**高性能计算互连大作业 –**

**Flow control based routing for 3D meshes**

<https://github.com/LGY35/NOC_Simulator>

原模拟器实现了2D Mesh网络拓扑，现需要实现基于流控制路由的3D Meshes网络，需要对网络结构、路由算法、消息结构进行更改。

**（一）网络结构：**

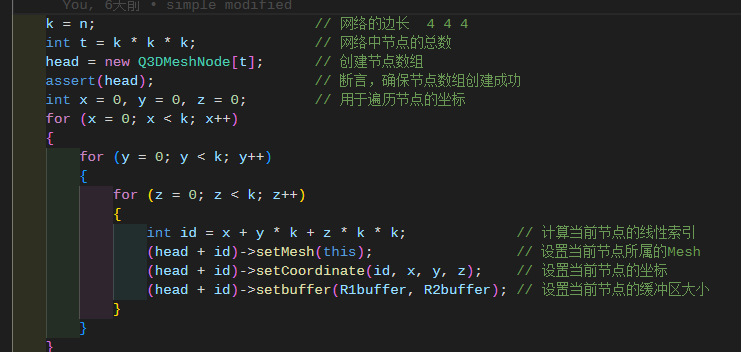
为防止混淆，将原来的Q2DTorus.cpp、Q2DTorus.h和Q2DTorusNode.cpp、Q2DTorusNode.h文件名称全部更改为以 \*3DMesh\* 命名的文件。文件中关于网络结构的定义，从2D Mesh全部更改为 3D Mesh。原来实现的为4 \* 4 的2D mesh，现增加一个维度。

**Q3DMesh.h：**

只需要将相应的2DTorus改成3DMesh即可

**Q3DMesh.cpp：**

构造函数中，需要增加z维度，将拓扑从2d增加到3d，比如：

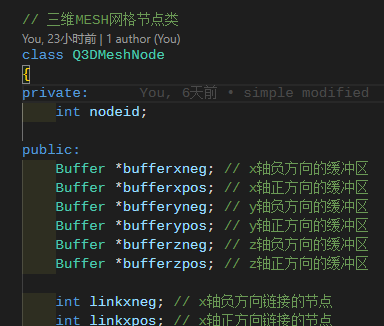


下面各个节点的链接也是如此。

最后的Clear中也要对循环索引最大值修改为k\*k\*k。

**Q3DMeshNode.h：**

在节点类定义Q3DMeshNode中，增加z维度，各个变量相同增加。



此处Q3DMesh.h和Q3DMeshNode.h 存在循环包含的问题，虽然是warning，但是还是解决一下。

（1）在 `Q3DMeshNode.h` 中继续使用前向声明 `class Q3DMesh;`。

（2）在 `Q3DMesh.h` 中包含 `Q3DMeshNode.h` 以获取 `Q3DMeshNode` 的完整定义。

（3）在 `Q3DMeshNode.cpp` 中包含 `Q3DMesh.h`。

**Q3DMeshNode.cpp：**

各个方法函数中，设置节点的坐标、buffer、linkbuffer、bufferPlus、clearBuffer都增加一个z维度（zneg、zpos为邻接节点）。



**（二）路由算法：**

路由算法方面，Flow control based routing for 3D meshes 任务，是基于流控制，而路由算法可以有多种选择。此处继续选择源代码中的DOR路由算法，不过需要增加z维度，并对路由代码进行了简化。

