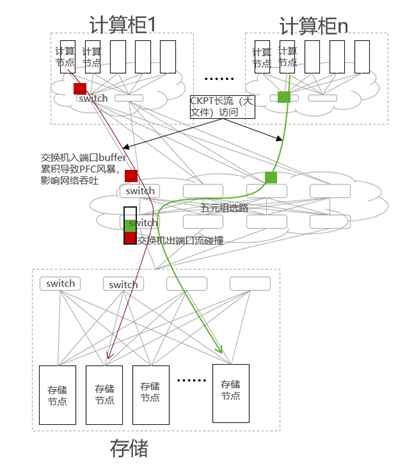
赛题二补充说明

## 一、赛题问题补充说明

**背景**：AI训练过程需要耗费大量的资源，通常在大规模的数据中心中进行。大规模的数据中心通常包含两类节点，一类计算节点（compute node），用来进行计算任务，一类存储节点（storage node)，用来存储数据。计算节点和存储节点之间通过多层交换机组网构成。如图片1所示，计算节点与存储节点中间由多层交换机（switch）进行互联。



图片 1

**一次存储访问**：在AI训练过程中会产生生大量的存储并行访问。我们将一个计算节点对一个存储节点的访问称为一次存储访问（即，计算节点访问存储节点）。一次存储访问由三个要素组成，即计算节点、存储节点、以及访问文件的大小。注意，此处可以是一个计算节点向一个存储节点写文件，也可以是一个计算节点从一个存储节点读文件。

**一次存储访问的路径**：一次存储访问中间会经过很多个路由器，所以访问路径多种多样。我们将一次存储访问的路径记录为(计算节点，路由器1，路由器2，...，存储节点）。

**网络碰撞与拥塞**：由于在大规模AI训练中会产生大规模的存储并行访问，如果不对每一个存储访问的路径进行合理的规划，那么可能会造成不同的存储访问的路径在某些路由器重合。例如，图片1中，红色路径就与绿色路径在一个路由器重合了，这个现象我们称为网络碰撞。网络碰撞会使得发生碰撞的路由器称为系统性能的瓶颈，大量的网络碰撞会使得网络中出现拥塞现象。

**我们的目标**：优化AI训练过程的网络吞吐性能，减少存储访问的时间。

## 仿真平台

本赛题通过开源NS3仿真平台进行存储访问的仿真。下面首先简要介绍NS3仿真平台，然后介绍如何搭建仿真环境。

### NS3仿真平台

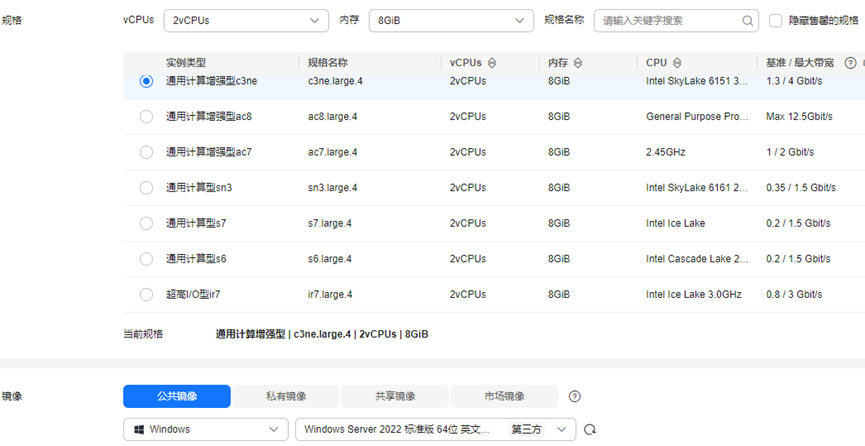
NS3是一个开源的可以用来模拟网络传输的模拟器，其代码发布在github平台，链接如下：<https://github.com/bobzhuyb/ns3-rdma>

在进行模拟时，ns3模拟器读入一个配置文件config.txt（其中包含了网络拓扑的说明，以及存储访问的说明），进行传输模拟，最终生成一个名为mix.tr的日志文件，以及运行时间。具体文件的配置我们在后面的章节进行说明。

