in each switch. Chapter 6 explains in more detail which configurations and settings are necessary to complete the interoperability test.

4 Capture Procedure

The capture program takes care of capturing the packets that are exchanged between SUT A and B and eventually also between SUT A and Tester A. It is responsible for setting up the analyzer, interaction with the user before and during capturing, and obtaining the buffers that were captured from memory. The procedure, *Capture*, can be found in the file named, *Capture.tcl*.

At first the analyzer must be setup to monitor the specified link or links. For each link the command, *createDatastream*, is called with the port ID of the interface card as parameter. This creates a data stream, which can be referenced by the certain name that was given, for example *mon_pnni1* or *mon_pnni2*. Now, this reference name is used to specify that the analyzer should filter all the PNNI Routing packets that are exchanged using AAL5 over the VPI=0 and VCI=18 pair. This VPI/VCI pair is the default for transmitting and receiving PNNI Routing packets for the lowest level in the topological hierarchy.

In case the user specified to the main program that the Hello test case for link failure must be run, some interaction with the user is necessary. First, the user must make sure that the monitored link is in a stable state, i.e. the Hello protocol is in a 2-Way state, Database Synchronization has finished and the Peer Group Leader has been chosen. When this is the case, the user can disconnect the link and inform the program that this has been done. At this moment the program is ready to capture, since the packets that will be exchanged are the first ones after the link failure.

Now capturing of the packets is ready to begin. If only packets from one link are supposed to be captured, only the command, *mon_pnni1 run*, is used. If there is a second link, then also *mon_pnni2 run* is used. At this moment all monitors are running and the user is informed that the link between node A and B can be connected. After a certain period of capturing, which was specified in the command-line of the main program, there are two possibilities. If there is only one link, the monitor for the link is stopped using *mon_pnni1 halt*. If a second link is available, then the user is asked to connect this link between the two nodes involved. After the same period of capturing, both monitors are stopped using *mon_pnni1 halt* and *mon_pnni2 halt*, respectively.

Processing the buffers that were captured is the next thing that is done. Using the *getDecodes* command retrieves information of all the packets that were captured per monitor. This information includes for each packet: a timestamp and the direction in which the packet was going. This information is returned in a keyed list, so that each element is easily reachable. For example, when the timestamp for the fifth packet is needed, the command: *klget [keylget InfoFrames –5] –time* is used The keyed lists are nested, so that first, the fifth packed must be grabbed and then the key for the timestamp can be accessed. The *detailDecode* command gives the decoded contents of each captured packet. Also here, a nested keyed list is returned which contains keys for every field of the packets that were captured.

Finally, all used monitors are removed using the command, *deleteDatastream*, and the captured buffers are returned to the main program.

5 Filter Procedures

Since the buffers that are captured with the *Capture* procedure are not in a format that is easy to use, a filter must be applied to convert the buffers into a different format. To be more precise, the main reasons for filtering are:

- In the output from the analyzer, every Node ID is split in separate fields. This has been done according to the suggestions that were made in section 5.3.3 of [ATM Forum, 1996]:

  1. In case the second octet has a decimal value of 160, the ID is split into parts under the assumption that the node does not represent a peergroup and thus is a lowest level node:

Level Indicator	Second Octet	Authority and Format ID	ICD	HO DSP	ESI	SEL

  2. In case the second octet has a decimal value different from 160, the ID is split into parts under the assumption that the node represents a peergroup and thus is a logical group node:

Level Indicator	Level	Identifier Information	End System Identifier	Remaining Octets

  The PNNI specification states that the Node ID must be formatted as a Level Indicator plus a twenty-octet opaque value. In this way, the decoding strategy of the analyzer is strange, certainly when the decision for a certain format is made by looking at the second octet. Since the tests only use the Node ID as one piece, without even looking at the level, the different parts should be combined into one value.

- Also the Peergroup IDs are split by the analyzer into two parts, namely *–Level* and *–Identifier_Information*. During testing only the complete Peergroup ID is used, so these fields must be appended to each other and form one key.

- In the analyzer output, every packet consists of a keyed list that is non-nested. This means that fields in the packet that have the same name have different key names. Appending a different number to the key differentiates them. For example, if there is more than one Type field in a packet the first field has the key –Type, the second –Type2, the third –Type3, and so on. Retrieving the information from the keyed lists becomes very tedious when this format is used, especially when keys that belong together (i.e. the keys within one PTSE) are numbered differently. Therefor the format of the packets must be changed and incorporate nested keys, so that fields that belong together can be accessed more easy.

- During decoding, the analyzer adds additional information to most of the fields. For example the hexadecimal value is additionally converted into a decimal value and added as a comment or text is added to explain the value of the field. Since the plain value of a packet field is enough during testing, the extra information is removed during filtering.

The *Filter* procedure takes every packet from a buffer and examines what type of packet it is. Accordingly the individual filters are called:

- Hello packet → *HelloFilter*
- Database Summary packet → *DBSFilter*
- PTSE Request packet → *ReqFilter*
- PTSP packet → *PTSPFilter*

Each filtering procedure accepts a keyed list containing the packet that must be filtered. Returned is the keyed list that has been filtered and reformatted. The *Filter* procedure replaces the corresponding key in the keyed list of the total buffer. The next sections will give comment on each of the packet filtering procedures.