**Allrouting.h：**

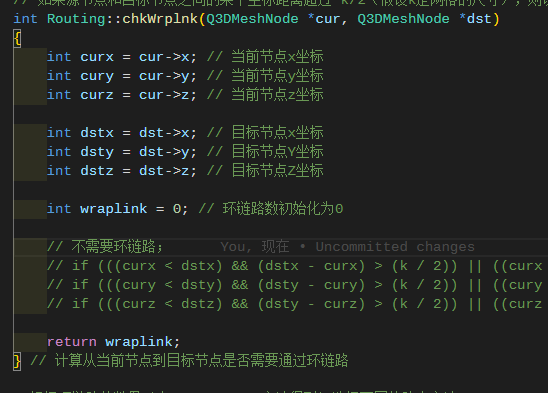
将相应的2dtorus命名改成3dmesh即可

**Routing.h：**

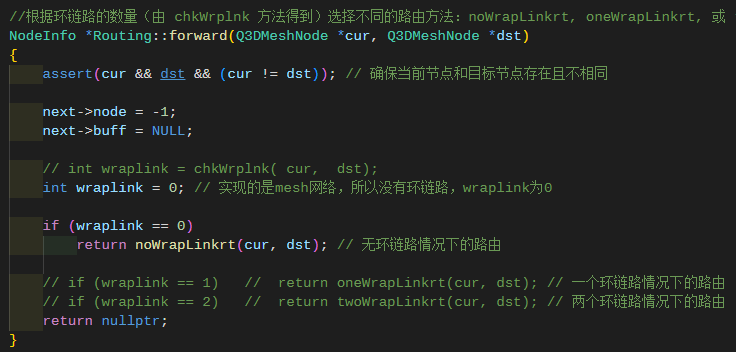
将相应的2dtorus命名改成3dmesh即可

**Routing.cpp：**

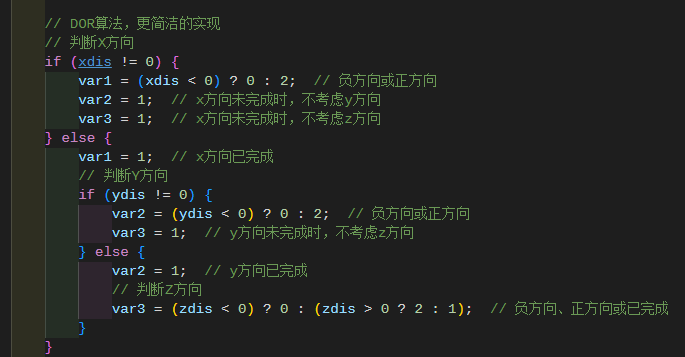
环链路相关函数：torus需要环链路，但是Mesh没有，所以此处环链路只需要保持0即可。

