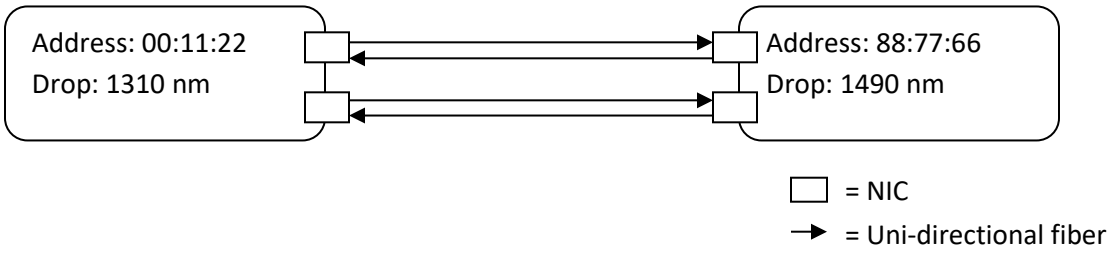
18-756 – Project 1 - SONET

Overview

You have been given some Java code that represents an Optical network. Your job is to complete the Optical network. The SONET network in the code given to you (and described in this document) does not work exactly like the real SONET system as that would take you around 6 months to implement.

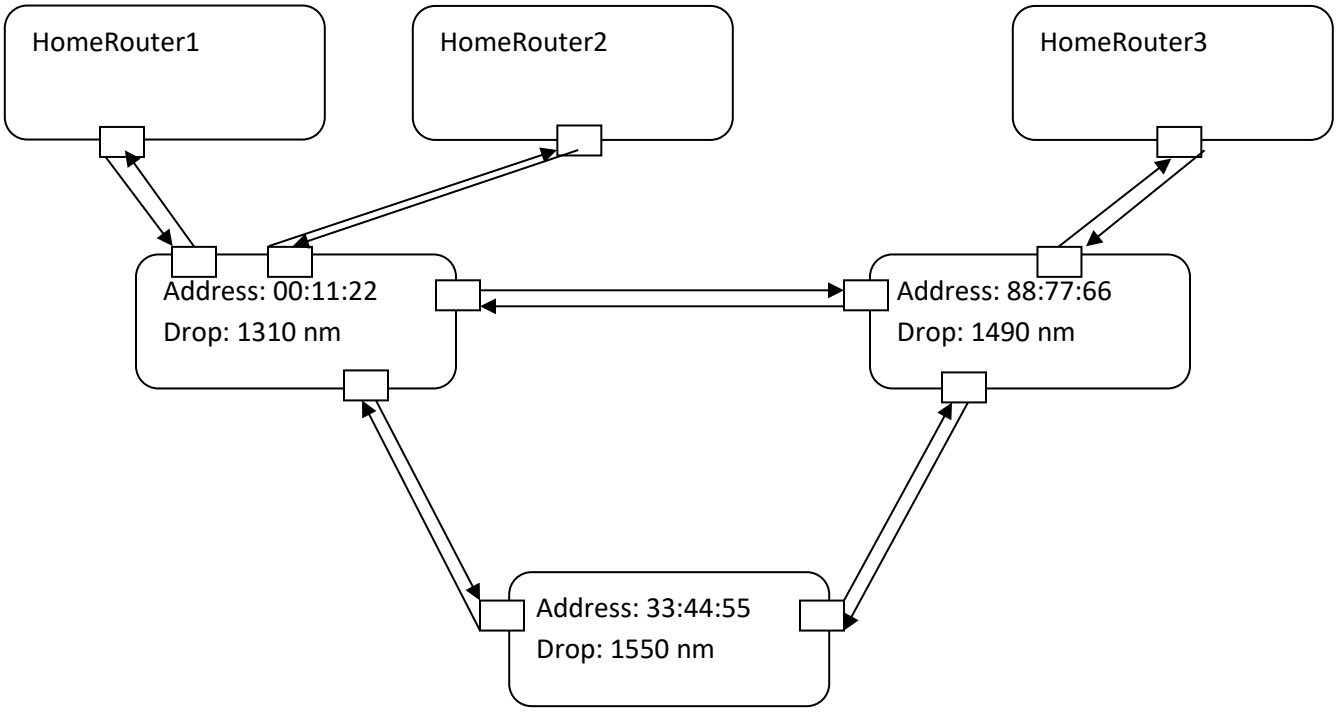
The first thing to do is look at Topology.java. This file creates a basic Optical network consisting of two SONET DXCs.