### 搭建仿真环境

#### 2.1 华为云服务器准备

推荐配置截图如下：

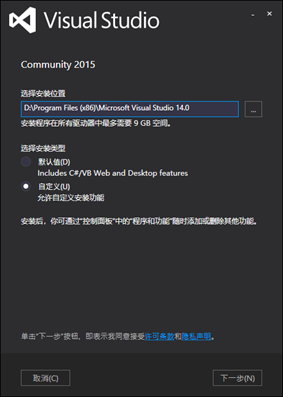


#### 2.2 Visual Studio 2015

下载地址：<https://go.microsoft.com/fwlink/?LinkId=615448>

安装过程如下：

* 打开程序选择安装路径，这里选择自定义安装。



* 根据自己的需求选择安装内容



* 安装完成。

#### 2.3 git相关工具的安装

参考<https://blog.csdn.net/lyorz/article/details/127424055>配置git和github

#### 2.4 NS3代码获取

用2.3中下载配置好的git工具运行下列命令：

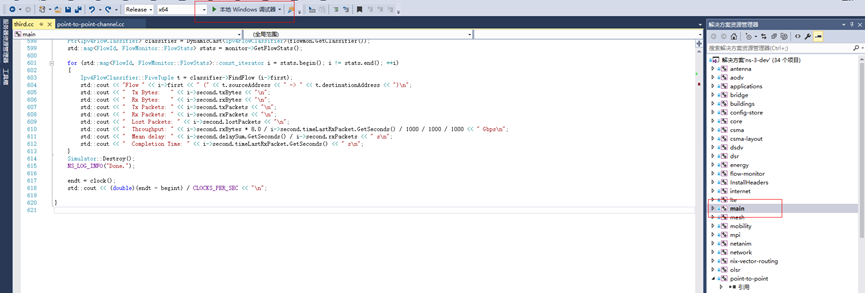
git clone git@github.com:bobzhuyb/ns3-rdma.git