5.1 Hello Filter

The procedure in charge of filtering the Hello packets is called *HelloFilter*, which can be found in the file, *HelloFilter.tcl*. The Hello packets are filtered on the following points:

- The keys that make up the Node ID, Remote Node ID and Peer Group ID are combined into three keys called -*Node_ID*, -*Remote_Node_ID* and -*Peer_Group_ID* respectively. This is realized by appending the fields to each other.
- Some fields have been filtered to remove additional information:
  -*Remote_Port_ID*
  -*Hello_Interval*

5.2 DBS Filter

The filter procedure for Database Summary packets is contained in the file, *DBSFilter.tcl*. The procedure *DBSFilter* filters the packets on the following points:

- The keys that make up the Originating Node ID are combined into one key. Appending the fields to each other does this.
- The keyed list which contains the DBS packet is given a hierarchical structure as follows:

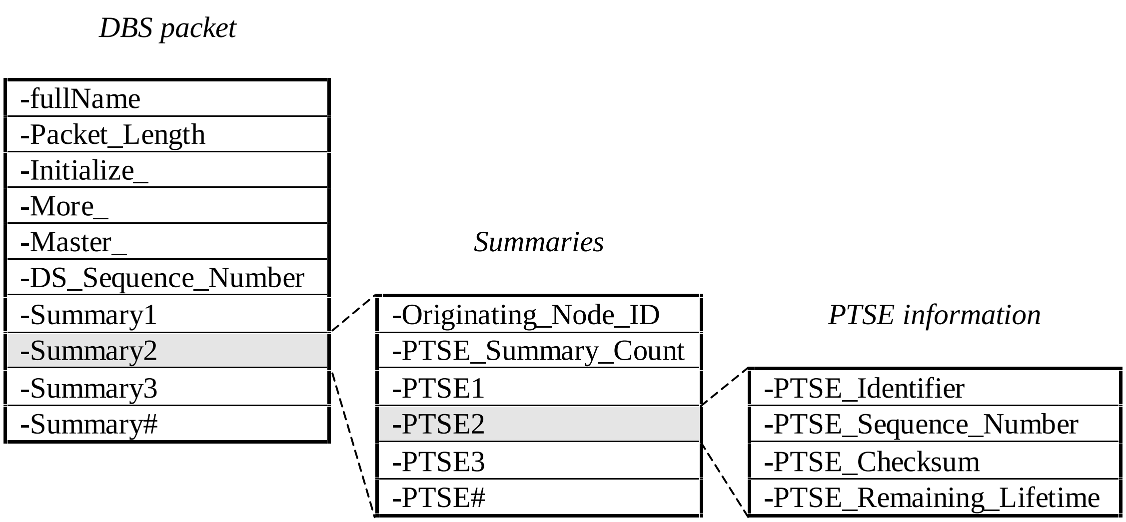

Figure 6: Structure of DBS frame in keyed list.

- Some fields have been filtered to remove additional information:

-*Packet_Length* :	only hexadecimal value is copied
-*Initialize* :	only the value of the Initialize bit is copied
-*More* :	only the value of the More bit is copied
-*Master* :	only the value of the Master bit is copied

-*PTSE_Summary_Count* : the hexadecimal value is converted into decimal

- Any other field that is not mentioned in the above structure is removed from the frame, since this information is not used by the tests currently.

5.3 PTSE Request Filter

The file, *ReqFilter.tcl* contains the filtering procedure for the PTSE Request packets, which has the name *ReqFilter*. The packets are filtered on the following points:

- The keys that make up the Originating Node ID are combined into one key. Appending the fields to each other does this.
- The keyed list that contains the Request packet is given a hierarchical structure as is shown in the following figure.

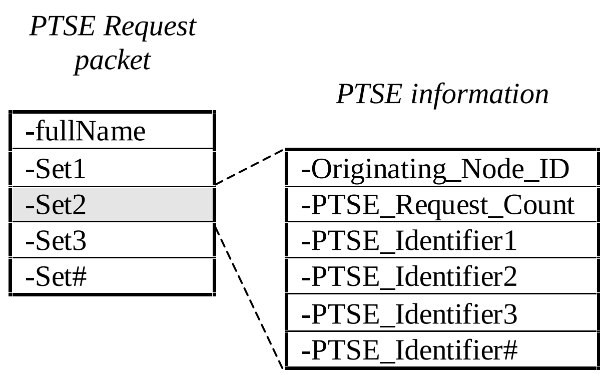

Figure 7: Structure of PTSE Request frame in keyed list.

- In the field -*PTSE_Summary_Count* the hexadecimal value is converted into decimal.
- Any other field that is not mentioned in the above structure is removed from the frame, since this information is not used by the tests currently.

5.4 PTSP Filter

The procedure that filters the PTSP packets is called *PTSPFilter* and is contained in the file, *PTSPFilter.tcl*. The PTSP packets are filtered based on the following points:

- The keys that compose the Originating node ID are appended together creating a single key called -*Originating_Node_ID*.

