CSE/IT 353 Project #2

Simple Token Passing LAN Protocol (STPLP)

Due: 11:59pm Sunday, November 2nd

GOAL

You are going to implement a Token Passing MAC protocol simulator. To do this, you are to implement and operate a "simplified" ***Token Passing*** system with a variable number of nodes. You are given input files for each node in the network, and you must implement the specified protocol ***STPLP*** to produce the expected correct output files at each node, for protocol validation.

DESCRIPTION

Create a program that simulates a physical ring network topology that uses token passing (i.e. modified Token Ring). This program will create autonomous threads that will act as nodes within the network, which represent the different computers on the network. These nodes will communicate via sockets, which represent the physical link that connect the nodes. Each node will need to transmit data to other nodes on the network using the protocol specified below. Each received packet will be recorded by the receiving node (details described below). In addition to the standard nodes, this network should also include a monitor node that monitors the health of the network (details described below).

After transmitting all data in each node, the program should exit, with return 1. (Note: This will require some thread synchronization.) The code must correctly process an input file, with a specific format (discussed later), and produce an expected results at each node (for protocol verification by the class TA upon grading your ***STPLP*** submission). Your program may print to *stdout* any messages that are useful in the protocol analysis and debugging. The output to the output-files must be correctly formatted as specified below for grading proposes. The protocol must conform to the specification outlined later in the document.

REQUIREMENTS

The implementation is to use *TCP/IP* sockets (for connecting neighboring nodes).

1. **Language:** You are required to implement the project in one of the following programming languages: Java, C#, or C++.
2. **Object Oriented Structures**: your code should use valid classes and object-oriented structures to implement the nodes.

3. **Environments**: Your program must be able to compile and run on the TCC machines. All projects will be graded on TCC, and if your project does not compile or run on this system, then your grade will suffer.
4. **Make file, build.xml, or MSBuild file**: You are required to create one of these in order to allow easy compilation from the command line. **In order for your project to be graded**, you must include **instructions** on how to use this file in order to compile and run your program from **command line**.
    a. **C++:** Create a make file.
    b. **Java:** You can create an Ant build.xml file or a make file. (Yes, make files work with Java programs; it's just not considered best practice).
    c. **C#:** Create an MSBuild file.
5. **Documentation:** You need to document the following
    a. Your code should be well commented.
    b. How to compile and run your program. If your program is platform specific, make sure to document which OS (Windows or Linux) must be used to compile and run your program.
    c. The names of all files in the project and a brief description of what they are.
    d. Provide a check list that includes what features are implemented and missing. See below for the checklist.
    e. A list of all known bugs. Reported bugs may reduce the corresponding point deduction.
    f. The Token Holding Time (THT) you selected for your program, and your reason for selecting that time.
    g. How you chose to simulate "garbled" frames (details in Monitor Node section below).

Monitor Node

The network should have a dedicated monitor node (node #0) whose sole purpose is to monitor the health of the network. The monitor node has the following responsibilities:

1. **Detecting lost tokens**. The monitor should keep an eye on the network traffic and look for whether or not the token has been lost. To accomplish this, it should keep a timer with the longest tokenless interval (sum of the token holding time of all nodes in the network). If the network traffic is dead for the length of the timer, then the monitor should assume that the token has been lost. It should then drain the ring and issue a new token.
2. **Cleaning garbled frames from the network**. The monitor node should check that all frames conform to the proper frame format for this network. If an invalid frame format is detected, it should drain the ring and re-issue the token.
3. **Detecting orphaned frames**. The monitor node should check if frames have been orphaned on the network. This is done through the "M" bit in the FC byte of the frame format. On frame creation, the "M" bit should be initialized to 0. The first time the monitor sees a frame, it should set the "M" bit = 1. If the monitor node sees a frame with an "M" bit = 1, it means that the destination node did not drain its

own frame for some reason. In this case, the monitor should clean the orphaned frame from the ring.

In order to ensure that the monitor node is functioning properly, other nodes in the network should have the following behavior:

- Each node should have a 5% chance of "losing" the token when passing the token frame to the next node.
- Each frame should have a 1% chance of having the wrong frame format at creation. You may implement something simple such as omitting a byte during frame creation from the frame format. However, the monitor node should check that all elements of each frame matches the proper frame format. **DOCUMENT your choice for how you simulated garbled frames.**
- Orphaned frames may occur as a result of frame rejection (see MAC protocol below). Each destination node should have a 20% chance to reject an incoming data frame, and an 80% chance to accept said data frame. Note that acceptance/rejection should only happen at the destination nodes, NOT during the relaying of frames through the network.

