状元谷实习第一周学习报告

报告人：温嘉宝，邓文豪

1. **拜占庭将军问题：**
   * 1. 介绍

拜占庭将军的问题是协议协议领域的众多问题之一。 1982年，Leslie Lamport在与Marshall Pease和Robert Shostak一起撰写的一篇论文中描述了这个问题。 在观察了Dijkstra的哲学家就餐问题所引起的过度关注之后，Lamport围绕一个故事问题构建了他的论文。

这个问题是围绕一个假想的将军做出决定进行攻击或撤退，并且必须将决定传达给他的副官而建立的。 特定数量的这些行为者是叛徒（可能包括将军）。不能依赖叛徒来正确传达命令; 甚至，他们可能会主动改变消息以试图颠覆这个过程。

把将军们统称为进程，发起命令的将军是源进程，发送给其他进程的命令是消息。 叛徒将军和副官是错误的进程，忠诚的将军和副官是正确的进程。 撤退或攻击的命令是带有一个bit信息的消息：1或0。

一般而言，协议问题的解决方案必须通过三项测试：终止，协议和有效性。 应用于拜占庭将军的问题，这三个测试是：

1.解决方案必须保证所有正确的进程最终都能得出关于命令值的决定。

2.所有正确的进程必须能在所给命令中做出相同的决定。

3.如果源进程是正确的进程，则所有进程最后做出的决定必须与源进程相同。

请注意，这样做的一个有趣的副作用是，如果源进程出错，则所有其他进程仍然必须就相同的值达成一致。 他们同意什么价值并不重要，他们只需要同意。 因此，如果将军具有颠覆性，所有副官仍然必须达成共同的，一致的决定。

* + 1. 难点

这个协议问题不适合简单的天真解决方案。 例如，想象一下，源进程是唯一的叛徒进程。 它告诉了一半的进程，他们的命令值为0，告诉另一半进程，他们的命令值为1。

在从源进程接收到命令之后，剩余的进程必须使用这些命令将决定的值达成一致。 这些进程可以快速相互轮询，看看他们从源进程中获得了什么值。

在这种情况下，想象一个进程的决策算法，它从源进程接收到的消息是0，但是看到其他进程有的说正确的值是1。 鉴于冲突，进程知道源进程有故障，给两个不同的对等体赋予不同的值，或者是它的对等进程有故障，谎报自己从源进程中收到了的命令。

可以得出某人撒谎的结论，但最终决定谁是叛徒似乎是一个不可逾越的问题。 事实上，可以证明在某些情况下无法做出决定。 用于显示这一点的经典示例是当只有三个进程时：一个源进程和两个对等进程。

在图1和图2所示的两种配置中，对等进程尝试通过在从源进程接收到它们之后相互发送它们的建议值来达成共识。 在图1中，源进程（P1）有故障，向对等体发送两个不同的值。 在图2中，P3出现故障，向对等体发送了错误的值。

可以看到P2在这种情况下遇到的困难。 无论它处于哪种配置，传入的数据都是相同的。 他无法区分这两种配置，也无法知道要信任哪两个其他进程

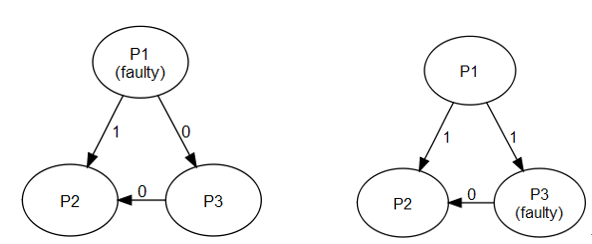


图1 图2

通过在问题上抛出更多无故障进程，这种情况不一定会好转。 如图1和图2所示的简单算法可能会让每个进程告诉其他进程从P1接收的内容。 然后，进程将通过在其传入消息中采用简单多数值来确定正确的值。

根据这种方法的规则，很容易证明，无论系统中有多少进程，一个协作者的颠覆性源进程都可能导致一半进程选择攻击，而一半进程选择撤退，导致最大值混乱。.