#### 2.5 NS3编译

用Visual Studio 2015打开ns3-rdma工程中的下列文件

windows/ns-3-dev/ns-3-dev.sln

如下图所示，点击“本地windows调试器”即可编译完成。



#### 2.6 运行NS3自带的一个样例

用git工具打开ns3-rdma项目的文件夹，运行下列命令：

cd ns3-rdma\windows\ns-3-dev\x64\Release

./main.exe mix/config.txt

仿真结束之后，我们可以看到生成了mix.tr日志文件。

## 如何用NS3进行存储访问的仿真？

本章节以一个小规模集群为例，说明如何使用NS3进行仿真。

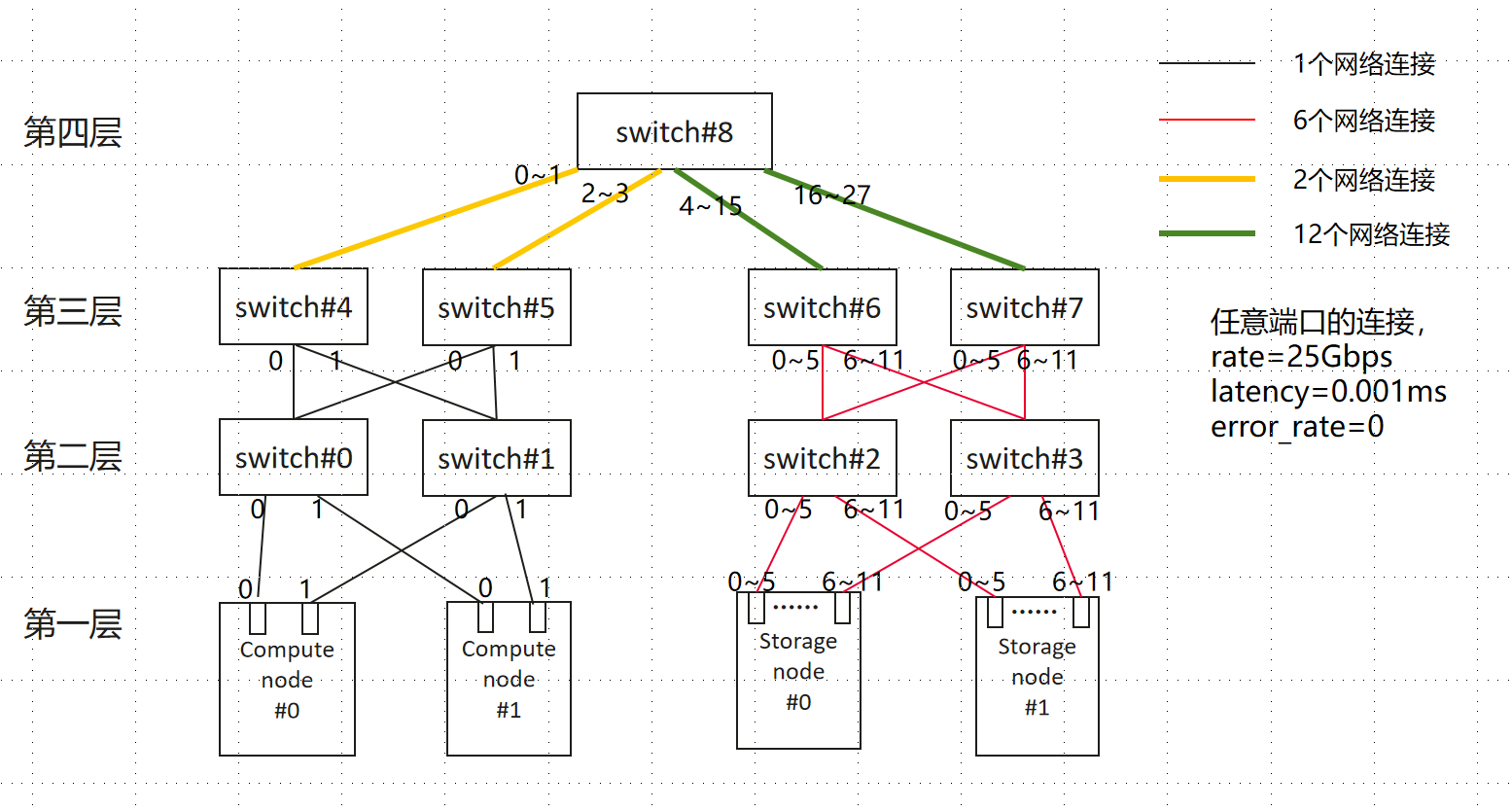
### 1、输入文件

我们需要三个输入文件：(1) topology.txt; (2) flow.txt; (3) trace.txt。下面依次对这三个文件的准备进行说明。

#### 1.1 topology.txt

topology.txt文件描述了集群的网络架构。下面我们给出一个简单的网络拓扑图以及其对应的topology.txt。

**网络拓扑图**：下图所示为一个4层的网络拓扑结构。从下往上依次为第一层、第二层、第三层、第四层。其中第一层包含2个compute node, 2个storage node; 第二层包含4个switch，第三层包含4个switch，第四层包含1个switch。



**节点说明：**

* Compute node**。**每个Compute node有2个端口（0~1）。
* Storage node。每个Storage node有12个端口（0~11）。
* Switch。每个Switch有32个端口（0~31）。

**连接关系说明**：

* Compute node - Switch。每个Compute node的两个端口分别连接到两个不同的switch中。例如，Compute node #0 的端口0连接到Switch #0, 端口1连接到Switch #1。
* Storage node - Switch。每个Storage node的端口每6个为一组，连接到一个siwtch；另外一组连接到另外一个switch。例如，Storage node #0 的端口0~5连接到Switch #2, 端口6~11连接到switch #3。
* Switch - Switch。
  + Switch #0 和 Switch #1。使用4个端口，分别连向Compute node #0, Compute node #1, Switch #4, 和 Switch #5。
  + Switch #2 和 Switch #3。使用24个端口，每6个为一组，共4组，分别连向Storage node #0, Storage node #1, Switch #6, 和Switch #7。
  + Switch #4和Switch #5。使用4个端口，其中有2个端口分别连向Switch #0 和Switch #1， 另外2个端口连向Switch #8。
  + Switch #6 和Switch #7。使用24个端口。其中6个端口连向Switch#2， 6个连向Switch #3，12个连向Switch #8。
  + Switch #8。使用28个端口。2个连向Switch #4, 2个连向Switch #5， 12个连向Switch #6， 12个连向Switch #7。

**生成topology.txt**:

* **第一步**：对Compute Node的端口， Storage Node的端口，以及所有的交换机进行全局统一编号。我们从下往上进行编号，如下面的表格所示：

|  |  |
| --- | --- |
| 名称 | 全局统一编号 |
| Compute Node #0 端口0 | 0 |
| Compute Node #0 端口1 | 1 |
| Compute Node #1 端口0 | 2 |
| Compute Node #1 端口1 | 3 |
| Storage Node #0 端口0~11 | 4~15 |
| Storage Node #1 端口0~11 | 16~27 |
| Switch #0~#8 | 28~36 |