Our implementation of a SONET DXC uses a FDM system to decide where data is sent, which allows us to use wavelengths/frequencies like addresses. Many SONET routers really use TDM and WDM to send data, however our SONET network does not constantly send data, so we are using FDM to make our implementation simpler. Each DXC has one (or more) drop frequencies. If a SONET DXC receives a packet on a wavelength that matches one of its drop frequencies, it removes that packet from the ring. If a SONET DXC receives a packet on a wavelength that is not on one of its drop frequencies, it forwards that packet around the ring. Other SONET DXCs in the network must know the wavelength that the destination DXC is dropping on to be able to send data to it. These wavelengths are configured manually by the SONET administrator. The SONET DXCs also have restoration feature, which should be implemented as UPSR.

Another feature of our SONET network DXC is that three STS-1 packets from the same source NIC will be multiplexed into a single SONET STS-3 packet. Our SONET DXC does not send SONET STS-3 frames unless there is data to be sent. If there are not enough STS-1 frames, SONET DXC will insert “place-holder” STS-1 frames to be sent along with real data STS-1 frames. The DXC will have only two links that it can transmit data on due to the UPSR configuration. Routing should be done according to the rules of UPSRs.

The following figure shows three DXCs forming a ring network. Two of them also connect to home routers. These home routers connect to the SONET DXCs through STS-1 links. SONET DXCs connect to each other through STS-3 links, which is the multiplex of 3 STS-1 links (this aggregation is a feature you will need to implement).



When answering the questions, please provide a small description and explanation of your code on the answer sheet. Also, make clear in your source code where you have changed things, and comment your source code so that it is clear what you are doing and why. We want to give you points but can only do so if you make it clear that you have answered the question.

To complete each question in the coursework you must have completed the question before it, otherwise your code will not function correctly. Make sure your submitted code runs, no matter what questions you reached in your attempt. If the code doesn't run, you will get zero points. Also, if you do not complete the entire coursework, please indicate clearly where you answered up to.

Grading

Implementation 80 points: *How well did you implement you answers? Did your code work correctly? Was it implemented efficiently? Did you understand and implement what was being asked?*

Report 20 Points: *Was your report clear and concise? Did you report help the TA's understand your work? Did you answer questions satisfactorily? Did you explain where and why you added/altered code?*

Deliverables

1) All the Java source files only (src folder in eclipse). Delete the .class files from the working folder. (In eclipse this is the bin folder which at the time of submitting needs to be empty). Call the folder 'andrewid_18756_project1' (use **your** AndrewID, not the word 'andrewid').

2) A project report with all the answers and code explanation, named as andrewid_project1_report.doc/pdf, to be placed in the project folder (the folder containing the src and bin folders).

Zip the working folder into a ZIP file only. Name the file as andrewid_18756_project1.zip (again using **your** Andrewid in replace of 'andrewid').

Upload the zip file in the Assignments section for Project 1 on canvas.

Getting started for Java newbie

The following instructions detail the installation of Eclipse in a Windows environment. You can use other environments as long as they are compatible with Eclipse and you provide the corresponding Java files. Use of other IDEs is also fine, though it is up to you to ensure that they are installed and configured correctly. If you already have an installed version of JRE, please make sure it is updated to the latest version by NOT skipping step 1.

1. Go to http://www.oracle.com/technetwork/java/javase/downloads/index.html and download the latest version of Java Standard Edition (Java SE) JRE. At this time of writing, it should be "Java SE 17". Note the architecture of the platform that you choose (either 32-bit or 64-bit), since you will need to download the same architectural version of Eclipse.

2. Install the JDK.

3. Go to https://www.eclipse.org/downloads/packages/ to download the latest version of "Eclipse IDE for Java Developers". Download the version whose architecture matches the JRE architecture which you downloaded above.

4. Extract the downloaded Eclipse software, which should be downloaded as a `.zip` file, to your directory of choice. Eclipse is then essentially installed in that directory.

5. Add the "`<Java installation directory>\bin`" to your system PATH environment variable. The directory should look like "`C:\Program Files\Java\jre17\bin`". For more information on how to do this, visit http://www.java.com/en/download/help/path.xml

6. Download and extract the project 1 files onto your local computer. Find the directory "`andrewid_18756_project1`" and note its full system path.

7. Navigate to your Eclipse directory and execute "`eclipse.exe`". If any errors about Java JRE are reported, then you may have installed Eclipse which has a different architecture from your JRE installation, or you may have not configured your system PATH properly.