COMMAND-LINE ARGUMENTS

The program must accept a single command line argument that specifies the number of nodes in the network (e.g .. /Tokenring 5). There may be up to **254** nodes in the ***STPLP*** (plus one monitor node #0).

INPUT FILE

This file will contain the node traffic (***STPLP*** frames) to be transmitted to other nodes in the network. Included with the project is a *python* script to generate a random *number of frames*, *frame size*, *data field*, and *destination node address*. The files will be named ***input-file***[node***#***]. Each frame will be placed on a separate line in the input file. The data field will contain arbitrary uppercase ASCII text with the length specified.

The input *frame format* will be:

<***Destination_Address***>, <***data_size(in bytes)***>, <***data***>

Example: A frame to be sent from node 1 to node 3 will appear as follows in the file input-file-1:

3,10,xxxxxxxxxx **x**:byte

In this example, the source address is implicitly 1 because it is located in the input file of node 1.

Output File

Upon receiving a frame, a station will extract the data part (LLC frame) and source address from the frame and place it on a separate line in the output file. This file will be used to verify the protocol correctness based on the input files used. The output files shall be named ***output-file***-[node #].

The output format will be:

<Source_Address>, <Destination_Address>, <***data_size(in bytes)***>***, <data>***

Example: The above frame from node **1** to node **3**, as extracted from the network, will appear in output-file-3 as:

1,3,10,xxxxxxxxxx **x**:byte

STPLP Physical Layer

There are many ways to implement the physical layer of the proposed LAN, ranging from the actual point-to-point physical connections to the virtual emulation of a network. For this project, you are required to use *TCP/IP* sockets to emulate the physical layer connection between the nodes in the network. You may use Java, any standard C++ socket headers, C, or the Windows API to implement sockets, depending on your language choice.

Medium Access Control Protocol of the *STPLP*

Frame Format: (All fields are one byte each, except the DATA field (to be of size SIZE)

AC	FC	DA	SA	SIZE	DATA (LLC PDU)	FS

AC: Access Control Byte

PPP	T	M	RRR

First 3 bits (PPP): Frame priority level. *Used for priority extra credit. If not implemented, set to* O.

Forth bit (T): Token bit- 0 if token, 1 if frame.

Fifth bit (M): Monitor bit- used to detect orphaned frames. At frame creation, the node should set M = 0. When the monitor sees a frame, and M = 0, then it should update the monitor bit to M = 1 (denoting that the monitor has now seen the frame). If the monitor sees a frame with M = 1, then the monitor should drain the orphaned frame.

Last 3 bits (RRR): Reservation level. *Used for priority extra credit. If not implemented, set to* O.

FC: Frame Control Byte - 0 if token, 1 if frame

DA: Destination Address Byte

SA: Source Address Byte

SIZE (bytes) - Size of Data PDU, up to 254 bytes

Data - Binary ASCII data transmitted (of length Size)

FS: Frame Status Byte - Initially set to 0 by the source node. The lowest two bits, xxxxxxXY, in this byte are used to ACK the sender. X determines whether the destination received the packet and Y describes whether the destination accepted the packet. Note: Frame acceptance shall be randomly determined in this project and *not* be the result of a checksum (We're not that mean!). **Nodes should have a 20% chance to reject a data frame, and a 80% chance to accept it.**

Since we assume that the destination node is always alive, the destination node can set the FS to 2 if the frame is accepted and 3 if the frame is not accepted.

Token Holding Time (THT)-Byte-Count is the upper limit to the number of data bytes that a node can send when holding the token. You may choose THT to be any value greater than the maximum PDU length (254), but DOCUMENT your choice and the REASON that you believe that your value is a good choice. Before sending a frame, be sure that the node will not violate the THT!

ADVICE

Remember that you have to shut down the network when all of the nodes are done transmitting. It is allowable that you implement this using some form of global variable because shutting down the network is not part of this protocol. However, there are better ways that will earn you extra credit (see the Extra Credit section for more information).

"TokenRing.cs" file provides a template which can help you to build the code structure. It is optional, you do not have to utilize it in your solution; unless you really need a "jump start".

