CSE/IT 353 Project #3

Bridged Token Passing Network (BTPN)

Due: 11:59pm Tuesday, December 2nd

GOAL

The goal of this project is to simulate a transparent bridge between two independent token sub networks, each with an independent, variable number of nodes. The bridge will act as a node on each sub network, producing **(fake)** acknowledgment, buffering frames, and transporting non-local traffic across sub networks. You will be extending your Project 2 implementation to support the insertion of a bridge adapter node into each sub network. Your bridge must obey the protocol rules on each sub network and handle frame buffering to allow for different sub network traffic speeds, and **(fake)** acknowledgment. In addition, the bridge may handle different frame formations or PDU size limits as extra credit.

DESCRIPTION

A common node in both sub networks will represent the bridge. The main function of the bridge is to abstract global transparent traffic at each node (i.e. a node should not be aware if the destination node is local or global to its hosting sub network). The protocol at all nodes except the bridge node will remain the same (except for the extra credit). The project will require synchronization because the bridge will have to operate as an entity on both sub networks, asynchronously. **In your REAME and Project Proposal, you must specify your methodology for synchronizing access to the shared buffers in the bridge.**

As with the previous projects, the implementation is to use TCP/IP sockets and be implemented in Java, C#, or C++. You are encouraged to write your bridge in a different language than your first project in order to gain experience with interfacing between different languages (sockets should make this relatively simple). Please use valid classes and object-oriented structuring, as before. You will extend your first project implementation so that each sub network will run as an independent program. The bridge will be run as a third program to coordinate communications between the previously-initialized, autonomous sub networks, connected through sockets.

After transmitting all data from each node in both sub networks, all of the programs should exit with return 0. This means that the shut down signal should be coordinated through your bridge and/or optionally your monitor node, depending on your Project 2 protocol. Your bridge must deal with buffering frames. It must obey the same protocol as the other nodes in detecting error transmissions and resending.

For this project, your bridge will use pre-established routing tables read from a configuration file (see below). The establishment of unique nodes on each sub network will require some modification of your Project 2. Your program may print to stdout whatever output you find useful in analysis and debugging. You must produce output files at each node just as in your first project. In addition, the bridge must produce a log file.

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. You must include **instructions** on how to use this file in order to compile and run your program from **command line**. If you do not, **I will not grade your project!**
    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. Make sure to specify how to run the three executables (two token rings and the bridge). 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. This helps the TA to understand your file structure, and helps frame the order in which the TA should read the source code files.
    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. Specify your methodology for synchronizing access to the shared buffers in the bridge.
    g. Specify your method for communicating socket numbers between the sub networks and the bridge.
    h. Description of what events are logged by the Bridge.

TECHNICAL SPECIFICATIONS

This section outlines the requirements for each of the programs in the project. The first program will be your token network from project 2, being run with different configurations. The second program will be a bridge.

Command-Line Arguments: The token network program must accept a single command line argument that specifies the configuration file (detailed later).

```
./TokenRing  ringX.conf
```

Note: Disabling node priority is highly recommended. If you allow priority, the bridge should always have the highest priority in each sub network.

There may be up to 254 nodes in each sub network (plus one optional monitor node #0 and the bridge adapter node #255).

The bridge program must accept three arguments that specify the configuration file and the log destination file. You will also have to figure out a way to connect the bridge into each of the token sub networks. You must find a way to communicate port numbers from the token sub networks to the bridge, so the bridge knows which ports to listen on as well as those to connect to. You can communicate these ports through a configuration file or through prompts in the bridge program. Also, assume that the token sub networks have started before the bridge. **Specify the method that you use to communicate socket numbers between the sub networks and the bridge.**

```
./Bridge  ring1.conf  ring2.conf  log.txt
```

Configuration files: There will be a script posted to generate all of the configuration files, as well as the input files for each node in each sub network.

The token network configuration files specify the node numbers in each sub network. The bridge will read these files and construct a routing table from them.

```
ring1.conf
---
2
4
5
8
13
...
===
```

```
Ring2.conf
---
1
3
7
32
```

Input files: These files will be exactly the same as in Project 2. There will be a new script provided to generate the input files and configuration files.

Output file: Output files for each node will be exactly the same as for the first project.