8. If Eclipse executes successfully, you should be shown a fancy "Welcome" screen. (You may also be asked to select a default workspace.) This is not what we need, so locate the "Welcome" tab in the upper left-hand corner of the application and click on the "X" button to close the "Welcome" screen.

9. Go to "File" > "New" > "Java Project".

10. Under project name, enter "`andrewid_18756_project1`", without quotes.

11. Uncheck the option "Use default location", and instead manually set the "Location" field to the full system path of the "`andrewid_18756_project1`" directory in step 6 or browse to locate the path to that directory.

12. Leave everything else in their default settings and click "Finish".

13. Under Package Explorer, expand the "`andrewid_18756_project1`" root project.

14. Navigate to "src" > "(default package)".

15. Right-click on " Topology.java", and move down to "Run As", and click on "1 Java Application".

16. You should see the text "Setting up two DXCs" in the console window.

Code setup

The code is setup to help you answer questions, although at first it may seem confusing. You are free to edit any of the files you want with one caveat. You will find that the Topology file has already defined some methods. **Those methods' names should not be changed, as we will use them to test your implementation in our test cases**. The implementation of the methods, however, is up to you.

Last point: **DO NOT HARD CODE**
Our test code will not be the same as your answer code, so hard coding DXC addresses to go to certain NICs will not work. Use the functions to setup the information during the creation of the network.

Questions

1. [**30pt**]

1.a) The extract below (from Topology.java) tries to send a new STS1 to 88:77:66 from 00:11:22

```
DXC1.create(new STS1Packet("", 1490));
```

The command above will create an STS-1 packet and store it inside the buffer. On the next time tock, the DXC will retrieve it to form a new STS-3 packet. It will then transmit it to both upper and lower DXCs per the rules of UPSR operation.

The processing method does not contain any code. Fill in the method with the following clarifications:

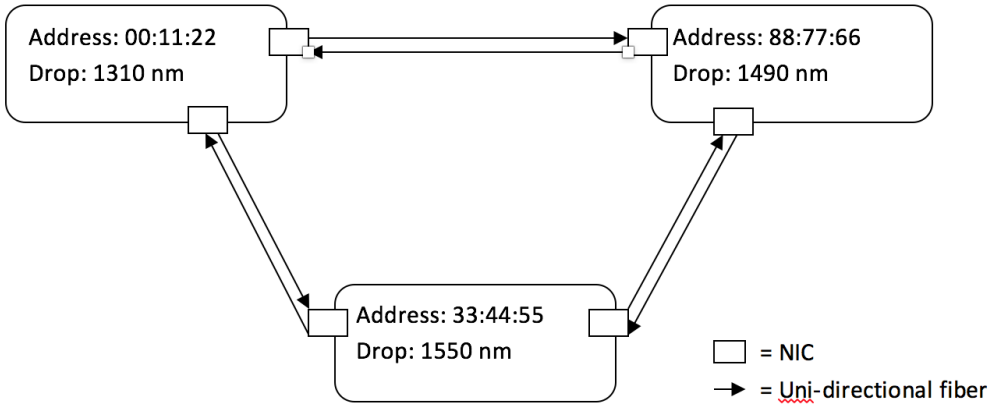
Note: the term frequency and wavelength are used interchangeably across the code, though they mean the same thing.

- If the STS-1 packet inside the STS-3 packet is on the DXC's drop frequency AND is also contained in the list of DXC destination frequencies, the STS-1 packet is forwarded to the `sink()` method. In real life the `sink` method would be sending the data to the layer above.
- If the STS-1 packet has a destination frequency which is not part of any of the destination frequencies listed for all DXCs, sink the packet in that case as well.
- If the STS-1 is not on the DXC's drop frequency, the STS-3 will be unpacked and the STS-1 packets inside will be stored in a buffer, waiting for next call `sendPackets()`. The `sendPackets()` call will frame the STS-1 packets from the buffer into a new STS-3 packet. This new STS-3 packet will then be sent out by `sendRingPacket()`.
- If an STS-1 packet is from the HomeRouter, place it in the buffer and send it on the next clock cycle. You can wait to add this feature during question 3 as it is not necessary for questions 1 and 2.