**Lamport，Pease和Shostak算法**

1982年，Lamport，Pease和Shostak发布了一个相当简单的解决方案。 该算法假设有n个进程，m个叛徒进程，其中n> 3m。 因此，对于诸如图1和图2中具有1个故障过程的情况，系统中必须至少有4个过程才能达成一致。 （对于本文的其余部分，n将始终作为进程计数，m将始终作为叛徒进程的数量。）

Lamport提出的容错的两个条件

IC1：即所有的忠诚的副官要遵守同一个命令，即达成一致；

IC2：假如将军是忠诚的，那么每一个忠诚的副官都应该按照将军的意思行事。

Lamport的算法是一个递归定义，m = 0的基本情况和m> 0的递归步骤：

|  |
| --- |
| **Lamport的算法定义** |
| 算法OM(0)   1. 将军把他的值发送给每一个副官 2. 每一个副官使用从将军那里收到的值   算法OM(m), m > 0   1. 将军把他的值发送给每一个副官 2. 对于每个i，把vi看作副官从将军那里收到的值。副官i在算法 OM(m‑1)充当将军，把vi发送给其他n-2个副官（不需要发送给自己和将军） 3. 对于每个i，j≠i，让vi作为副官在第二步中从将军处收到的值（使用 算法OM(m‑1)）。副官i使用{v1,v2,...vn}中的最多的值 |

**相关证明**

**证明1： 为什么在没有签名的情况下，n>3m即可容错？**

在Lamport的论文中，在证明OM(m)在最多只有m个叛徒，以及超过3m个将军的时候可以满足IC（1）和IC（2）条件的时候，先引入了一个引理：

LEMMA1：对于任意m 和k ，如果有超过2k+m 个将军和最多k 个背叛者，那么算法OM(m) 满足IC2 （IC2 指的是，如果将军是忠诚的，所有的副官遵守将军命令）。

证明：

（1）当m=0的时候，如果将军是忠诚的，因为在OM(0)的时候忠臣会遵守将军发来的命令。而此时的将军是忠臣的所以，即满足IC2

（2）当m>0的时候。用数学归纳法，通过证明OM(m-1)有效来证明。因为假设了n>2k+m,在OM(m-1)时剩余的将军数是n-1,但是叛徒数仍然是k(因为上1轮中的将军是忠臣)，n>2k+m -> n-1>2k+m-1. 又m-1>=0 -> n-1>2k. 即在OM(m-1)中有忠诚的副官比叛徒多，所以可以投票得出正确结果。在OM（m-1）时可证。

接着证明“证明1”中的问题：

**证明：通过m的归纳法证明，我们通过假设OM(m-1) 成立来证明OM(m) m>0。**

（1）首先考虑发送命令的将军是忠诚的。那么将引理中k 设为m 则OM(m) 满足IC2 ，IC1 在发令将军是忠诚的情况下也满足。

（2）如果发送命令的将军是叛徒。那么在下一轮中，总共有3m个副官，其中有m-1个副官是叛徒，有3m-(m-1)=2m+1个忠臣,2m+1>m-1 即 在这里所有的忠臣是可以达成一致的（这一步是为了说明OM（m-1）是成立的）。下面再推出OM（m-1）的表达式：这里除去要发送命令的一个将军外还有超过3m-1个，有3m-1>3(m-1) 即 n'>3m'，所以在OM(m-1)是满足条件的，得证。

在这里也同时证明了为什么是m+1轮交换。因为OM（m）满足。m->0 共 m+1

Lamport的算法实际上分两个阶段进行。 在第一步中，进程迭代m + 1轮发送和接收消息。 在算法的第二阶段，每个进程都会获取已经给出的所有信息，并使用它来做出决策。