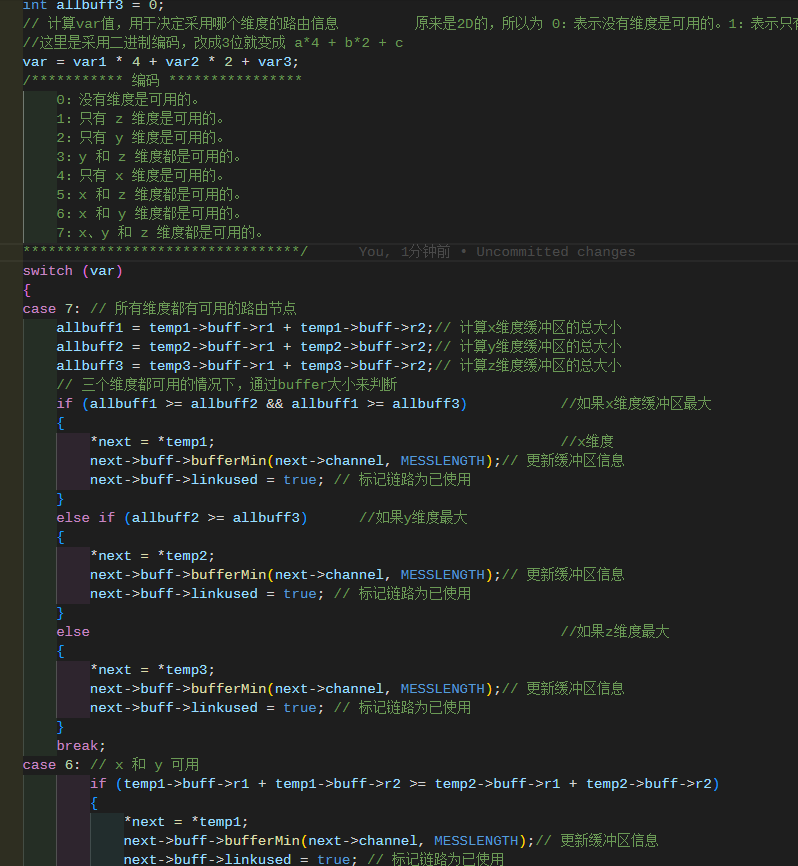


同理，在对路由选择时，只保留环链路为0的情况即可。最后有1、2个链路的路由函数直接删除即可。



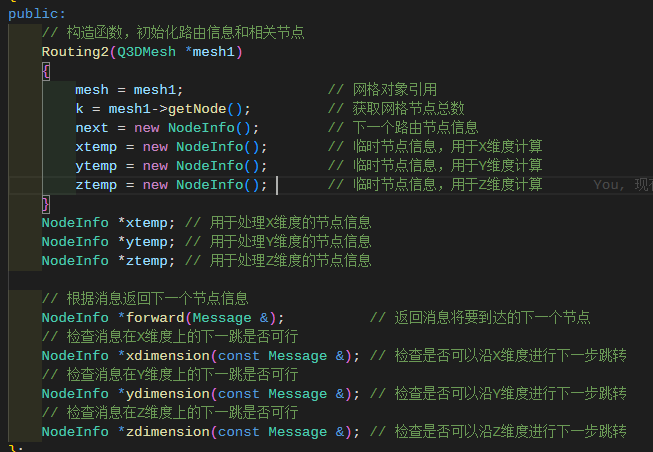
在无链路路由方法函数中，增加z维度，然后更简洁的实现DOR算法。





**Routing2.h：**

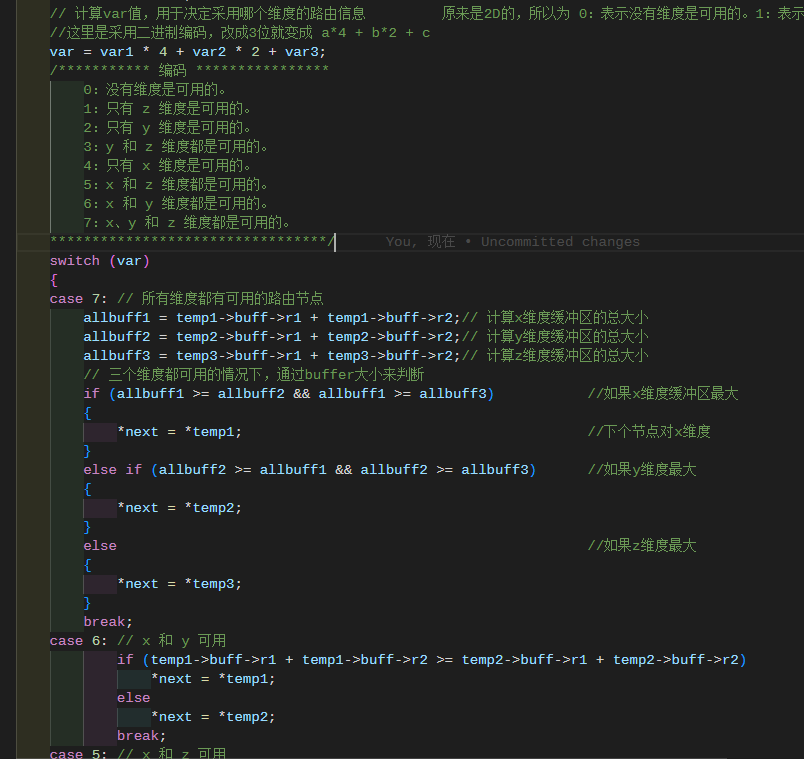
负责路由信息相关，增加z维度即可。



**Routing2.cpp：**

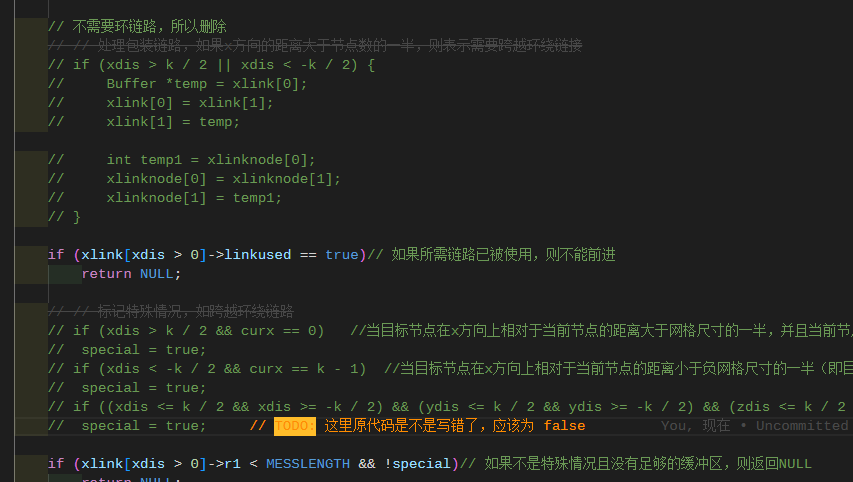
首先是转发函数，根据消息s决定其路由路径NodeInfo \*Routing2::forward(Message &s)