注意：Switch的端口不参与全局编号。

* **第二步**：构建topology.txt。构建规则如下
  + 第一行：#编号总数量 #Switch的数量 #连接的数量
  + 第二行：Switch的全局统一编号
  + 第三行开始的每一行：#编号1 #编号2 #rate #delay #error\_rate
    - 描述#编号1和#编号2的连接的网络传输速率rate, 延时delay, 以及错误率error\_rate

上述例子构建的topology.txt如下

37 9 82

28 29 30 31 32 33 34 35 36

0 28 25Gbps 0.001ms 0

1 29 25Gbps 0.001ms 0

2 28 25Gbps 0.001ms 0

3 29 25Gbps 0.001ms 0

4 30 25Gbps 0.001ms 0

5 30 25Gbps 0.001ms 0

6 30 25Gbps 0.001ms 0

7 30 25Gbps 0.001ms 0

8 30 25Gbps 0.001ms 0

9 30 25Gbps 0.001ms 0

10 31 25Gbps 0.001ms 0

11 31 25Gbps 0.001ms 0

12 31 25Gbps 0.001ms 0

13 31 25Gbps 0.001ms 0

14 31 25Gbps 0.001ms 0

15 31 25Gbps 0.001ms 0

16 30 25Gbps 0.001ms 0

17 30 25Gbps 0.001ms 0

18 30 25Gbps 0.001ms 0

19 30 25Gbps 0.001ms 0

20 30 25Gbps 0.001ms 0

21 30 25Gbps 0.001ms 0

22 31 25Gbps 0.001ms 0

23 31 25Gbps 0.001ms 0

24 31 25Gbps 0.001ms 0

25 31 25Gbps 0.001ms 0

26 31 25Gbps 0.001ms 0

27 31 25Gbps 0.001ms 0

28 32 25Gbps 0.001ms 0

28 33 25Gbps 0.001ms 0

29 32 25Gbps 0.001ms 0

29 33 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

30 34 25Gbps 0.001ms 0

30 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

31 34 25Gbps 0.001ms 0

31 35 25Gbps 0.001ms 0

32 36 25Gbps 0.001ms 0

33 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

34 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

35 36 25Gbps 0.001ms 0

#### 1.2 flow.txt

flow.txt用于描述AI训练过程中所有的存储访问，文件结构如下：

* 第一行：存储访问的个数
* 第二行开始，每一行：#编号1 #编号2 #priority #packet #start\_time #end\_time
  + #编号1向#编号2传输一个文件，分为#packet个包进行传输，优先级为#priority, 开始时间为#start\_time, 结束时间为#end\_time。

下面的flow.txt描述了一个具有2个存储访问的过程：

2

2 21 3 1 2.00 9.50

1 25 3 1 2.00 9.50

注意：本例中优先级相同。

#### 1.3 trace.txt

trace.txt描述所有storage node的端口，目的是用于ns3进行日志文件的记录。即ns3在仿真过程中将对trace.txt中包含的编号对应的存储节点进行日志文件的记录。trace.txt文件格式如下：

* 第一行：所有storage node的端口总数量
* 第二行开始的每一行：
  + 一个storage node端口的全局统一编号

在本例子中，一共有24个端口，trace.txt内容如下：

24

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

First line : tracing node #Node

IDs...