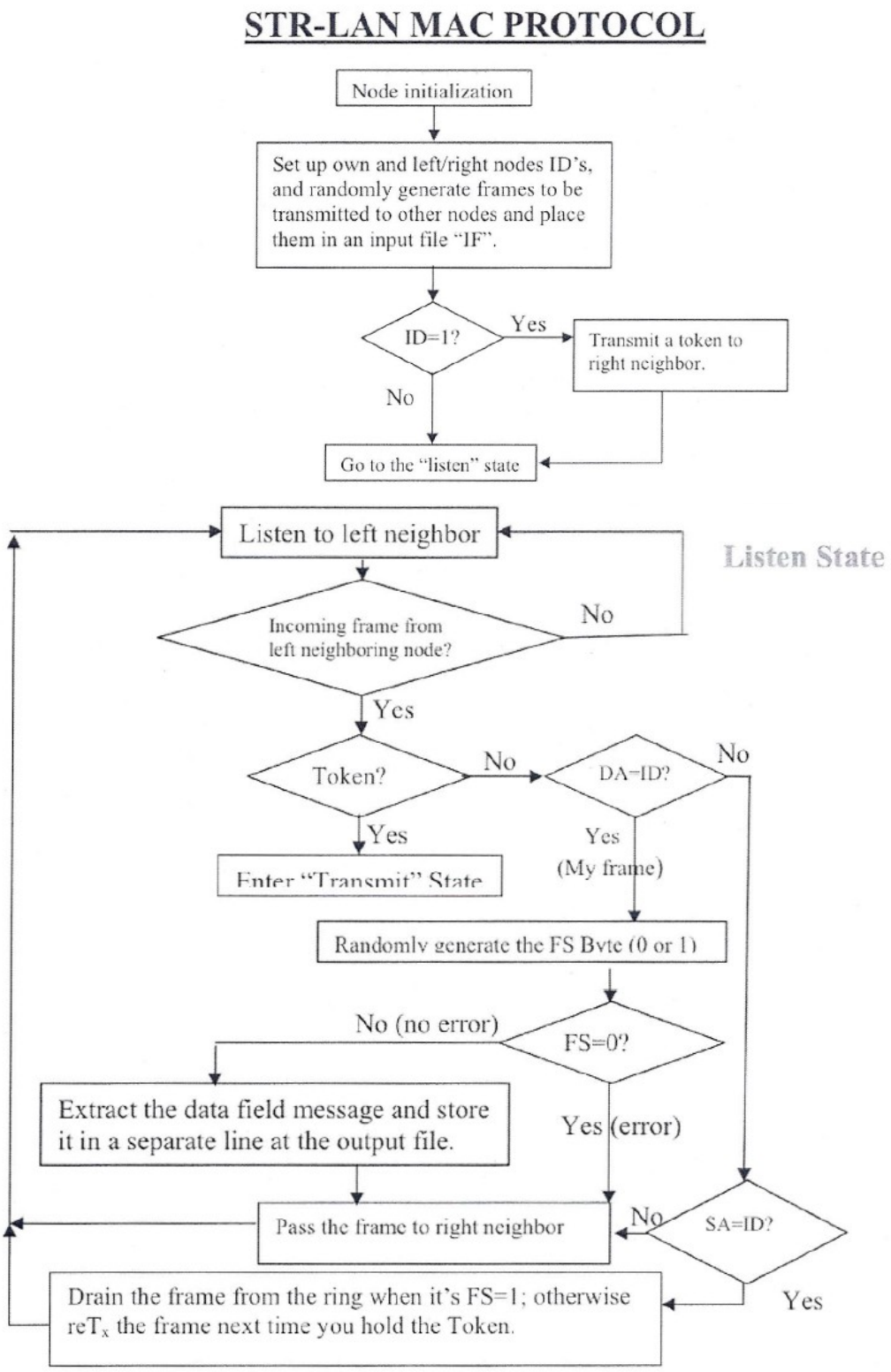
"Project_2_GenerateInput.py" is used to generate input data.

