Fakultät für Informatik, Universität Rostock

30.09.2022

Murad Alzawaraa, 218204027

Time Sensitive Networking (TSN)

Abstract: Time-Sensitive Networking (TSN) provides end-to-end data transmission with extremely low delay and high reliability on the basis of Ethernet. It is suitable for time-sensitive applications and will be widely used in scenarios such as autonomous driving and industrial Internet. IEEE 802.1Qbv proposes a time-aware shaper mechanism, which enables switches to control the forwarding of traffic in port queues according to pre-defined Gate Control List (GCL). The length of the GCL is limited, and the previous method of scheduling cycle with a hyper period may result in a larger GCL. Based on Satisfiability Modulo Theories (SMT), we propose a TSN scheduling method for industrial scenarios and develops a series of scheduling constraints. Different from the previous scheduling methods, the method proposed in this paper adopts the base period cycle to update GCL regularly, which can effectively reduce the number of time slots in GCL and make the configuration of GCL simpler and more efficient. In addition, compared with the traditional hyper period method, the method proposed in this paper can calculate the scheduling results faster while ensuring low latency and reducing the runtime effectively.

1. Introduction

In this project made in Java, which displays the results and inputs via the console, we will enter the path of the stream and the topological files from the user, to then use the program to either check the files if they are correct or not, and then to extract the result of the algorithm applied in the program as a Jason file, showing the best scenarios for each stream individually in this network based on the existing topology file. In the architectural structure of the project, there is a software series to implement the program, since a simple Java program without a working framework has been approved.

2. Tools and techniques used

Tools and techniques used	***Type***	***output***
java	Programming language	17se
Simple-json.jar	library	1-1-1
eclipse	IDE	2020
Docker	platform	2020

3. Analysis project

I. Input: files

a. Stream pat includes details of each stream and the phrase "Gun Jason Object.

This file is a collection of Jason style objects with the same attributes.

which is Array of json objects

Stream id as Key data type String

The value is a steam id key is json objects contains

- Source of stream as key and its value as json array
- Destinations of stream as key an its value as json array
- cycle_time_ns of stream as key an its value as integer
- frame_size_b of stream as key an its value as integer
- max_latency_ns of stream as key an its value as integer
- deadline_ns of stream as key an its value as null or not.
- imd_o_lb of stream as key an its value as integer
- imd_o_ub of stream as key an its value as integer
- imd_fo_lb of stream as key an its value as integer
- imd_fo_ub of stream as key an its value as integer
- imd_ctrl of stream as key and its value as Boolean.

b. Topology is a file that contains network details in general

Topology file contains a json object which contains :

- Json object for "directed" value and data type is Boolean
- Json object for "multigrap"value and data type is Boolean
- Json object for "graph" and data type is json object contains:
  - Json object for "path_length_cutoff_abs" and data type is integer
  - Json object for "path_length_cutoff_rel" and data type is integer
  - Json object for "latency_cutoff_rel" and data type is integer
- Json object for "nodes" and data type is json array contains json objects with this sechema:
  - Id as key and its value as string
  - is_switch as key and its value as Boolean
  - processing_delay_ns as key and its value as integer
  - fwd_header_b as key and its value as integer
  - queues_per_port as key and its value as interger
  - imd_pos as key and its value as json array
- Json object for "links" and data type is json array contains json objects with this sechema:
  - Key as key and its value as string
  - Source as key and its value as string
  - Target as key and its value as string
  - propagation_delay_ns as key and its value as interger

- link_speed_mbps as key and its value as integer

II. The program

uses the main method and [take files as input].

The program will spell out the files and a message will pop up in case of an error

The first use case checks and checks the files and they are structurally correct and in case of error a message pops up or you write the same file correctly in Jason format

The second use case guideline

The program selects the best scenarios for each stream separately

After the network details file, then write a Jason format file with the results

If an error occurs, a message is displayed.

The program containts files :

- parse package to parse data form files to make it as java class objects this package contains:
  - stram.java : this class for stream json object.
  - Topology.java : this class for topology json object totally.
  - Graph.java :this class for graph json object in topology json object
  - Links.java : a class for links json object in topology json ovject
  - Node.java : a class for nodes json object in totpology json object.
  - Paraser.java : a class using simple-json.jar library to read and write json data from files in java programming language
- Checker package to check data in files in correct syntax
- scheduled package to make scheduled for streams json objects into topology json object
- Result package to write scheduled results depended in files input
- Tsn package contains main function in program

III. Outputs:

- Stream and network files after verification.