**第一阶段**

算法的第一阶段只是数据收集。 该算法定义了所有进程之间的m + 1轮消息传递。

在第0轮，将军将命令发送给所有副官。 在完成他的工作后，将军将退出，等待剩余的工作完成。 没有人向将军发送任何其他消息，并且将军将不再发送任何消息。

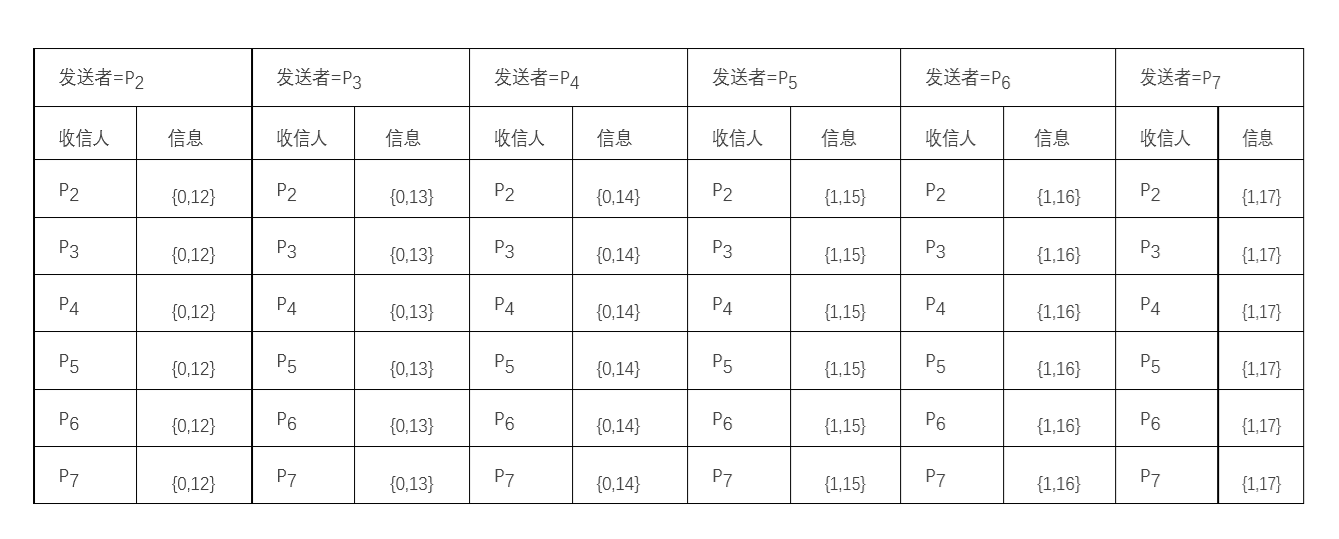
在每个剩余轮次中，每个副官组成一批消息，每个消息都是包含值和路径的元组。 该值只是1或0.路径是一串进程ID，

<ID1，ID2，... IDn>路径的含义是，消息从PID1发出经过PID2...一直传到PIDn。路径中不包含环，即一个PID在路径中最多只能出现一次。

第1轮。每个进程向所有其他进程（包括其自身）广播消息，但不包括将军，消息是内容包括从将军处收到的值，以及自身的ID。

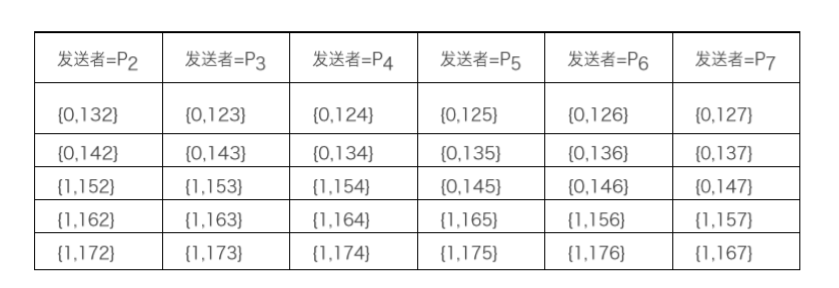
在随后的几轮中，事情变得更加复杂。 每个进程获取它从上一轮收到的所有消息，将其进程ID附加到允许的位置，并将这些消息发送到所有其他进程，包括其自身。 （“允许的地方”只是意味着进程会跳过所有加入自己ID之后会形成环的路径）

例如，假设在第0轮，P1（叛徒）告诉P2，P3和P4命令值为0，并告诉P5，P6和P7命令值为1。在第1轮中，以下信息将会被发送：



消息数量在第二轮中上升。 从上一次迭代中，我们知道每个进程现在有六个值，它们在前一轮中收到 - 一个来自其他六个非源进程的消息 - 并且需要将每个消息发送到所有其他进程 ，这可能意味着每个进程将发送36条消息。

