《通信网理论与技术基础》教材配套实验

爱尔兰即时拒绝系统

实验指导书

1 实验目标

利用 OMNeT++软件，构建爱尔兰即时拒绝系统的仿真，并对占用中继线数的均值和呼损进行数值分析验证。

2 实验背景

爱尔兰即时拒绝制系统描述了有 m 条中继线，电话呼叫流的到达率为λ的一类电话交换系统。每到达一个呼叫，均可以随机占用任何一条空闲的中继线，并完成接续通话。每个呼叫的服务时间均服从参数为μ的指数分布。当系统中的 m 条中继线全部繁忙时，呼叫被拒绝。

3 实验方法提示

3.1 理论值计算

(1) 占用中继线数 N 的均值

业务量的强度通常称为呼叫量，理论值为：

$$a = \frac{\lambda}{\mu} \qquad \text{（式 1）}$$

其中λ代表呼叫的到达率、μ代表单条中继线的服务率。

利用生灭过程分析爱尔兰即时拒绝系统，我们可以得到系统的稳态分布为$p_k = \frac{a^k}{k!}e^{-a}$。可知一个泊松流进入交换系统后，输出仍然是一个泊松流，即占用的中继线数满足参数为 a 的泊松过程。依据泊松过程的性质可知：

$$E[N] = a \qquad \text{（式 2）}$$

其中 N 代表中继线的个数，a 代表呼叫量。

(2) 呼损 B(m,a)

呼损代表当呼叫到达系统时，由于中继线全部繁忙被拒绝的概率，记为：

$$\frac{a^m / m!}{\sum_{r=0}^{m} a^r / r!} \qquad \text{（式 3）}$$

其中 a 代表呼叫量、m 代表中继线的条数。

3.2 仿真

(1) 仿真图

在 OMNeT++软件中，编写简单模块 Source、Exchange、Server 和 Sink 建立仿真。其中，Source 模块负责产生到达率为 lambda 的呼叫；Exchange 模块在中继线都繁忙时负责拒绝呼叫、在中继线有空闲时负责将呼叫送至空闲的中继线；Server 模块负责模拟中继线，并按照服务率 mu 服务呼叫；Sink 模块负责模拟呼叫结束。在仿真中，不同中继线的服务率相同。

在 Source 模块中构建一个统计量，负责统计产生的总的呼叫请求数。在 Exchange 模块中构建一个统计量，负责统计新呼叫到达时平均被占用的中继线的条数。在 Sink 模块中构建一个统计量，用于统计通过的总呼叫数。

仿真图如下：

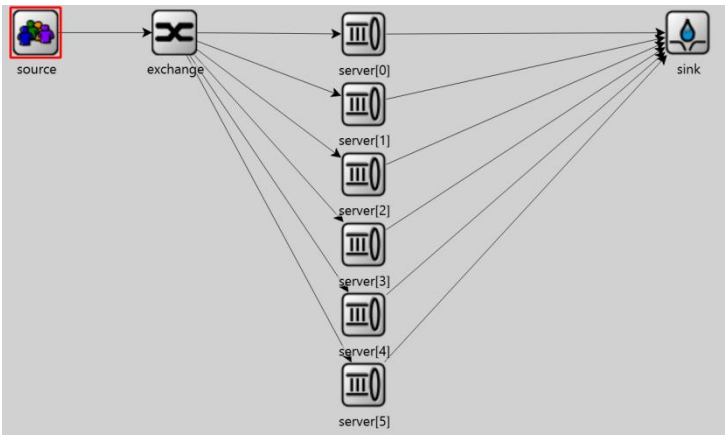

图 1 “M/M/m/m”排队模型的仿真图

(2) 参数设置

在表 1 中，参数“lambda”代表 Source 模块中泊松流的到达率；参数“mu”代表 Server 模块服务呼叫的服务率；参数“numServers”代表服务台的个数，也代表 Exchange