### NS3配置文件准备

* 我们需要将前面的三个输入文件放到ns3工程的下面的路径中：

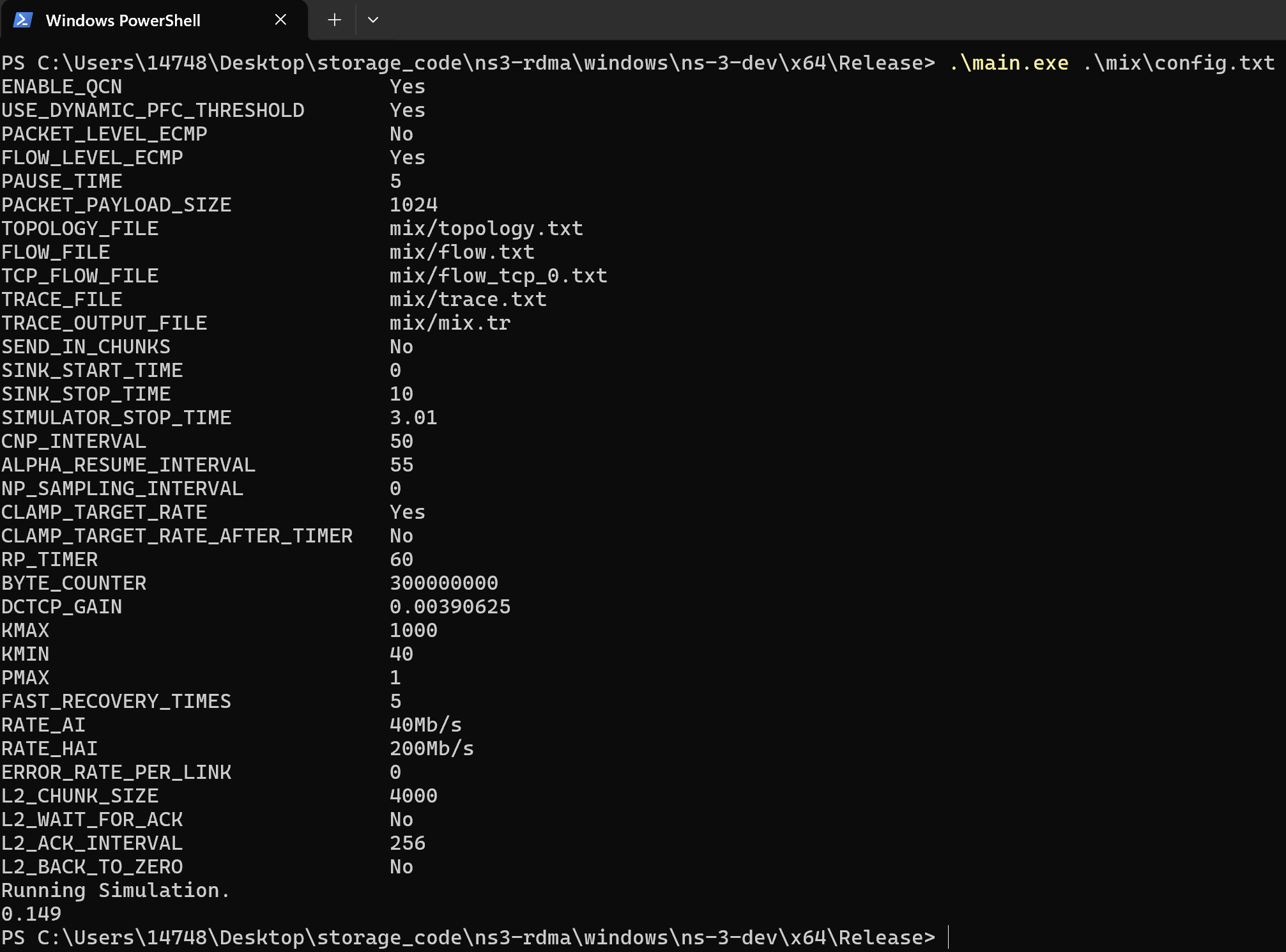
ns3-rdma/windows/ns-3-dev/x64/Release/mix/

注意：替换ns3项目中的同名文件。

* config.txt中的重要配置参数
  + **APP\_START\_TIME：**数据开始传输时间，应设置为0
  + **SIMULATOR\_STOP\_TIME：**仿真停止运行时间，这个参数可以尽可能的设置的大一些，例如可以设置为10.0

### 仿真运行

下面，我们运行第二章,2.6小节中的命令可以得到如下的仿真结果



上图倒数第二行显示，本次仿真运行时间为0.149s。

### 输出文件

输出文件为:

ns3-rdma/windows/ns-3-dev/x64/Release/mix/mix.tr

输出文件的内容为：

2.000008 /25 1.2>1.26 u 6436 0 3

2.000008 /21 1.3>1.22 u 19237 0 3

输出文件的每一行的含义：

#时间戳 #存储节点端口全局统一编号 #发送端ip>#接收端ip #协议类型 #端口号 #包的序列号 #优先级

例如，第一行：

* 时间戳：2.000008
* 存储节点端口全局统一编号: 25
* 发送端ip: 1.2
  + 这个ip是由ns3进行编码的，这里1.2对应的是编号为1的存储节点的端口
* 接收端ip: 1.26
  + 这里对应的是编号为25的存储节点端口
* 协议：u
  + udp
* 端口：6436
* 包的序列号：0
* 优先级：3