第2轮中每个发送者发出的信息如下：



P2在第1轮中收到的6条消息是{0,12}，{0,13}，{0,14}，{1,15}，{1,16}和{1,17}。 根据之前的定义，P2将其进程ID附加到路径，并将每个结果消息转发给所有其他进程。 它可以在第2轮广播的可能消息是{0,122}，{0,132}，{0,142}，{1,152}，{1,162}和{1,172}。 第一条消息{1,122}在元组的路径值中包含一个循环，因此它被丢弃，留下五条消息发送给所有进程。P2在第二轮发出的第一条信息{0,132}表示：P2告诉你在第1轮P3告诉它在第0轮P1告诉它值是0。

很容易看出，随着进程数量的增加，交换的消息数量开始迅速增加。 如果有N个进程，则每个进程在第1轮发送N-1个消息，然后在第2轮发送（N-1）·（N-2），（N-1）·（N-2）·（N-3） 在第3轮中。

**第二阶段**

在每轮发送消息时，进程也会累积传入消息。 消息以树格式存储，每轮消息占据树的一层。 图3显示了具有六个进程的简单配置的树的布局，其中一个进程可能有问题。 由于m = 1，因此只有两轮消息传递：第一轮，其中将军向每个副官进程发送一个值，第二轮，其中每个进程将其值广播到所有其他进程。 两轮消息传递相当于树中的两个层。

树中的每个节点都有三个元素：输入值，路径和输出值。 输入值和路径是从对等进程接收的消息。 输出值未确定，在下图中，输出值最初设置为“？”，表示它们目前未知。

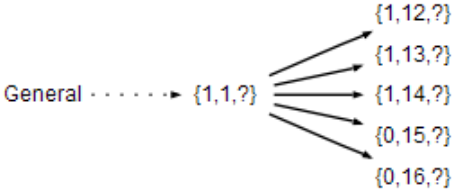


图3

在图3中，有六个进程，并且将军（P1）是叛徒 - 向前三个副官发送1，向后两个发送0。 随后的一轮消息传递导致P2具有一个看起来如图3所示的信息树。（因为只有将军是叛徒，在这种情况下，所有其他进程将具有相同的树。）

一旦进程完成构建其树，就可以决定一个值。 它通过从树的叶子向上工作，计算每个等级的多数值并将其分配到它上面的等级来实现这一点。 每个级别的输出值是附加到每个节点的数据结构中的第三个项目，并且这些值在信息收集阶段都是未定义的。

**计算输出值是一个三步过程：**

1.树中的每个叶节点（等级m的所有值）将其输入值复制到输出值。

2.从等级m-1开始并向下工作，每个内部节点的输出值被设置为其所有子节点的大部分输出值。 如果出现平局，则使用任意的平局判断器来分配默认值。 所有进程都必须使用相同的默认值。

3.完成后，该过程在等级0的唯一节点的输出中具有决策值。

在图3中，该过程的步骤1将初始值分配给叶节点。 在下一步中，评估{1,1,1,0,0}的多数值并返回值1，该值被赋值为等级0中的输出值。因为这是最高等级，所以过程完成时，P1决定值为1。

例子中的每个副官值对于其所有节点都将具有相同的路径，在这种情况下，由于只有将军是叛徒，我们知道所有副官在其所有叶子上都具有相同的输入值。 因此，所有进程都将达成相同的值1，即履行协议属性。