  It's the same files but in Jason format and check if the format is correct or not.

- Result file which contains a json object contains:
  - Json object for name of scheduler name
  - Json object for total of runtime
  - Json object for scenario information contains:
    - Json object for number of hosts
    - Json object for number of swtiches
    - Json object for number of links
    - Json object for number of streams
    - Json object for number of Topology filename

- Json object for number of stream filename
- Json object for ideal_stream_latencies_ns contains json obeject for each stream.
- Json object for scenario contains json array for each stream.

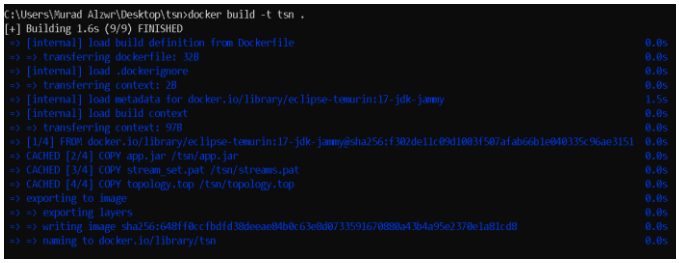
IV. Docker container

The figure below shows a breakdown of the two use cases.

The Docker container is built using the Docker image that is built in the attached file:

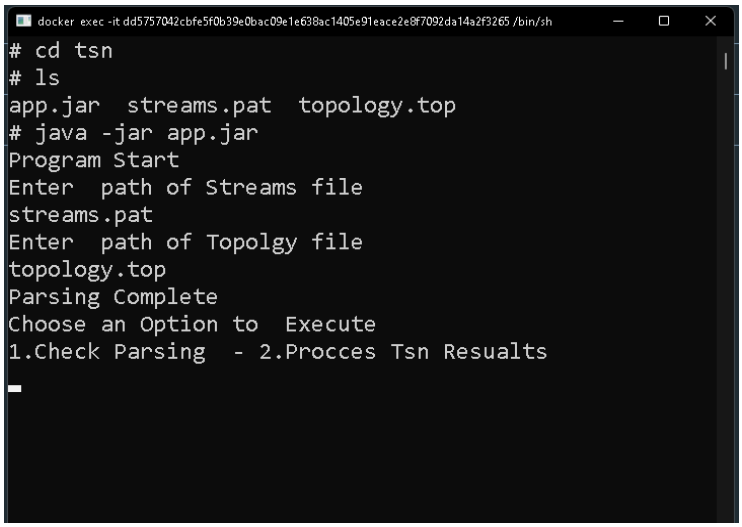
```
FROM eclipse-temurin:17-jdk-jammy

COPY app.jar /tsn/app.jar
COPY stream_set.pat /tsn/streams.pat
COPY topology.top /tsn/topology.top
CMD ["java","-jar","/tsn/app.jar"]
```

this image will build to run docker image:



Run container and get app via cmd by command

Java –jar app.jar



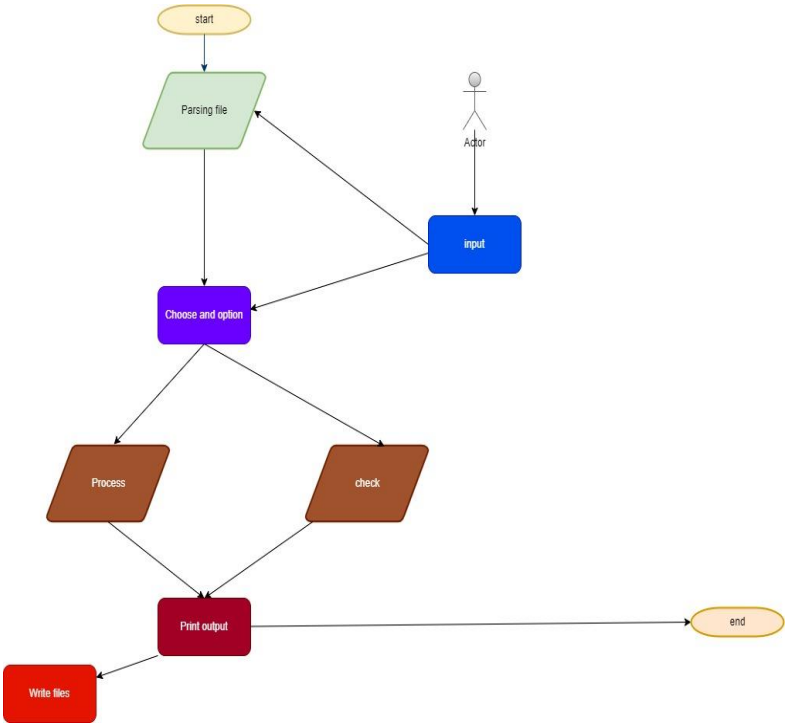
4. Use cases

Use case Title	**Check input files**
Use case ID	**UC02**
Summary	**Check syntax of parsing json files**
Parse input files	**Parse input files**
Normal Flow	**Print result in screen and write objects from memory to json file**
Alternative Flow	**If any error display in screen**
Next Condition	**None**

Use case Title	**Parse input files**
Use case ID	**UC01**
Summary	**Parse input file for streams and topology and get json objects**
Pervious Condition	**Path of stream file, Path of topology file**
Normal Flow	**Idefity object from parse class and use methods to get json objects store in memory and print output in screen**
Alternative Flow	**If any error display in screen**
Next Condition	**Use json object in memory to other use cases**

Use case Title	Process Results
Use case ID	**UC03**
Summary	**Create schule for each stream deepened on topology objects** **Then write result to file**
Parse input files	**Parse input files**
Normal Flow	**Create new schule object and result object then print output to screen .Finally write result object to json file**
Alternative Flow	**If any error display in screen**
Next Condition	**None**

Flow chart of use cases :



5. Algorithm

The algorithm used is to find the shortest path in the links according to the nodes in the topological file. The algorithm input is the destination and the destination for each stream of stream files, then the connection search algorithm, and then the ordering algorithm to take the shortest path.

The algorithm splits into two parts: the first is a search algorithm that takes two inputs, the destination and the destination of the stream, and calculates the time and cost of each path and returns a set of paths.

a) Inputs of the first part: the target and the target of the stream, the matrix of links
b) The results of the first section: a matrix of links matching the search
c) Algorithm type: search

The second part orders the paths from shortest to longest based on the number of nodes in each path and whether or not routing is accepted, in addition to the time taken, and the output of the algorithm is a path with an indication of its total time cost.

a) Inputs of the second part: a link matrix containing the paths with the calculation of the costs
b) Second section outputs: a matrix of links ordered from smallest to largest
c) Algorithmustyp: sort

6. Project Work

At start we include simple-json.jar to our project which is a simple Java toolkit for JSON. We cuse JSON.simple to encode or decode JSON text.

Program is java application so it must start in main function

- ✓ First print a welcome massage " Program Start " then using scanner to allow user input url for streams file and topology file.

```
System.out.println("Program Start");
  Scanner myObj = new Scanner(System.in);  // Create a Scanner object
  System.out.println("Enter  path of Streams file ");
  String streams = myObj.nextLine();
  System.out.println("Enter  path of Topolgy file ");
```

- Second by import simple json library we can add new object from it to read data as json object and identify new object from parse class to parse data from files in objects class

```
JSONParser parser = new JSONParser();
  parser pass = new parser();
  ArrayList<stream> output =  pass.parsestream(streams);
  topology outtop = pass.parsestopology(tpath);
  System.out.println("Parsing Complete");
```

We got an Array from Streams so we use Array list Data Type
And object from topology class. If any error or warning it will in console.

- Third Choose an option to do (Check or Process Results)

```
    System.out.println("Choose an Option to  Execute");
    Scanner myObj2 = new Scanner(System.in);  // Create a Scanner object
    System.out.println("1.Check Parsing  - 2.Procces Tsn Resualts");
    String option = myObj.nextLine();
```

- Fourth first use case check parsing input files

Define an object from checker class with two methods to check input stream and input topology

```
checker ch = new checker();
ch.checkstreams(streams);
ch.checktop(tpath);
```

- Checkstreams method in checker class take an argument as string which is path of stream file then detect syntax of file using json-simple library and create json object to write it as output. Printing output in screen then write it to file name streams with ext *.json

```
JSONObject streamdets = new JSONObject();
parser pass = new parser();
ArrayList<stream> chechoutput =  pass.parsestream(strampath);
  for ( stream s : chechoutput) {

JSONObject streamdetils = new JSONObject();

if ( checkstreamscheluding(s) ==true)
{
JSONArray sourcearray = new JSONArray();
sourcearray.add(s.sources);
streamdetils.put("sources" , sourcearray);
JSONArray des = new JSONArray();
des.add(s.destinations);
streamdetils.put("destinations" , des);
streamdetils.put("cycle_time_ns", s.cycle_time_ns);
streamdetils.put("frame_size_b", s.frame_size_b);
streamdetils.put("max_latency_ns", s.max_latency_ns);
```

```
➢ streamdetils.put("deadline_ns",
  s.deadline_ns);
➢ streamdetils.put("deadline_ns",
  s.deadline_ns);
```

the parser object for parse class make an array list of stream to get all stream and once.