这一行日志文件表示在时间戳为2.000008的时候，在编号为25的存储节点端口观测到了一个存储访问的包，该访问是由编号为1的存储节点端口发出，目的地为编号为25的存储节点端口，使用的协议是udp协议，udp端口为6436， 观测到的包为存储访问的第一个包，优先级为3。

由于我们在flow.txt中定义的每个存储访问仅仅只包含一个包，所以mix.tr中只观测到了两个包的日志。

### 5、仿真结果评价

组委会提供了一个python 脚本"gen\_result.py"，用来对当前仿真过程进行评价。运行命令为：

python gen\_result.py --config config.txt --trace mix.tr

运行完成后会生成一个result.txt

## 本赛题仿真及解题思路说明

1. 赛题提供3个输入文件

* topology.txt 及对应的拓扑结构图（附件1）
* trace.txt
* flow.txt

1. 解题思路提示

* **Step1:把项目运行起来。**请同学们根据前面的提示，将赛题提供的三个输入文件替换到项目中，在不做任何优化的情况下，尝试把仿真运行起来。（提示：由于赛题数据量较大，完整仿真一次耗时大约5个多小时）
* **Step2:阅读项目代码。**请同学们阅读这个项目的源代码，弄清楚该项目是如何进行仿真的。资料如下
  + NS3模拟基础。<https://www.cnblogs.com/myworld7/p/12189686.html>
  + 入口函数：ns3-rdma\examples\tutorial\third.cc的main函数
* **Step3:学习DCQCN算法。**请同学们学习RDMA拥塞控制的经典论文DCQCN，参考资料如下
  + Congestion Control for Large-Scale RDMA Deployments.Zhu et.al.

<https://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p523.pdf>

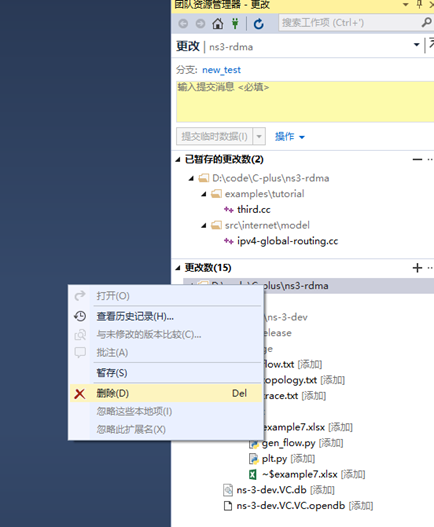
* + RDMA拥塞控制经典论文DCQCN中文解读。

<https://zhuanlan.zhihu.com/p/663891770>

* **Step4：理解DCQCN是如何在NS3中进行实现的。**
  + 这一步需要同学们在Step2和Step3的基础上进行进一步的学习与理解。
* **Step5：观察运行过程，寻找优化点。**思路提示如下
  + **思路一：设计新的拥塞算法，提升网络吞吐能力。**
    - 拥塞控制算法实现，可以参考代码仓库的master和timely分支。
  + **思路二：设计新的负载均衡算法，提升网络吞吐能力。**
    - 例如，动态更改存储访问的路径。输出文件mix.tr中可以观察到每个文件的传输在每个交换机的端口使用情况，可以修改端口从而修改存储访问的路径。
  + **思路三：修改Hash算法，提升网络吞吐能力。**
    - 代码位置：Ipv4GlobalRouting::RouteOutput (Ptr<Packet> p, const Ipv4Header &header, Ptr<NetDevice> oif, Socket::SocketErrno &sockerr)。
  + 其他提示：
    - 如有需要修改MTU，位置在udp-client.cc的76行
  + 更多思路请同学们自行探索。

## 结果提交

本赛题提交以下三个内容：

1. 可编译的源代码
   1. 注意，删除Visual Studio运行生成的相关过程文件
   2. 
2. result.txt
   1. 该文件在仿真结束之后，通过gen\_result.py脚本生成
3. 方案设计文档

请同学们将提交内容打包为xxxx.zip 进行提交。