- The keyed list which contains the PTSP packet is given a hierarchical structure as follows:

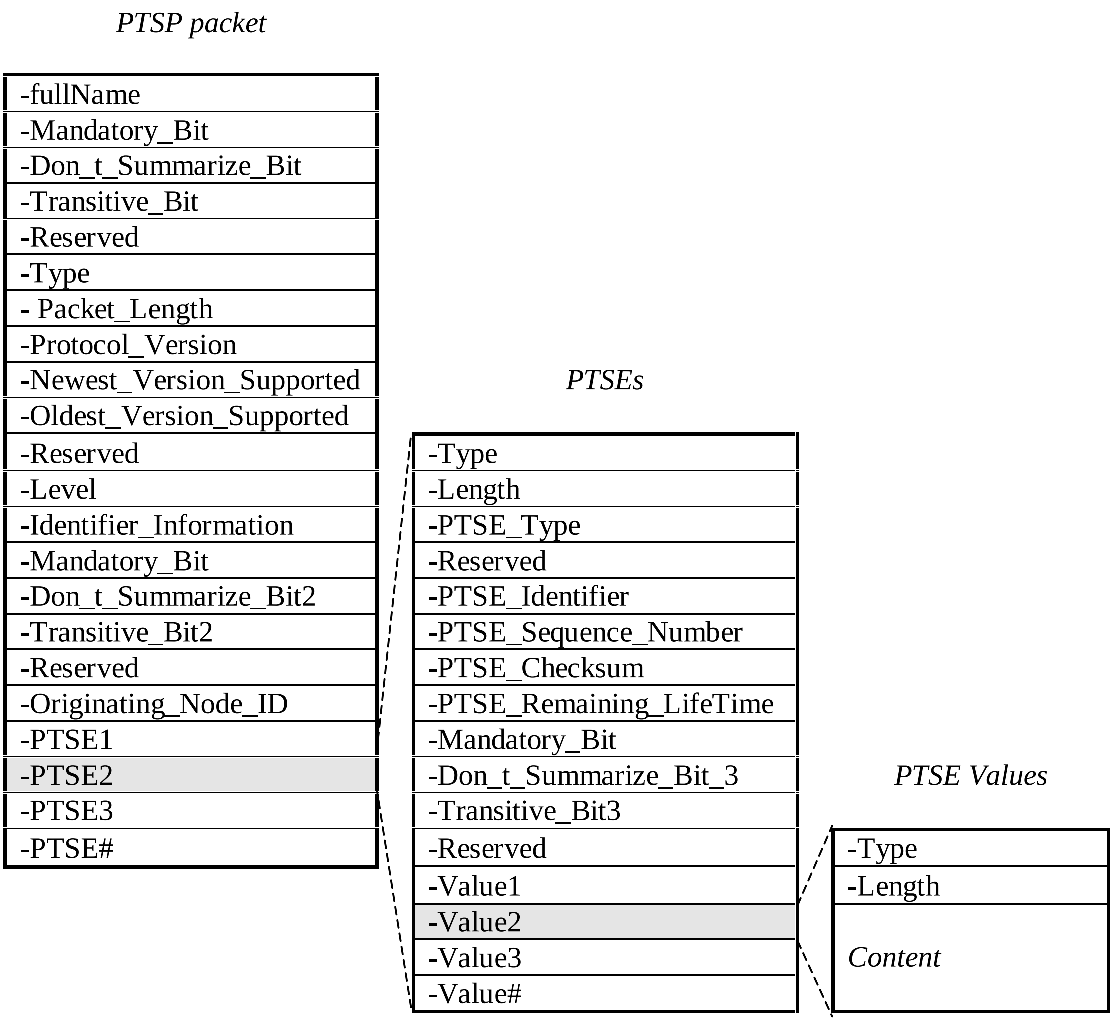

Figure 8: Structure of PTSP packet in keyed list.

- When a PTSE contains a Nodal IG, this IG is filtered as follows:
  - The keys that make up the Preferred Peer Group Leader ID are appended together in a single key called -*Preferred_PGL_ID*.
  - From the following fields additional information is removed:

    -*Leader_Flag* :
    -*Leadership_Priority* :
    -*Non_transit_for_PGL_Election_Flag* :

- When a PTSE contains a Nodal IG, in this IG the keys that make up the Remote Node ID are appended together in a single key called -*Remote_Node_ID*.

6 IOP Test Procedures

Due to the different topology configurations, the tests are divided in three different sections. These three sections are called, section A, B and C. This division is necessary since the data captured for running the tests for one of the sections can not be used for running another section. Three procedures have been implemented, each one running the tests for each section that are called, *IOPSectionA, IOPSectionB* and *IOPSectionC*.

These three procedures have been implemented with the same structure. First, general information is extracted from the captured data:

- Node, Peer Group and Port IDs from the nodes involved in the tests. These values are obtained from the first Hello packet received from each node.
- Leadership Priorities from the nodes involved in the tests. The Leadership Priority is obtained from the first Nodal IG contained in a PTSP originated by each node.