1.b) The UPSR has the following features:

- When a DXC is the source of an STS-1 packet (label field NIC in the packet as null), it will send the newly framed STS-3 packet containing the STS-1 packet on both the clockwise and anticlockwise links of the UPSR.
- When an STS-3 packet is received from somewhere else, the DXC will unpack it, retrieve the contained STS-1 packets, and update each packet's NIC field to the NIC they were received on. At the next clock cycle, it will try to find 3 STS-1 packets from the same NIC and form a new STS-3 packet. It will transmit this new STS-3 packet in the direction following the UPSR direction that its component STS-1 packets were received on (so clockwise flowing packets will remain clockwise, anticlockwise will remain anticlockwise).
- Supplement for the second rule. If the DXC has a broken link such that it cannot transmit packets in the normal flow direction, it will send the newly formed STS-3 packet in the opposite direction.

Why does the STS-3 arrive at 88:77:66 twice? Why do we want this to happen?



1.c) You may notice that when you send a STS1 from 00:11:22 to 88:77:66 the delay is zero.

```
(SONETDXC) 88:77:66 has received a STS1 packet at the sink on
wavelength 1490 [,,Delay: 0]
```

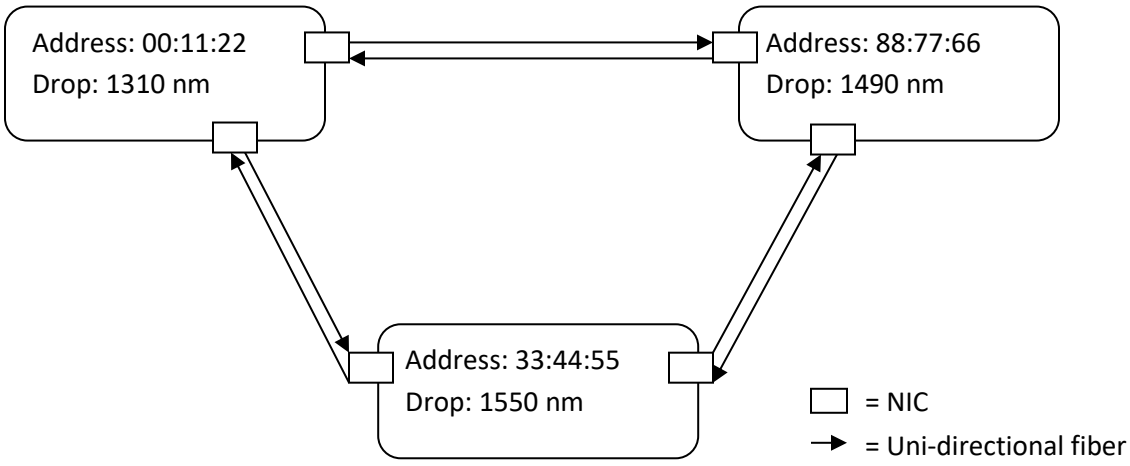
Edit the OtoOLink (and the STS3 and STS1 objects) to add delay of 1 every time they travel over a link. Since STS-3 will be unpacked when received, the delay will always be 1 if the link has constant distance.

NOTE: SONET STS-3 frames and STS-1 frames do not have a field for delay in real life. The actual delay is a property of the speed of light and the length of the transmission line. That said, we will add a fake delay for simulation purposes since we do not have a 10km optical fiber to send our STS-3 packets along.

1.d) What happens if you send data on a frequency that neither DXC 1 or DXC 2 is dropping on? Fix this problem so that the data of unknown frequencies are never sent. Edit your code so that it checks STS-3 packets to ensure we have the frequency of the STS-3 being sent in our destination frequencies object.

2. [**30pt**]

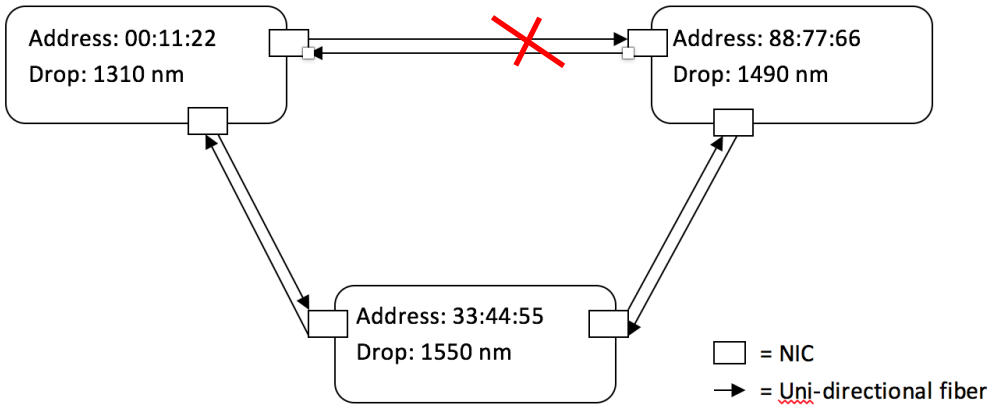
2.a) Now that we have two DXCs working, we want to create a SONET ring. Create the topology below in q2a.java. You can copy most of the code from Topology.java. Make sure you add the new destination and frequency in each DXCs routing table.