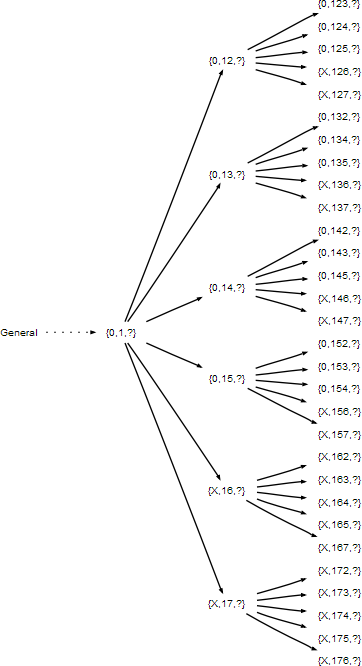
**一个更复杂的例子**

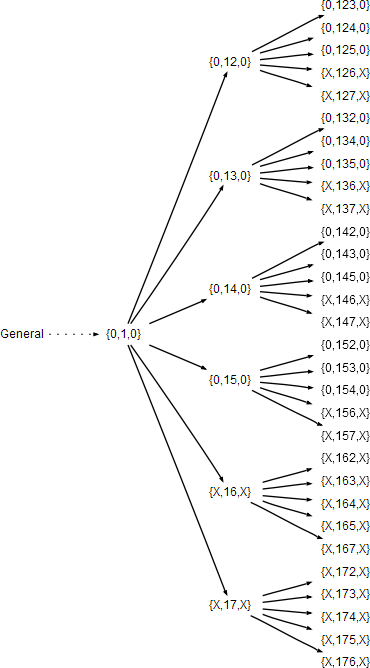
考虑一个n = 7和m = 2的例子。 使用P1作为将军，将P6和P7作为叛徒。 在最初的三轮信息交换之后，我们有了如图4所示的三排树。

在这些树中需要注意的重要一点是，两个叛徒进程的任何输入值的输入值插入了值“X”。 我们不知道P6和P7可以在给定的轮次中发送什么，所以一般来说，将尝试通过算法而不将其不正确的消息限制为任何特定值。

您将看到在第1级，路径17和16的值都设置为X.在第一轮中，两个叛徒进程可能将错误的值传递给所有其他进程，并且可能随意更改发送到不同进程的值。

由于第1层中的那些错误值，我们看到它们在第2层中经常出现。不正确的值不仅出现在来自叛徒进程的直接消息中，而且出现在来自正确进程的任何消息中，只要它经过了叛徒进程。

总之，在叶节点处，我们在叶节点处有18个欺骗性值，并且只有12条准确的消息通过正确的过程一直追溯到将军。 显然，如果我们只是对我们收到的大部分信息进行投票，那么我们就容易选择了错误的值。

图4 图5

幸运的是，树的布局保证了我们实际上会获得正确的值。 在图4中，尚未发生输出值的汇总，因此每个节点在输出值中都有一个问号。 在图5中，显示了输出值。 叶级别的输出值设置为输入值，X用于指示叛徒进程的未知值。

当最后一层回溯到第二层时，具有路径12,13,14和15的节点的输出值都是0，其中16和17设置为X，因为它们的值是不确定的。

最终回溯到最高等级成功地将输出值设置为0，因为四个输入设置为0并且只有2个设置为X.任务完成。 并且由于计算方式，无论两个叛徒进程发送什么欺骗性值，都将获得正确的结果。

**算法实现及应用于foxtable：**

**1、算法实现代码逻辑：**

**加入功能(实现模拟BFT网络函数模块一个)**

vector<String>(Decisions) SimBFT(int SourceID, int FaultyNum, int ProcessNum, [String srcValue]){}

**主要逻辑(下文提到网络中的主机即为进程):**

主要逻辑(下文提到网络中的主机即为进程):

注意:若源进程非坏进程,达到共识的所有进程决策均与源进程一致.否则其他进程的共识有可能与源进程不一致.

当输入网络的坏主机数 m >= n/3 提供警告

0.收到外界输入的信息

1.生成网络拓扑特征

->定义网络中的总主机数目

->定义输入网络的源信息(即将外界信息转换为源信息)

->定义发信主机编号

->随机设置坏主机编号 SetFaultyProcess()

->发信主机被设为坏主机,提供警告

->统一定义坏主机的发信行为:随机发送8位字符长度的字符串 GetValue()

->设置主机决策默认值,用于打破决策时1与0相等的情况 GetDefault()

->DefaultValue = "Retreat!"