This information is stored in global variables that are used for automating the execution of the tests within a section.

- Depending on the Peer Group ID the tests for an outside link from the Hello protocol are run or not. If the Peer Group IDs from the nodes are different, then the tests for Database Synchronization, Flooding and Peer Group Leader Election are not run.
- Depending on the value of the Leadership Priorities, different tests for the Peer Group Leader Election are run.

After running the tests according to the internal configuration of each switch, a summary of the success, failure or non-execution of the tests is given.

For fully testing Interoperability between two switches the following steps are needed.

- First it is necessary that each SUT successfully passes the tests from section A. For running all the test from section A, the settings of each SUT must be changed (settings are considered to be the Node ID, Peer Group ID and Leadership Priority).
  - Both switches within the same Peer Group.
    1. Both switches with Leadership Priorities higher than zero and different.
       (1.1) Where SUT A has a Leadership Priority greater than SUT B.
       (1.2) Where SUT B has a Leadership Priority greater than SUT A.
    2. Both switches with the same Leadership Priorities, that is higher than zero.
       (2.1) Where SUT A has a higher node ID than SUT B.
       (2.2) Where SUT B has a higher node ID than SUT A.
    3. Both switches with Leadership Priority equal to zero.
    4. SUT A with Leadership Priority higher than zero and SUT B with Leadership Priority equal to zero.
    5. SUT B with Leadership Priority higher than zero and SUT A with Leadership Priority equal to zero.
  - Each SUT is in a different Peer Group.
  - After the Hello protocol in each SUT is in the state, Two Way Inside, disconnect the switches, and run the test V4201H008 for link failure.

- Once the SUTs have passed section A successfully, section B and C must be run, in either order. It does not matter.
  - The SUTs have to passed all the tests from section B. For this purpose different combinations of the Leadership Priorities are needed.
    1. SUT B has a Leadership Priority of zero. SUT A has a Leadership Priority higher than zero. The Tester A has a Leadership Priority greater than SUT A's original Leadership Priority plus Group Leader Increment.
    2. SUT B has a Leadership Priority of zero. SUT A has a Leadership Priority higher than zero. The Tester A has a Leadership Priority smaller than SUT A's original Leadership Priority plus Group Leader Increment or equal to zero.
    3. SUT B and SUT A have a Leadership Priority of zero. Tester A has a Leadership Priority greater than zero.
  - The SUTs have to passed all the tests from section C as well. The different combinations of the internal settings of the switches used for running all the tests for this section are:
    1. Tester A, SUT A and SUT B have Leadership Priority of zero.
    2. Tester A has a Leadership Priority greater than zero. SUT A and SUT B have a Leadership Priority equal to zero.

6.1 Section A Tests

The section A of the Interoperability tests, is the group of tests that should be run when the switches have a topology configuration 1 (see Figure 2). The procedure for running the tests from section A is called *IOPSectionA*, and is stored in the file *IOPTests.tcl*.

This procedure is called by the main program, and has as parameters: *{FrameBuff InfoFrames LinkFailureTest }*, where *FrameBuff* is the keyed list containing the frames interchanged by SUT A and SUT B that have been captured by the analyzer. *InfoFrames*, is the keyed list containing the information frames about the packets exchanged between SUT A and SUT B. *LinkFailureTest* is a variable, which value can be 0 or 1. When *LinkFailureTest* has a value of one, this means that the test that must be run is V4201H008, the test that checks if the switches can recover from a link failure.

The flowchart in Figure 12 in the appendix shows how the tests for section A are executed.

6.2 Section B Tests

The section B of the Interoperability tests, is the group of tests that should be run when the switches have a topology configuration 2 (see Figure 3) and when the link between the SUT A and SUT B is operational and stable before the link between SUT A and Tester A is brought up.

A "stable link" is defined to be when the SUTs at both sides of the link have passed the tests from section A successfully (i.e. the Hello protocol in each SUT is in the state, Two Way Inside; the Database Synchronization protocol in each SUT is in the state, Full; and the Peer Group Leader Election is finished). The procedure for running the tests from section B is called *IOPSection*B, and is stored in the file, *IOPTests.tcl*.

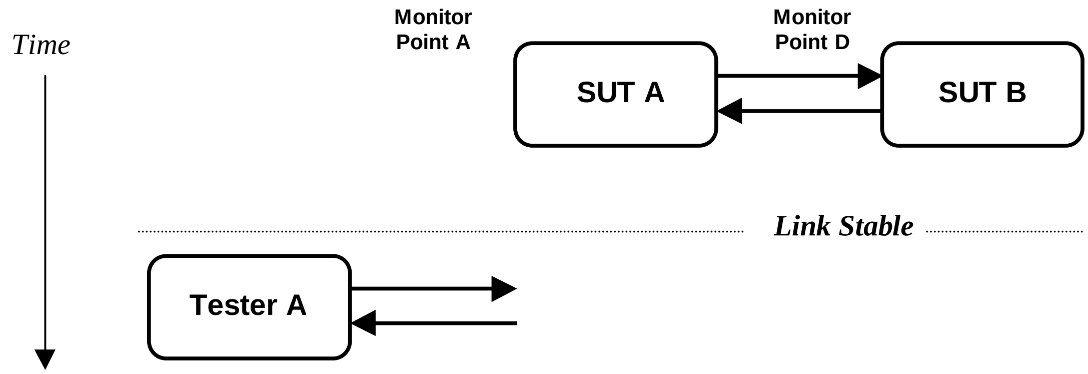

Figure 9: Sequence of actions for capturing packets. When the link between SUT A and B is stable, Tester A is connected to SUT A.