2.b) Now, send a new STS-1 from 00:11:22 to 88:77:66. Why (if you have implemented the code correctly so far) do you get the following output? Explain the delay times and why the packet is received twice.

```
(SONETDXC) 88:77:66 has received a STS1 packet at the sink on wavelength 1490
      [,,delay: 2]
(SONETDXC) 88:77:66 has received a STS1 packet at the sink on wavelength 1490
      [,,delay: 1]
```

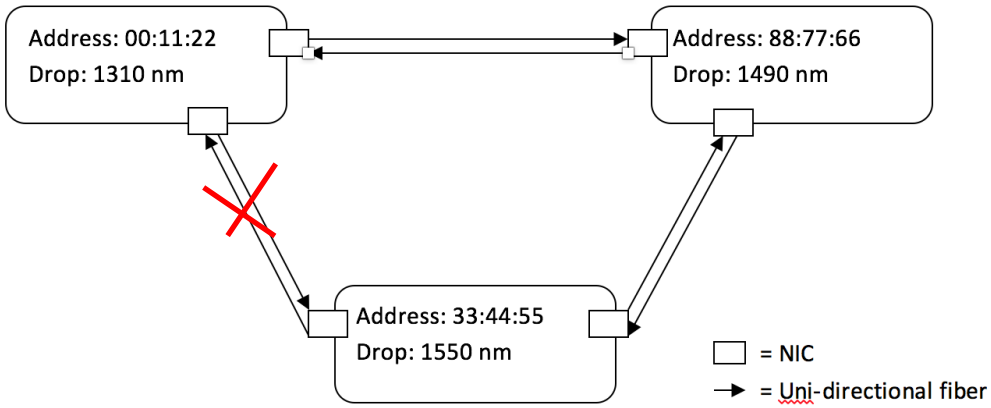
2.c) We now want to do some test on UPSR, send a new STS1 from 00:11:22 to 88:77:66.



If your implementation on the 1b) is correct, you should have the following output.

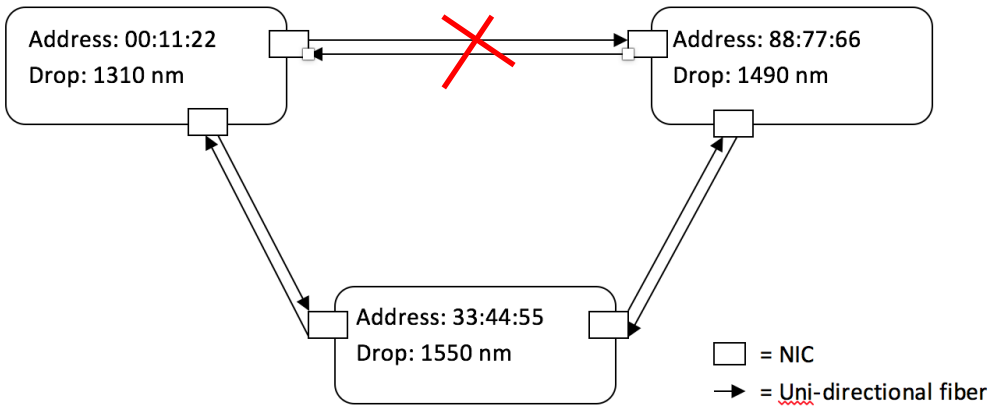
```
(SONETDXC) 88:77:66 has received a STS1 packet at the sink on wavelength 1490
      [,,delay: 2]
```

2.d) Now, send a new STS1 from 00:11:22 to 88:77:66.




```
(SONETDXC) 88:77:66 has received a STS1 packet at the sink on wavelength 1490
      [,,delay: 1]
```