需要增加z维度，并且在消息转发机制上，从2维度的编码改成3维度的编码，即var = var1\*2 + var2 改成 var = var1\*4 + var2\*2 + var。然后修改路由选择逻辑。如下：



下面有处理沿xyz三个维度的路由决策函数。这里也需要相应的修改。

首先需要增加一个z维度的决策函数。3D mesh中无环链路，所以关于链路判断的部分可以删除，下面是x维度的部分，y、z维度相同，都要删除。



**（三）消息结构-流控制算法：**

流量控制(flow control)管理网络缓冲区和链路的分配，决定何时以及以什么粒度将缓冲区和链路分配给消息，并且决定如何在使用网络的众多消息之间共享这些资源。

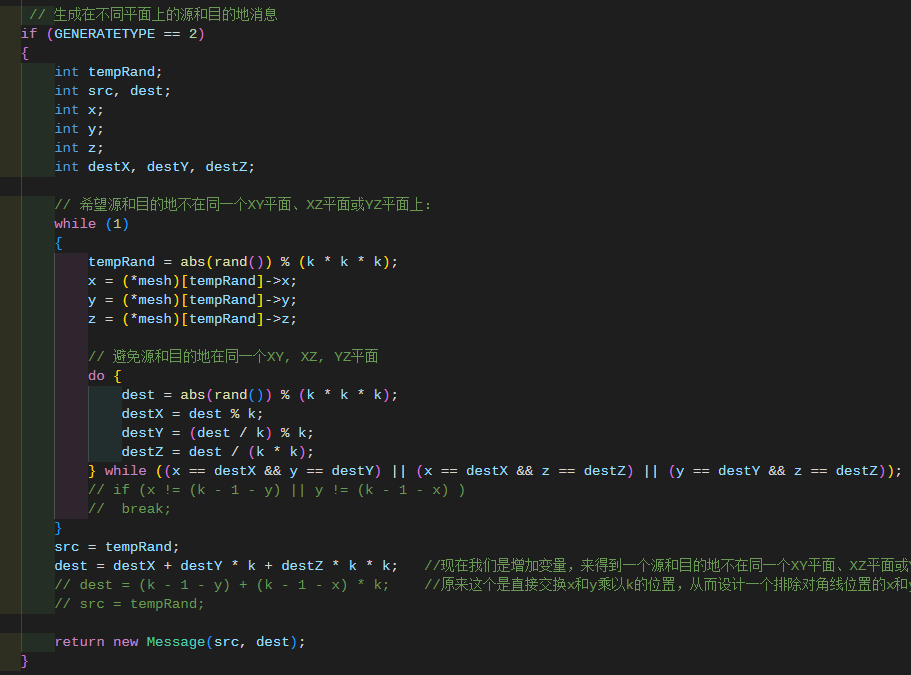
Event.cpp中完成了基于flit的流量控制机制，所以我们可以使用，即：

**wormhole flow control：**

一旦头部flit（header flit）为自身找到了前进的路径，后续的flits会跟随相同的路径前进，而不需要为每个flit重新确定路径。一旦后续节点有足够的缓冲区用于该flit，该flit就可以离开当前节点。以flit大小的单位分配存储和带宽，但链路在数据包在路由器中的生命周期内被持续占用。结果是，当一个数据包被阻塞时，该数据包占用的所有物理链路都处于空闲状态。虫洞流量控制通过允许flit在下游缓冲区可用时尽快离开路由器来降低数据包的延迟（在没有争用的情况下，延迟与虚拟截断相同）。

**Event.cpp：**流量模式选择uniform模式。内部的维度相应改变为k^3。同时后面几个消息类型也相应的改变。





在**testfuc.cpp**中， tail flit的索引原来是19，但是消息长度在Q3DMesh.h中定义为16，所以这里需要更改，不然会出现指针指向未知内存区域。