This procedure is called by the main program, and has as parameters *{ FrameBuff InfoFrames FrameBuff_T InfoFrames_T }* where *FrameBuff* is the keyed list containing the frames exchanged by SUT A and SUT B that have been captured by the analyzer. *InfoFrames*, is the keyed list containing the information frames about the packets interchanged between SUT A and SUT B. *FrameBuff_T* is the keyed list where the packets exchanged by the SUT A and Tester A have been stored. *InforFrames_T* is the keyed list where the information frames about the packet exchanged between SUT A and Tester A have been stored.

In the appendix in Figure 13 a flowchart is included showing how the tests for section B are automatically executed.

6.3 Section C Tests

The section C of the Interoperability tests, is the group of tests that should be run when the switches have a topology configuration 2 (see Figure 3). The link between the SUT A and SUT B is operational and stable before the link between SUT A and Tester A is brought up. It is considered a stable link, when the Hello protocol of the SUTs is in the state, Two Way Inside; the Database Synchronization is in the state, Full; and the Peer Group Leader Election protcol is finished in both SUTs. This is also equivalent to when both SUTs have passed the tests from section A successfully. The procedure for running the tests from section C is called *IOPSection*C, and is stored in the file, *IOPTests.tcl*.

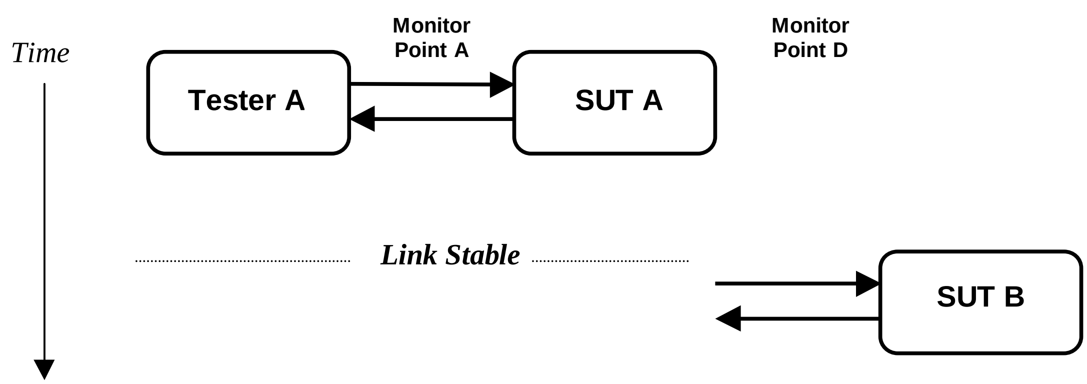

Figure 10: Sequence of actions for capturing packets. When the link between SUT A and Tester A is stable, SUT B is connected to SUT A.

The procedure, *IOPSectionC* has as parameters *{ FrameBuff InfoFrames FrameBuff_T InfoFrames_T }* where *FrameBuff* is the keyed list containing the frames exchanged by SUT A and SUT B that have been captured by the analyzer. *InfoFrames*, is the keyed list containing the information frames about the packets exchanged between SUT A and SUT B. *FrameBuff_T* is the keyed list where the packets exchanged by the SUT A and Tester A have been stored. *InforFrames_T* is the keyed list where the information frames about the packet exchanged between SUT A and Tester A have been stored.

In the appendix, Figure 14 is a flowchart showing how the tests for section C are automatically executed:

6.4 Implementation of a test case

Each test is implemented according to the test cases as presented in [ATM Forum, 1998]. Basically there is a one to one relationship between the theoretical test cases and the implementation thereof. Each implemented test has about the same structure.

The frames from a buffer, passed to the test as a parameter, are scanned one by one using a foreach loop. The values of each scanned frame are examined, looking for the type of packet with specific content, specified by the Interoperability tests specification. The tests look for a sequence of sent packets with certain content. Once this sequence is found, a variable called result is set to 1 meaning that the test has been successful. Otherwise, if the sequence is not found, the variable, result, will be set to zero, meaning that the test fails. If the test was successful, in most of the cases a variable is set to the number of the last scanned packet, this variable is called mark. When a test is used as a preamble of another one, the variable mark tells the last packet needed to be scanned by the preamble. Figure 11 shows the general structure of an implemented test case as was explained.