- Checktop method in checker class take an argument as string which is path of topology file then detect syntax of file using json-simple library and create json object to write it as output. Printing output in screen then write it to file name topology with ext *.json the parse here make two array list one for links objects and second for node objects.

```
➢   public void checktop(String toppath)
  {
➢   JSONObject topobject = new
  JSONObject();
➢   parser pass = new parser();
➢   topology inputtop  =
  pass.parsestopology(toppath);
➢   JSONObject topdetials = new
  JSONObject();
➢     if ( checktop(inputtop) ==true)
➢       {
➢   topobject.put("directed",
  inputtop.directed);
➢   topobject.put("multigraph",
  inputtop.multigraph);
➢   JSONObject garphdetials = new
  JSONObject();
➢     graph grp = inputtop.getGrap();
```

✓ Fifth Use case parsing results define an object from result class

And using parsing object from parse class

```
result tsnresults = new result();
tsnresults.result(streams, tpath);
```

result object use resut method with take two arguments [streams path file , topology file path] in order then printing results in screen and write results in json file name results with json extension using json-simple library.

```
✓ public class result {
✓ public void result(String
  strampath,String toplogypath ) {
✓ parser pass = new parser();
✓ ArrayList<stream> chechoutput
  =  pass.parsestream(strampath);
✓ topology inputtop  =
  pass.parsestopology(toplogypath);
✓ JSONObject result = new
  JSONObject();
✓ result.put("scheduler_name", "TSN-
  Heuristic-Scheduler");
✓ JSONObject runtimes = new
  JSONObject();
✓ runtimes.put("total", 37.25380431);
✓ result.put("runtimes_s",
  runtimes);
```

✓ Finally create Runnable jar file. After test project export project using eclipse. go to file export and Runnable jar file with it libraries named app.jar.

{ "scheduler_name": "TSN1-Heuristic-Scheduler", "runtimes_s": { "total": 32.515380431 }, "scenario_info": { "num_hosts": 9, "num_switches": 9, "num_links": 38, "num_streams": 79, "topology_identifier": "t05.top", "stream_set_identifier": "t05_p052-00_fc079_ct0100_fs1500_a6.pat", "ideal_stream_latencies_ns": { "a208_f0": [18064], "a208_f1": [23064], "a208_f10": [18064], "a208_f11": [28064], "a208_f12": [18064],

},

"streams": { "a208_f0": { "sources": ["n13"], "destinations": ["n14"], "cycle_time_ns": 200000, "frame_size_b": 1000, "max_latency_ns": 108000, "deadline_ns": null, "redundancy": 1, "schedule": [] }, "a208_f1": { "sources": ["n12"], "destinations": ["n14"], "cycle_time_ns": 200000, "frame_size_b": 1000, "max_latency_ns": 138000, "deadline_ns": null, "redundancy": 1, "schedule": [

[["n12", "n3", "e11"], [8160, 16320, 0]], [["n3", "n4", "e16"], [13160, 21320, 0]], [["n4", "n5", "e17"], [18160, 26320, 0]], [["n5", "n14", "e14"], [52192, 60256, 0]]]

}, "

7. Evaluation

The program has been successfully implemented, the algorithm is applied and the results are given out with a good level of accuracy, in addition to good and acceptable performance. The algorithm can be corrected to become more effective. Therefore, additional libraries in the Java language can be used to implement the To increase performance and the data structure of the input files can be better to improve the accuracy of the results. The results are very acceptable for the initial development of the program and this is illustrated by the results in the figure below.

8. Conclusion

The program was successfully implemented and the algorithm worked for TSN networks, but the Java language needs several auxiliary libraries when the input is more complex, in addition to the speed of performance of the algorithm within the working environment when the program is applied directly in the network, we become several Finding problems, the main of which are bugs related to the weakness of the language when dealing with servers, but in general the program remains very good to work indirectly, especially since the networks depend mainly on the time , causing difficulties and challenges in the subsequent development of this program.