**（四）运行**

**main.cpp：**



main.cpp中只需要做简单的名称调整即可。

**环境配置：**

使用VSCode开发，所以编写Makefile脚本如下。执行时：

**make** : 编译

**make run** : 编译并运行，生成coredump文件用于调试（或直接运行）

**make debug** : 编译并启动gdb（或直接启动gdb）

**make debug-core** : 编译并使用gdb调试core文件

1. TARGET\_EXEC := NOC
3. BUILD\_DIR := ./build
5. SRC\_DIRS := ./src
6. SRCS := $(shell find $(SRC\_DIRS) -name '\*.cpp')
8. INC\_DIRS := ./include
9. INC\_FLAGS := $(addprefix -I,$(INC\_DIRS))
11. CPPFLAGS := $(INC\_FLAGS) -MMD -MP -g  # 加入 -g 以启用调试信息
13. DEPS := $(OBJS:.o=.d)
14. OBJS := $(SRCS:%.cpp=$(BUILD\_DIR)/%.o)
16. # 将目标文件链接成最终的可执行文件。
17. $(BUILD\_DIR)/$(TARGET\_EXEC): $(OBJS)
18. $(CXX) $(OBJS) -o $@ $(LDFLAGS)
20. # C++源代码构建步骤
21. $(BUILD\_DIR)/%.o: %.cpp
22. mkdir -p $(dir $@)
23. $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $< -o $@
25. # 新增一个目标来运行程序，允许生成 core 文件
26. run: $(BUILD\_DIR)/$(TARGET\_EXEC)
27. ulimit -c unlimited; ./$(BUILD\_DIR)/$(TARGET\_EXEC)
29. # 这个目标是专门用来调试已经崩溃并生成了 core 文件的程序。
30. debug-core:
31. gdb -c core $(BUILD\_DIR)/$(TARGET\_EXEC)
33. # 这个目标用于直接启动 GDB 调试正在运行的或是可重新启动的程序。
34. debug: $(BUILD\_DIR)/$(TARGET\_EXEC)  # 新增 debug 目标
35. gdb $(BUILD\_DIR)/$(TARGET\_EXEC)
36. # 这个目标用于直接启动 GDB 调试正在运行的或是可重新启动的程序。
38. .PHONY: clean
39. clean:
40. rm -r $(BUILD\_DIR)
41. rm -f ./core\*
43. -include $(DEPS)

**输出：**

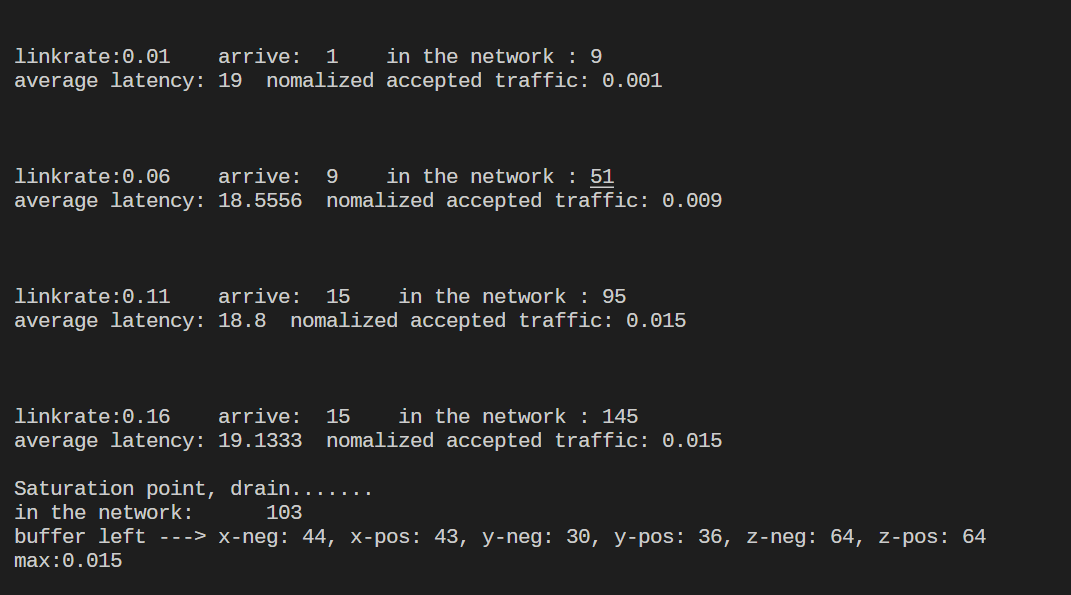
**通过改变threshold 和 totalcircle的值，测试了3组输出数据。**