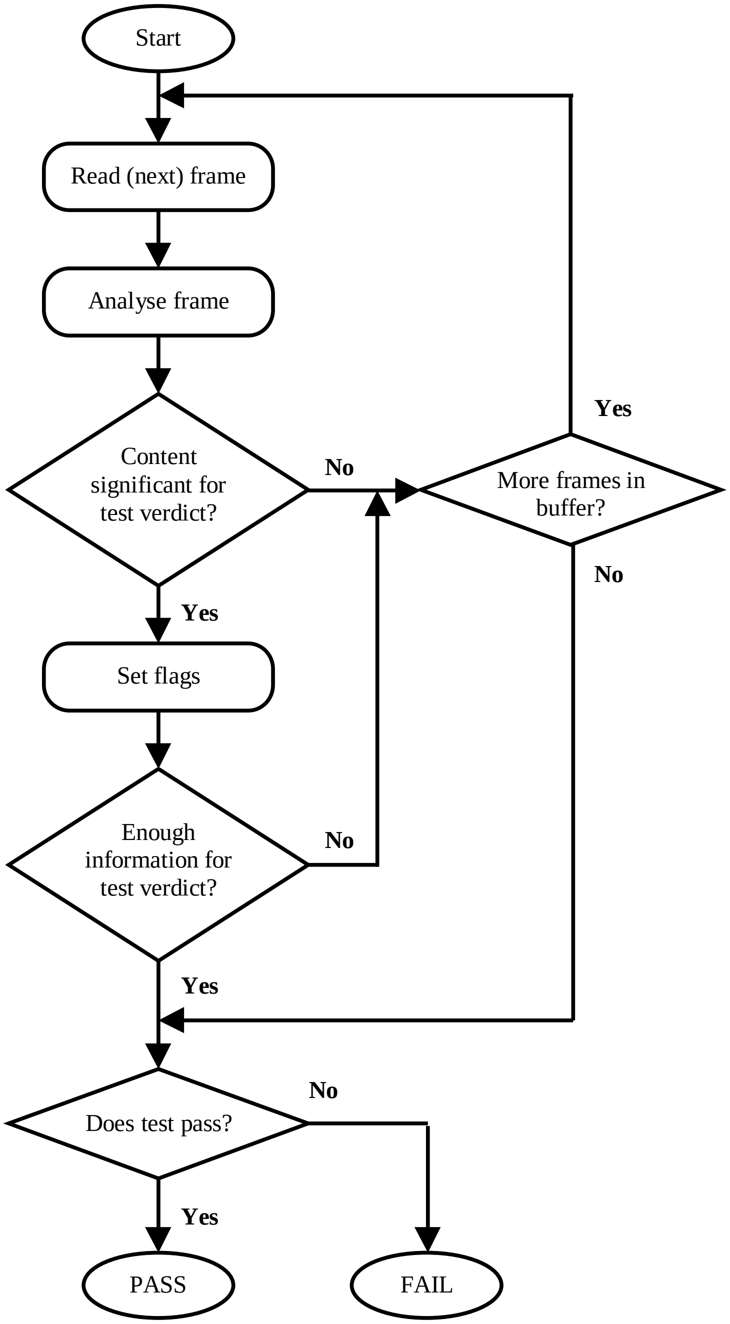

Figure 11: General operation of implemented interoperability test case.

7 Testing Results

These tests were run on the NIST ATM PNNI testbed, which consisted of ATM PNNI switches from three different manufacturers. The results were satisfactory. However, not all of the tests were completed on all of the switches due to the limited implementation of the PNNI protocols some switches.

Conclusions

Conformance testing must be done first, to reduce the number of manual decodes and traces when automating interoperability tests.

Using these tests are helpful in trouble shooting the inter-operation of ATM switching systems, even though it is not real time.

The Signalling interoperability tests that are described in the [ATM Forum, 1998] could be added. There has been some additional routing tests added to the straw vote document, "Interoperability Tests for PNNI since the October meeting. See the latest version of this document for possible additions to this interoperability test suite.

Abbreviations

AAL	-	ATM Adaptation Layer
DBS	-	Database Synchronization
PGLE	-	Peer Group Leader Election
PNNI	-	Private Network to Network Interface.
PTSE	-	PNNI Topology State Element
PTSP	-	PNNI Topology State Packet
SUT	-	System Under Test.

References

[ATM Forum, 1996] Private Network-Network Interface Specification Version 1.0 (PNNI 1.0), af-pnni-0055.000, March 1996

[ATM Forum, 1997]. PNNI v1.0 Errata and PICS, af-pnni-0081.000, May 1997

[ATM Forum, 1998] Interoperability Tests for PNNI, str-test-pnni-iop-00.01, October 1998.