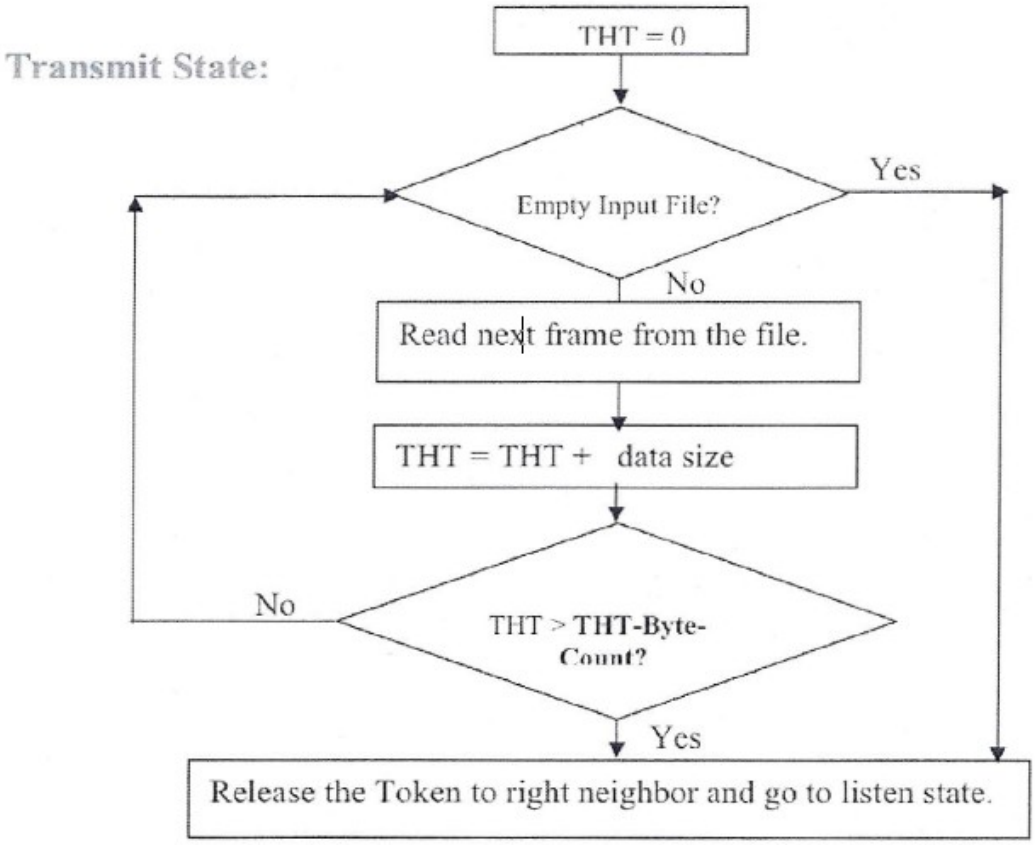
"Grade.py" is used to validate your output.

"Project_2_GenerateInput_prio_2013.py" is used to generate input data with priority. Only used for extra credits.

“Grade_prio_2013.py” is used to validate your output for priority Token Rings. Only used for extra credits.

PROTOCOL DIAGRAMS





Grading

We will use several randomly-generated input files of differing sizes to check that your program issues the correct output files for the given input. The files used for grading will be generated by the same script that is made available to you for testing. The project will be graded according to the following rubric:

Implementation (75):

- Make file: _____/5
- Correct implementation (it works): _____/50
- *Correct protocol implementation: _____/10
- **Program exits appropriately: _____/10

Documentation (25):

- In code documentation (comments): _____/10
- ***Documentation of project in your README: _____/15

*Correct protocol implementation means that your protocols match the specifications.

**The program is considered to have an appropriate exit if it the network shuts down if and only if all nodes are done transmitting. Also, all of the sockets should be closed at exit.

***Your README Documentation should include at least:

- DETAILED compilation and run instructions
- File descriptions
- Feature checklist
- Token Holding Time and reasoning
- Garbled frame simulation choice
- List of known bugs (or a clear statement that thee are no known bugs)

All features in your program must be thoroughly documented in your README. You are allowed to make "improvements" to this protocol with the mutual consent of the instructor and the class graders. Points will be deducted for inconsistent coding style and improper commenting.

CHECKLIST

Include the following checklist in your documentation. Each item should be marked as complete, missing or partial. In the case of a partial status, make sure to add a description. Also list any extra credit work (given that you obtained the TA's permission to do it). The following is an example. Change the "Status/Description" column for your own project.