Log file: The bridge must emit a log file that includes the following:

- Timestamp
- Event description

You must define what events you log in your README. Events could be:

- Buffers reaching a certain threshold
- Retransmission due to erroneous frames
- Globally routed frames and any retransmission
- Any resolution of mismatching sub network protocol's semantics/syntax (For extra credit projects)

Grading

We will use randomly-generated input and configuration files for multiple sub networks of differing sizes. The files used for grading will be generated by a script that will be made available to you for testing. You will be provided with our grading script to pre-evaluate your project before submission. The project will be graded according to the following rubric:

Implementation (70):

Make file: _____/5

Correct implementation (it works as specified): _____/55

Useful logging data: _____/10

Documentation (30):

Proposal/Design documentation: _____/15

In code documentation (comments): _____/5

*Documentation of all features in your README: _____/10

*Your README Documentation should include at least:

- Compilation and run instructions
- File descriptions
- Bridge buffer synchronization method

- Method for communicating socket numbers to bridge
- Feature checklist
- List of known bugs
- Description of design for extra credit features that you implemented (if applicable)
- If your program has a long execution time (run time > 3 min) then you must add an estimate of run times for varied system sizes (10 nodes, 25 nodes, 50 nodes, 100 nodes, etc). This will help the TA to distinguish between long run times and programs that fail to terminate!

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

CHECKLIST

Include the following checklist in your documentation to document the status of your code. 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). This checklist is useful to help you complete the protocol submission, as well as declare what you were able to accomplish to the TA which helps in grading.

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
Multiple executables (each subnet has its own executable)	missing
Configuration file loading and parsing	missing
Construct all the networks specified in configuration file	missing
Join the bridge into all the token sub networks	missing
Successful frame buffering in bridge	missing
Bridge successful transports non-local traffic across sub networks	missing
All nodes and networks shut down after all the data has been transmitted	missing
Bridge creates log file	missing
Program loads input file	complete
Program parse input file	missing
Program generate tokens (one for each ring)	partially - token may be missing.
Nodes generate proper frames	complete

Nodes run as autonomous threads	missing
All nodes terminate after all data in the network has been transmitted	partially - some nodes cannot terminate
Program can be executed via command line	missing
Randomly accept a frame (randomly generate the FS)	missing
Extract all fields from a frame	missing
Nodes switch between *listen* and *transmit* states.	missing
Pass the tokens and frames to the next neighbor.	missing
Print to the Output file with correct format.	missing
Documentation is complete (README).	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)**

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. Protocol Mismatching
    a. Different frame formats on each token network: 10%
    For example, sub network 1 has the following frame format:

AC	FC	DA	SA	Size	Data (LLC PDU)	FS

And sub network 2 has the following format:

AC	SA	DA	FC	Size	Data (LLC PDU)	FS

The bridge must translate between these different frame formats. You will need two different executables, one for each sub network. Make sure to document which frame formats your sub networks use in your README.

    b. Different maximum allowable frame size: 10%
    If sub network 1 has a frame size limit X and sub network 2 has a frame size limit of Y, and X > Y, then the bridge will have to fragment long frames from sub network 2 going to sub network 1. Note that this will require the

receiving nodes to reconstruct the fragmented frames before entering them into their output files.

2. Dropping Packets: 20%
   The bridge should have a 'reasonable' buffer limit. Once the buffer is full, the bridge should drop packets. Therefore, sending nodes should set a timeout before waiting for an ACK. If the node does not receive an ACK, it can assume that the bridge is busy. It should queue the frame and try to send the next frame, continuing in this fashion for the rest of the THT. After an exponential backoff, the node should retry sending the queued global frames.

3. Dynamic bridge learning for dynamic network topology: 25%
   Your bridge must build its own routing table and *will not take configuration files*. When the destination node is unknown, it must flood the frame to both sub networks and then drain the frame from the sub network on which it is orphaned. It must put known sources and destinations in its routing table.

4. Exceptional log files: 5%

5. Bridge is implemented in a different programming language than the token rings: 5%

6. The implementation of any ***creative features*** will be rewarded, but you MUST obtain prior authorization from the class TA.

If there are features not specified here, you can fill in the details from the lecture notes on Media Access Control and Bridging.

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 3 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_project3_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