2.生成网络主机(进程)

->每个主机共用同一个网络拓扑特征

->定义每个主机收信后存储信息的决策树

->生成决策树 GenerateChildren()

->用一个广义表(List)存储每个结点与子节点间的路径关系,用于深度遍历 mChildren

->用一个广义表(List)存储每一层所有结点的路径信息,用于每层的水平遍历 mPathsByRank

->用一个广义表(List)存储每个路径对应的结点信息 mNodes

->定义每个主机的行为

->发信 SendMessage()

->即往其他主机的决策树的每个结点填入信息

->收信并存储入决策树 ReceiveMessage()

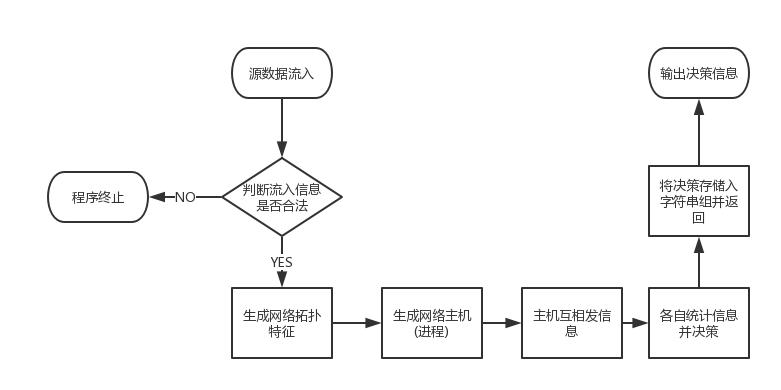
->对本机决策树每层水平遍历,并调用决策函数

->决策 Decide()

->统计数目最多的信息,送往上层

3.主机互相发信息

4.各自统计信息并决策

5.将决策存储入字符串组并返回 Return vector<String> decisions

**2、应用于Foxtable**

**原理**：在Foxtable中Table的指定单元格值更新事件触发函数运行。

**设计**：

用家账号：用户11人，开发者1人

Table设计：

用户X可见：Table: Table.UserX

Column: Table.SendingBox(FaultyNodeNum, Userx)

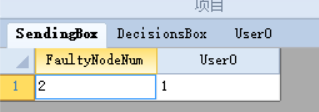
开发者可见：所有Table

**运行示例**：

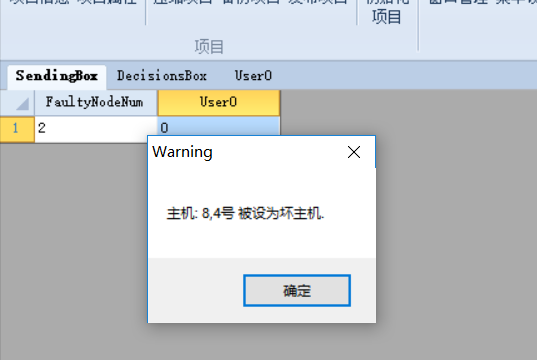
**10.14更新：可任意输入16位字符串，具体请看文末**

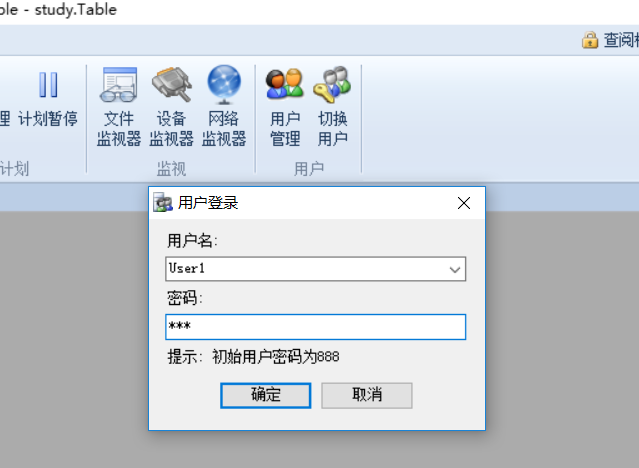
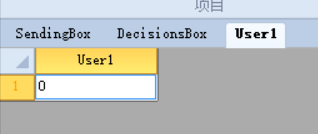
指定用户作为Command发送信息。登陆某个用户，在Table：SendingBox中的FaultyNodeNum填入当前网络坏主机数