[ATM Forum, 1994] Introduction to ATM Forum Test Specifications, af-test-0022.000, December 1994

Appendix

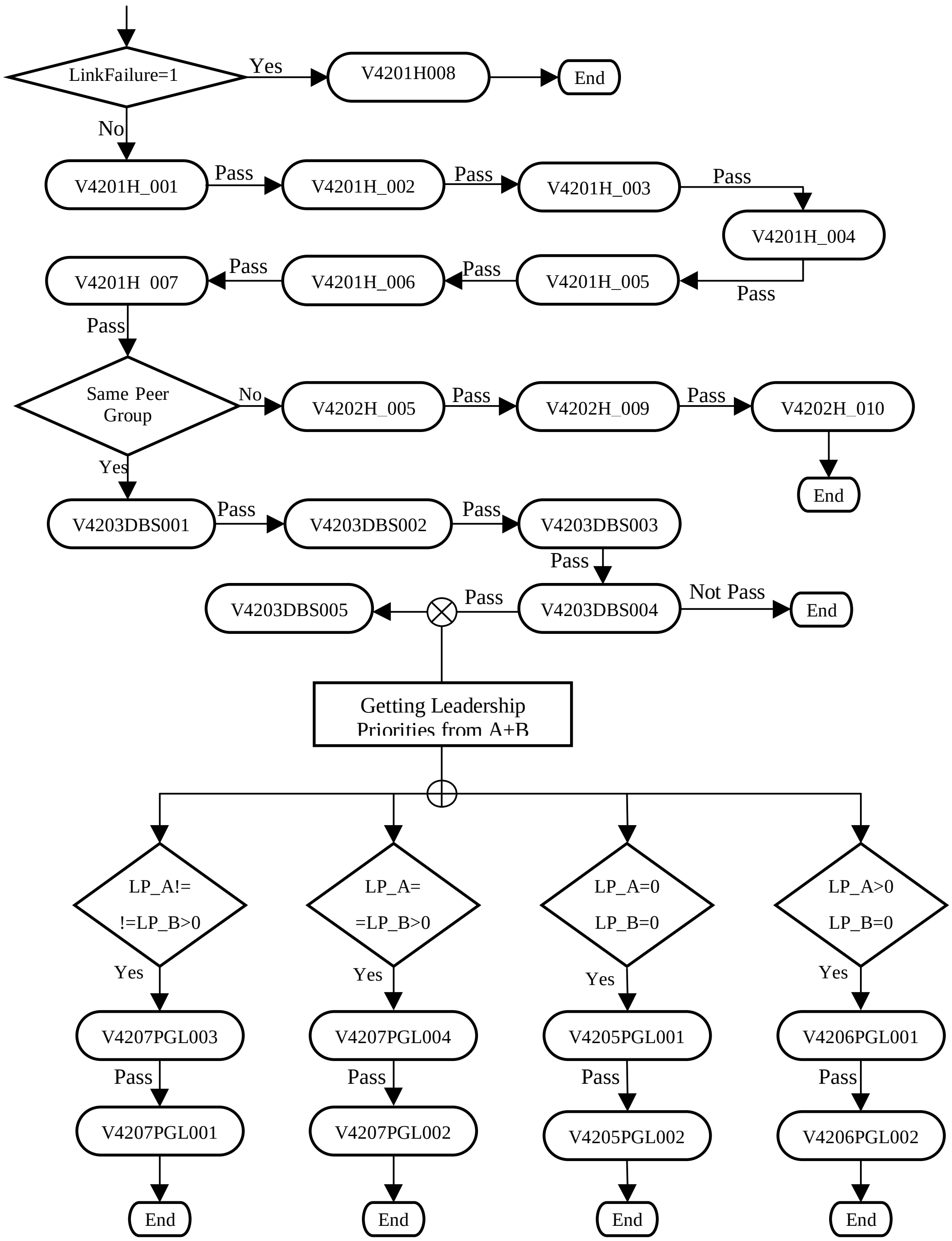

Figure 12: Flowchart showing the sequence of execution of test in section A