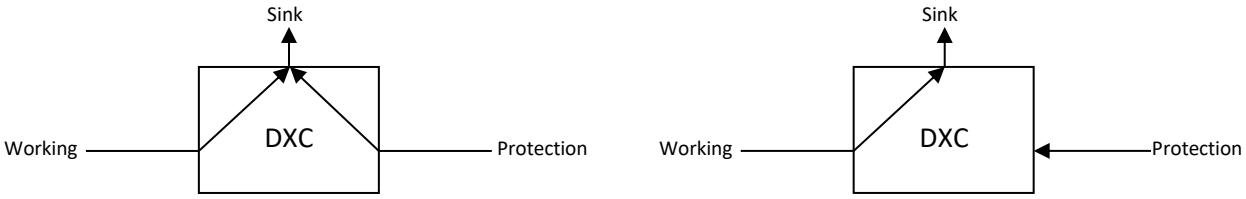
2.e) Now, send a new STS1 from **33:44:55 to 00:11:22.**




```
(SONETDXC) 00:11:22 has received a STS1 packet at the sink on wavelength 1310
      [,,delay: 1]
(SONETDXC) 00:11:22 has received a STS1 packet at the sink on wavelength 1310
      [,,delay: 3]
```

You can test how you UPSR is performing by using the OtoOLink.cutLink method.

There are some additional factors you should consider. For example, the sink should only receive one copy of the STS-1 frame, where currently your implementation acts like the following left picture.



You may want to edit the receivePackets function to use a preferred NIC for a signal. However, be careful if that link goes down as you need to know how to use the protection NIC to listen for the STS-3 frames.

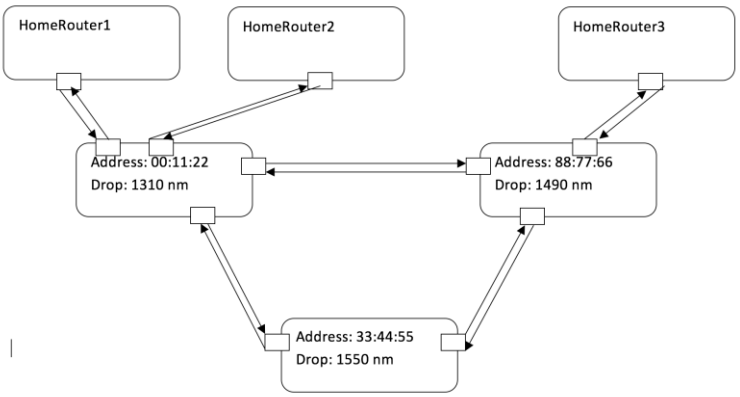
Summary

1) Send in both directions if it is the DXC is the source of the packet

2) Take the receiving direction path when available if the DXC is forwarding the packet
3) The DXC should receive each STS-1 frame twice if there are two **full** distinct paths available

You are free to implement the UPSR however you wish if it conforms to the instructions above (and is not hardcoded to only work with the exact 3 DXCs you have created. Additionally, since this is not really SONET and we don't really have STS-3 frames constantly streaming even when there is no data to be sent, you can create an STS-1 packet and send it if you need to send data from one DXC to another (in real life there is always STS-3 frames being sent so this isn't a problem, you can always send your signaling data – though obviously not down a link that has been cut).

3. [**20pt**]



3.a) Now you have set up the SONET network successfully. We will use it to process packets from our HomeRouters. Since our SONET network is using STS-3, for simplicity we will have at most 3 HomeRouters connected to the SONET DXC ring, each through an STS-1 network (only STS-1 packets will flow between the HomeRouter and DXC).

You will need to implement the sending methods that will send data from HomeRouter to DXC and between DXCs. You do not need to receive packets at Router end.

Our test case will manually define the topology for the simulation's SONET DXCs and HomeRouters. We will then invoke the sending method from HomeRouter (not DXC) to place packets on the network. Refer to the comments in Topology.java for how we will test this part. You can use the topology above to test your implementation.

3.b) The DXC has a constant transmission rate, which means it also has a constant bandwidth. Thus, we assume it can only send 5-character payload in each STS-1 packet. In the real world, you will need to know the actual payload length and how it is determined (it's also important for this course ☺). A DXC will aggregate three STS-1 packets from the same NIC into a single STS-3 packet. We must ensure that all aggregated STS-1 packets come from the same NIC so that we can guarantee they are transmitted along the correct UPSR direction.

Use null STS-1 frame as place holders if there are fewer than 3 STS-1 frames. You also need to implement segmentation for any payload larger than 5- character into multiple STS-1 frames.