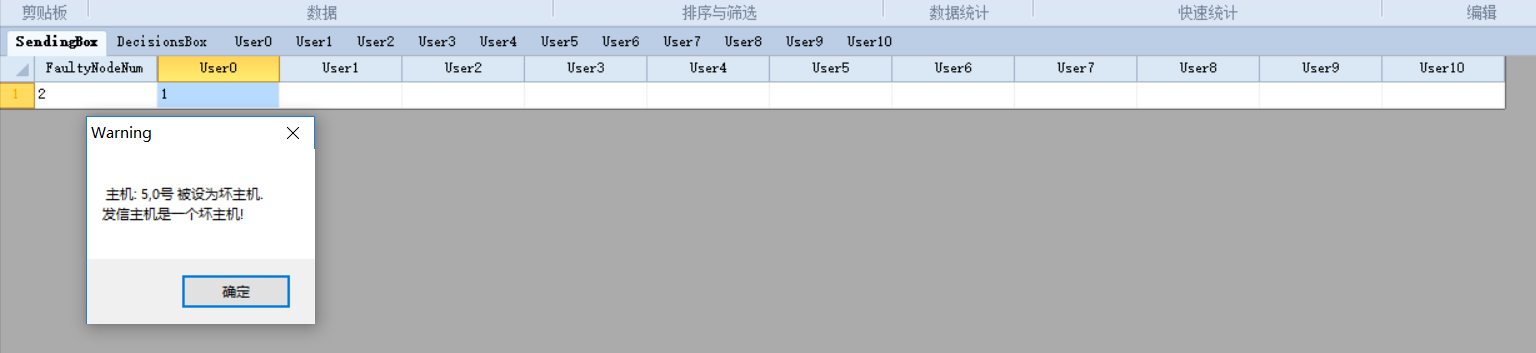
在UserX填入要对整个网络发送的信息。此后，**网络会随机指定坏主机为哪些主机**。在DecisionsBox可得到本机的决策。

填入坏主机数量：

填入需要对全网络广播的信息（1或0）：

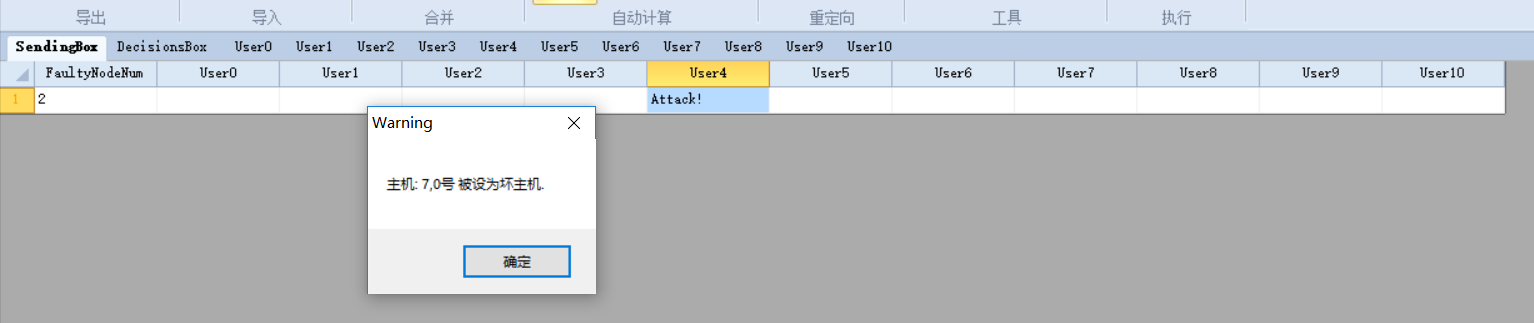
在DecisionBox可看到本用户的决策。

此时登陆其他用户账号，可以查看其他用户的决策：

也可以直接登陆开发者账号，来查看网络全局，指定某个主机发信：

查看全局决策：

**10.14更新：**

**在SendingBox输入任意16位内字符串**

**网络达到了共识。**

**值得注意的是：当发信主机被指定为坏结点时，整个网络达成的共识将有可能与发信主机发出信息不一致。**