(You are not limited to the included items, you might add more as you see important to report)

Item	Status/Description
Load input file	complete
Parse input file	missing
Generate token.	partially - token may be missing.
Generate frame.	complete
Node runs as an autonomous thread	missing
All nodes terminate after all data in the network were transmitted	partially - some nodes cannot terminate
Follow the command line to execute the program	missing
Randomly accept a frame (randomly generate the FS)	missing
Extract all fields from a frame	missing
Switch between *listen* and *transmit* states .	missing
Pass the tokens and frames next neighbor.	missing
Print to the Output file with correct format.	missing
Documentation	missing
Executables files	missing
Monitor detects lost token	complete
Monitor cleans up garbled frames	Partially - sometimes it misses one
Monitor detects orphaned frames	missing

Well documented code.	Partially - all classes are documented, but only a few functions and methods are documented. **(It's not necessary to strictly follow certain documentation format, nor document every parameter. Just briefly describe what a class / method [or func/proc in non-OOL] will do)**
Optional extra credit work: node ***priority***	Partially - this feature may crash when ....

EXTRA CREDIT

There are several options for extra credit. Each one can add additional points to your grade. Note that all extra credit must be well documented in order to receive the full points.

1. **Binary frame** implementation (as opposed to bumping strings over the wire): 10%
2. **Network shuts down using inter-node communication** in unused protocol bits: 10%
   The monitor node makes it much easier to shut down the network. Note that there are unused data fields in the above frame format. Develop a protocol using the unused data fields that allows you to communicate between the monitor and the nodes and shut down the network when all of the nodes are done transmitting. Document exactly what you do to implement this feature.
3. **Node Priority:** 15%
   The implementation of node priority is specified in the lecture notes. Please conform to that protocol or document any deviations. Note that the python scripts designed for testing priority support require a slightly different input and output file formats (to include priority of frames).
   Input file format:
   <Destination_Address>,<Frame_Priority>,<Data_Size>,<data>
   Output file format:
   <Source_Address>,<Destination_Address>,<Frame_Priority>,<Data_Size>,<data>
4. The implementation of any ***creative features*** will be rewarded, but you MUST obtain prior authorization from the class instructor and class TA.

If you still have questions, please send an email to:

rroach@nmt.edu

ALL WORK MUST BE YOUR OWN AND YOU MUST CITE ANY CONTRIBUTIONS THAT YOU RECEIVE FROM CLASSMATES OR EXTERNAL SOURCES. All rules in the Academic Honesty Policy strictly apply.

PROJECT SUBMISSION

Project 2 should be submitted online via Canvas.

Please either **tar** or **zip** your submission files and include the course name, your full name, and the project number in the tar/zip file's name.

Example submission file: cse353_project2_JohnSmith.zip

The following items must be submitted:

1. Source code (properly commented) in Java, C#, or C++ that can be compiled and run on the TCC machines.
2. The make file, build.xml, or MSBuild file that compiles your source code from command line.
3. Documentation file describing how to run your program, all features implemented (including EXTRA credits features, if any approved), and your design approach, as well as any caveats in your design.
   Note: Please, **clearly report all the EXTRA credit additions** that you have carried out in your protocol implementation (given that they are preapproved), otherwise **you WILL NOT receive any credit for them.**

Adding any descriptions in the Canvas comment box is optional, but please also place those descriptions in the submission files if they are important since the comment box may be overlooked by the class TA.

REFERENCES

Sockets:

Java
"All About Sockets":
http://docs.oracle.com/javase/tutorial/networking/sockets/index.html

C#
"Sockets": http://msdn.microsoft.com/en-us/library/b6xa24z5.aspx

C++
"Practical C++ Sockets": http://cs.baylor.edu/~donahoo/practical/CSockets/practical/
"Sockets Tutorial": http://www.linuxhowtos.org/C_C++/socket.htm

Multiprocess and multithreads:

Java
"Java Docs": http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html
"Java Threads Tutorial" (5 pages): http://www.javabeginner.com/learn-java/java-

threads-tutorial

C#

"Threading Tutorial": http://msdn.microsoft.com/en-us/library/aa645740(v=vs.71).aspx

C++

"Introduction to C/Unix Multiprocess Programming": http://www.osix.net/modules/article/?id=641

"Threads in C++": http://www.linuxselfhelp.com/HOWTO/C++Programming-HOWTO-18.html

Makefile, build.xml, MSBuild

Java

Make file for Java programs: http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html

"Tutorial: Hello World with Apache Ant": http://ant.apache.org/manual/tutorial-HelloWorldWithAnt.html

C#

"How to: Write a Simple MSBuild Project": http://msdn.microsoft.com/en-us/library/ms171479(v=vs.90).aspx