模块输出门向量的长度。“numServers”参数不在简单模块中，而是在用来定义复合模块的“MMss.ned”文件中。

表 1　“M/M/m/m”排队模型的仿真参数设置

网络模块	参数名
Source	lambda
Server	mu
——	numServers

(3) 代码提示

创建呼叫的方法，参见“实验 3”代码提示部分的“cMessage 类”。

①　门

在本次仿真中，Exchange 简单模块的输出门需要与多个 Server 简单模块的输入门连接。因此，我们构建门向量来实现这个需求。

门是模块之间的连接点。OMNeT++中门分为三种类型：输入门、输出门和输入输出门。在创建门时，可以创建单个门或门向量。若创建门向量，其长度可以在定义时在后面的方括号中指定，也可以在使用该向量时再指定。

- 定义门

在“Exchange.ned”文件中构建如下代码。下面的代码表示针对输入门，构建了单个名为 in 的门；针对输出门，构建了名为 out[]的门向量。门向量的长度未指定，在构建复合模块时再指定。

```
gates:
    input in;
    output out[];
```

②　自定义中继线的条数

下面阐述如何实现在 Qtenv 仿真中，可以自定义中继线的条数。效果如下图所示：

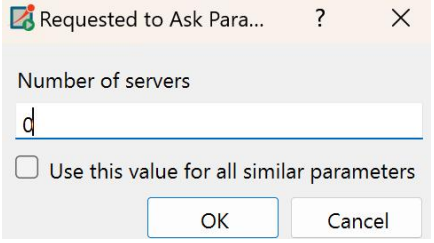

图 2 Qtenv 中实现用户输入中继线条数

“MMss.ned”文件用于构建“M/M/m/m”的网路拓扑结构。在文件中，利用如下代码实现上述的功能。代码中的内容不全，需要在“submodules”关键字中继续添加除了Server 的其他简单模块的子模块，并在“connections”关键字中将子模块连接起来。

在代码中，利用“parameters”关键字声明了变量“numServers”，该变量表示当前仿真中中继线的条数。“@prompt("Number of servers")”实现在 Qtenv 仿真界面弹出的输入框中，提示用户这个变量的含义。之后在“submodules”关键字中，生成了“numServers”个 Server 简单模块的子模块，并且命名为 server[0]、server[1]...server[numServers-1]。最后，在“connections”中，利用循环将子模块连接起来。

```
import ned.IdealChannel;
network MMss
{
    parameters:
        int numServers @prompt("Number of servers");
        @display("bgb=1000,1000");
    submodules:
        server[numServers]: Server {
            @display("p=467,64,m,1");
        }
    connections:
        for i=0..numServers-1 {
            exchange.out++ --> server[i].in;
            server[i].out --> sink.in++;
        }
}
```

③ 获取中继线的忙闲状态

当新的呼叫到达 Exchange 模块时，Exchange 模块需要获取各条中继线的忙闲状态，来决定是否拒绝呼叫。

- 在 Server 模块中构建变量反应中继线的忙闲状态

“serverBusy”是一个布尔类型的变量，当中继线繁忙时，设置该变量的值为“true”；当中继线空闲时，设置该变量的值为“false”。

- 在 Exchange 模块中获取变量“serverBusy”的值

需要先引用“Server.h”文件。在下面的代码中，“gateSize()”函数用来获取 Exchange 模块中输出门的门向量大小。字符串“modulePath”表示 Server 简单模块的子模块的名称，子模块的名称被命名为 server[0]、server[1]...server[numServers-1]。利用指针获取变

量“serverBusy”的值。

```
// 利用循环检测所有的中继线是否繁忙
for (int i = 0; i < gateSize("out"); i++) {
    // 字符串 modulePath，用来表示需要检测的中继线的名称
    std::string modulePath = "server[" + std::to_string(i)+"]";
    // 中继线的指针
    Server* server = check_and_cast<Server*>(getModuleByPath
(modulePath.c_str()));
    // 获取中继线的状态
    status = server->serverBusy;
}
```

4 实验步骤

1）编写 Source、Exchange、Server 和 Sink 四个简单模块；

2）利用编写的四个简单模块构建网络拓扑结构；

3）设置参数的值，进行仿真；

4）仿真结束后，利用仿真得到的数据对系统进行分析。

5 提交要求

学生需要提交一份实验报告，并将实验报告、源代码、数据打包成压缩包上交。其中实验报告内容包括但不限于：

1）研究背景与问题描述

2）总体思路与方案论证

3）结果分析与结论

4）课程学习收获与体会建议