**输出（1）：**

**threshold = 80000;**

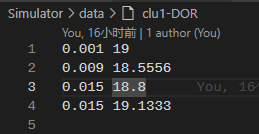
**totalcircle = 10000;**

**此时峰值吞吐率为0.015**

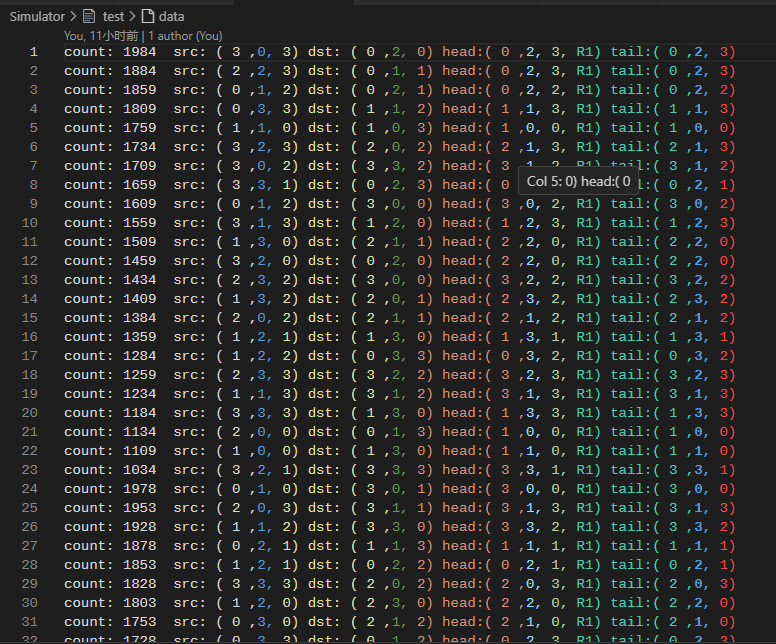


输出文件：

/NOC\_Simulator/Simulator/data/clu1-DOR



/NOC\_Simulator/Simulator/test



这组结果的threshold设置的非常大，其实没有必要。而totalcircle比较小，饱和点很快就达到了。

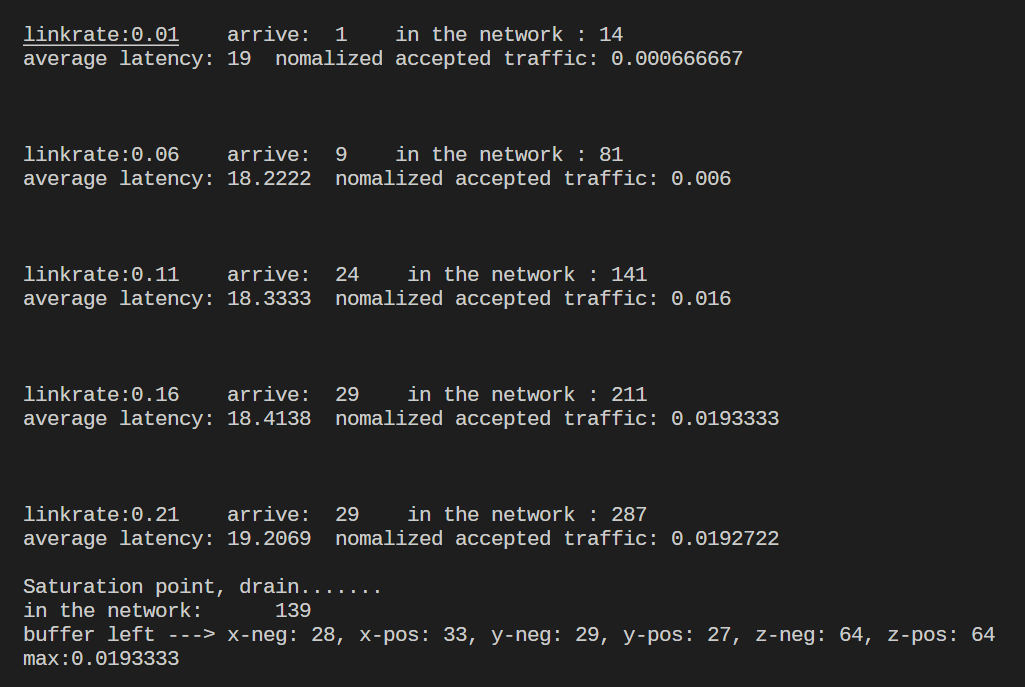
**输出（2）：**

**threshold = 8000;**

**totalcircle = 15000;**

**此时峰值吞吐率为0.0193333**

**饱和速度变慢。