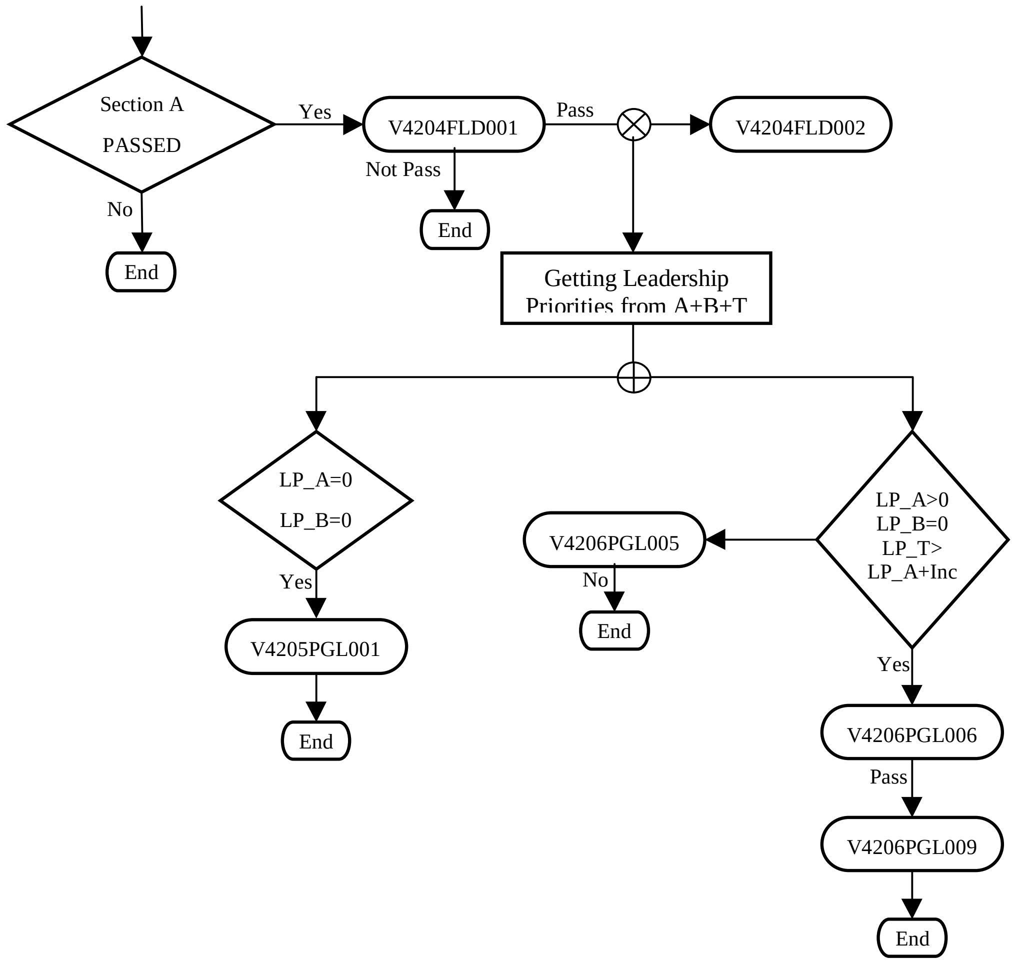

Figure 13: Flowchart showing the sequence of tests run during section B.

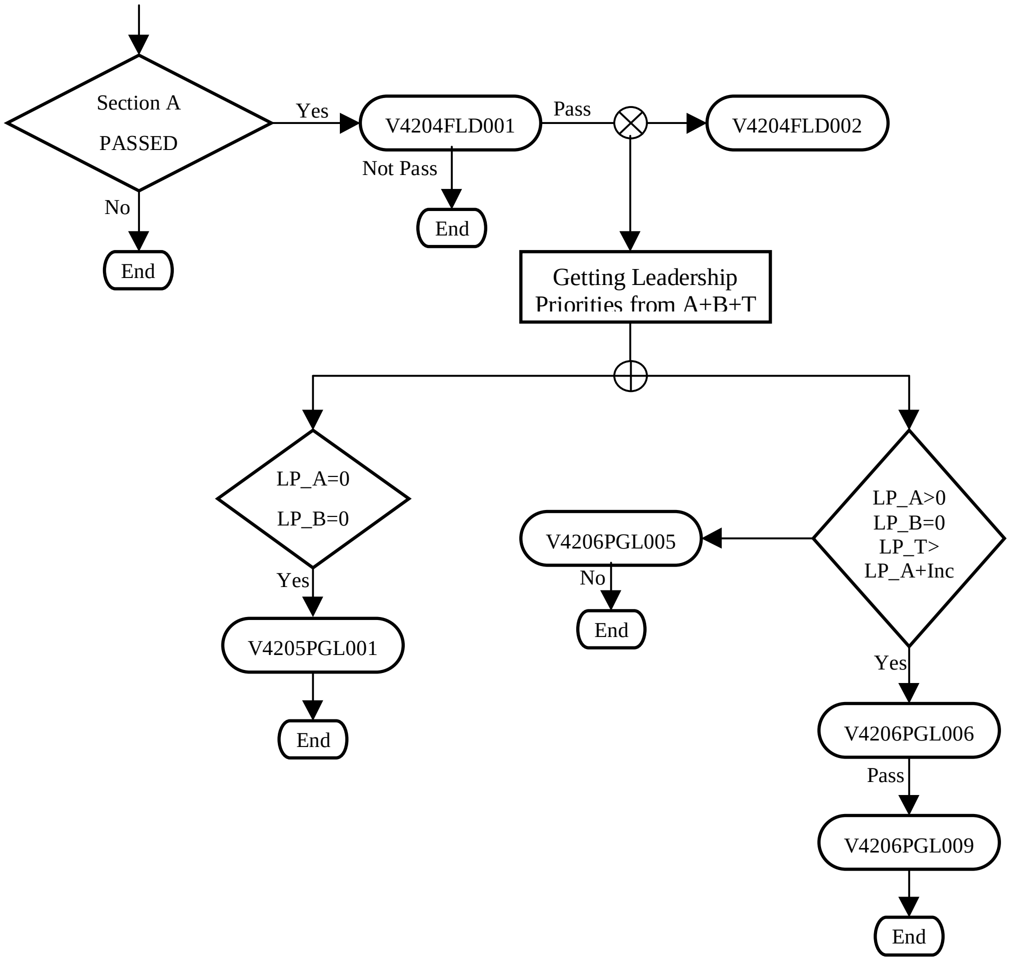

Figure 14: Flowchart showing the sequence of tests in section C