**



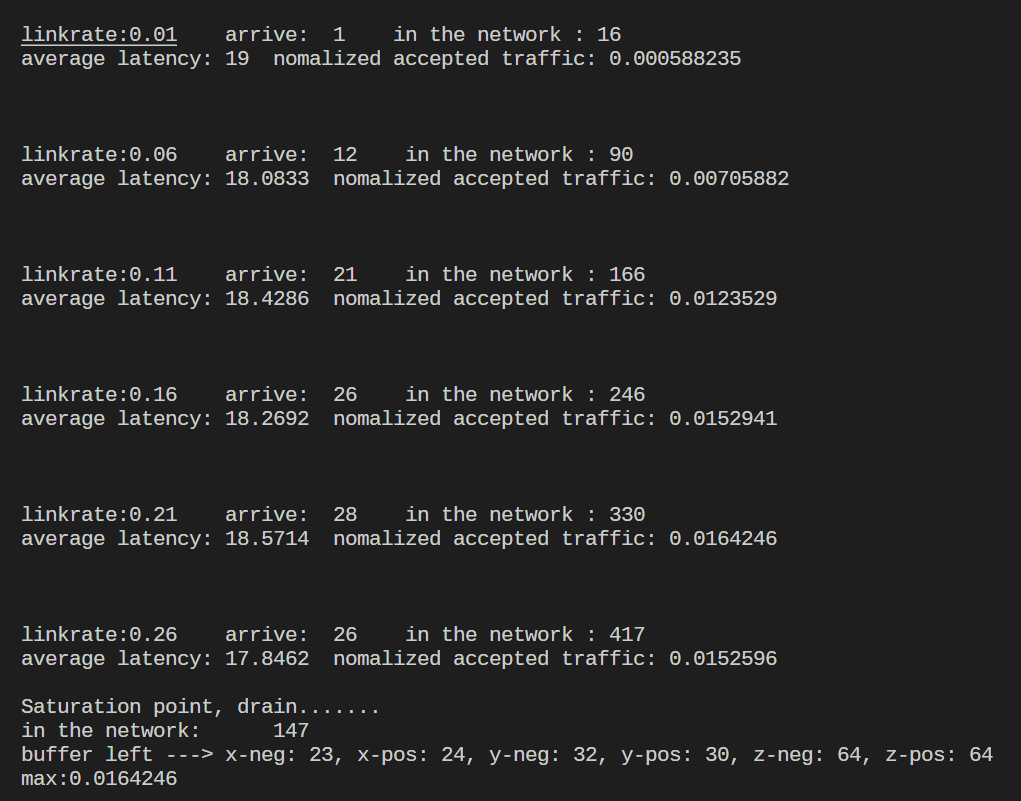
**输出（3）：**

**threshold = 10000;**

**totalcircle = 17000;**

**此时峰值吞吐率为0.0164246**

**饱和速度更慢。**



**总结：**

1.提高threshold 可以允许更多消息在网络中存在，这有助于提高系统的利用率，但同时也可能引起拥堵，尤其是在节点或链路资源有限的情况下。

2. 增加totalcircle 提供了更长的时间窗口来处理消息，这有助于处理更多的累积消息，但也意味着在达到饱和点前，系统需要运行更长时间。

3.随着 linkrate的增加，网络能够在饱和前处理更多的消息，但随后迅速达到饱和点。饱和点后，网络中的消息数量显著增多，表明进一步增加 linkrate 已无法有效提升吞吐量，反而可能导致网络性能下降。