3.1 学习目标

完成此任务后,您将获得以下技能的充分专业知识:

1. 设计路由协议
2. 链接状态(迪克斯特拉的)算法
3. UDP 套接字编程
4. 处理路由动态

4. 分配规范

本节提供分配的详细规范。

4.1 实施详情

在此分配中,您将实现链路状态路由协议。

您的程序应命名为Lsr.py(或**Lsr.java**或**Lsr.c**)。它将接受一个命令行参数

- *配置TXT,*此文件将包含路由器ID及其端口No在第一行。第二条线路具有此路由器的邻居数。后续线路将包含邻居的信息,包括路由器ID、相邻路由器的成本以及邻居用于交换路由数据包的端口号。 下面提供了此文件的示例。

由于我们不能让您使用真正的网络路由器,因此模拟网络中所有路由器的路由程序将在单个台式计算机上运行。但是,路由协议的每个实例(对应于网络中的每个路由器)将侦听不同的端口号。如果您的路由软件在单个台式计算机上正确执行,它也应该在真正的网络路由器上正常工作

假设路由协议正在实例化为路由器A,有两个邻居B和
C。有关如何执行路由程序的简单示例(假设它是名为 Lsr.py 的 Python 程序)如下:

Python *Lsr.py* *配置A.txt,*其中 configA.txt 是路由器 A

的配置文件,具有以下详细信息:

configA.txt：

A 5000
2
B 6.5 5001
C 2.2 5002

第一条线路具有此路由器 (A) 的路由器 ID 和用于通信的端口 No 5000(传输和接收链路状态数据包)。此文件的第二行指示路由器 A 的邻道数。之后,每个邻居都有一条线。它从邻居 ID 开始,然后是到达此邻居的费用,最后是该邻居用于通信的端口号。 例如,上面配置A.txt 中的第二行表示邻居B 的成本为 6.5,并且此邻居使用端口号 5001 接收和传输链路状态数据包。路由器 ID 将是大写字母表,您可以假定测试方案中的节点不会超过 10 个。但是,不要假设路由器 IP 必须从字母 A 开始,或者它们始终按顺序排列。链路成本应为浮点数(最多第一个十进制),端口号应为整数。这三个字段将在配置文件的每一行中的两个连续字段之间用单个空白分隔。链接成本将是静态的,一旦初始化,将不会更改。此外,链路成本在两个方向上都是一致的,即如果从 A 到 B 的成本为 6.5,则从 B 到 A 的链路也将具有 6.5 的成本。您可以假定用于标记的配置文件将与上述描述一致,并且没有任何错误。

重要提示:值得重新指出,最初每个路由器只知道其直接邻居的成本。
路由器在启动时没有全局知识(即有关整个网络拓扑的信息)。

规范的其余部分分为两部分,从基本规范作为第一部分开始,然后部分向基本规范添加新功能。***CSE***学生要尝试两个部分的作业,而***非 CSE***学生只需尝试基本规范。因此,CSE 和非 CSE 学生的标记指南是不同的。这些出现在指示标记分布的规范的末尾。

第 1 部分:基本规范

在链路状态路由中,每个节点将链路状态数据包广播到网络中的所有其他节点,每个链路状态数据包包含节点邻居的标识以及到达这些节点的相关成本。您必须在程序中实现简单的广播机制。初始化后,每个路由器创建一个链路状态数据包(包含适当的信息 - 参见教科书中链路状态协议的描述),并将此数据包发送到所有直接邻居。您将使用的链接状态数据包的确切格式由您决定。收到此链路状态数据包后,每个相邻路由器依次将此数据包广播到其自己的邻居(不包括在第一个连接状态数据包中接收此链路状态数据包的路由器)地方)。这种简单的泛洪机制将确保每个链路状态数据包在整个网络中传播。

某些节点可能比其邻居更早启动。因此,节点可能会将链路状态数据包发送到尚未运行的邻邦。您不必担心这一点,因为每个节点的路由程序将反复将链路状态数据包发送到其邻居,并且启动速度慢的邻居最终将获得信息。也就是说,当我们测试您的分配时,我们将确保所有节点同时启动(使用脚本)。

每个路由器应定期向其邻居广播链路状态数据包。
更新_间隔。应将此间隔设置为 1 秒。换句话说,路由器应每秒广播一个链路状态数据包。

真正的路由协议使用UDP交换控制数据包。因此,您必须使用UDP作为传输协议,用于在邻居之间交换链路状态数据包。请注意,每个路由器都可以查阅其配置文件,以确定其邻居用于交换链路状

态数据包的端口号。不要担心UDP不可靠的性质。由于您在一台计算机上模拟多个路由器,因此极不可能丢弃链路状态数据包。此外,由于链路状态数据包定期广播,偶尔的数据包丢失不会影响协议的操作。**如果您使用 TCP,将评估重大罚款**.

从所有其他节点接收链路状态数据包时,路由器可以建立网络拓扑的全局视图。给定整个网络拓扑的视图,路由器应运行Dijkstra的算法,以计算到网络内所有其他路由器的最小成本路径。每个节点应等待一个 ROUTE_UPDATE_INTERVAL(默认值为30秒)自启动后,然后执行Dijkstra的算法。由于网络中的节点不会超过10个,并且周期链接状态广播频率为1秒,因此每个节点的持续时间足够长,可以发现整个拓扑的全局视图。

路由器一旦完成Dijkstra算法的运行,就应打印到终端,这是每个目标节点(不包括自身)的最小成本路径以及此路径的成本。下面是某些任意网络中节点 A 的示例输出:

我是路由器 A
路由器 B 的最小成本路径:ACB,成本为 14.2
路由器 C:交流器的最小成本路径,成本为 2.5

在运行程序后,我们将等待ROUTE_UPDATE_INTERVAL的持续时间,以便输出显示(一些额外的时间将作为缓冲区添加)。如果输出在这段时间内没有出现,您将受到严厉的惩罚。如前所述,我们将测试拓扑中的网络大小限制为10个节点。默认值30
秒足够长,所有节点都能够接收来自其他每个节点的链路状态数据包并计算成本最低的路径。

您的程序应永久执行(作为循环)。换句话说,每个节点应保持每个 UPDATE_INTERVAL 和 Dijkstra 算法的广播链路状态数据包,并打印出每个ROUTE_UPDATE_INTERVAL
的输出。您应该能够终止路由协议的实例(例如,在相应的终端上键入 CTRL-C)。

限制链路状态广播:请注意,一种天真的广播策略;其中每个节点重新传输它接收的每个链路状态数据包将导致不必要的广播,从而增加开销。为了阐述此问题,请考虑在规范的后半部分(图1)中讨论的示例拓扑。节点 B 创建的链路状态数据包将发送到其直接邻居 A、C、D 和 E。这三个节点将依次向邻居广播此链路状态数据包。让我们考虑节点C,它广播B的链路状态数据包到 D. 请注意,节点 D 已经广播过 B 的链路状态数据包一次(当它直接从 B 接收它时)。节点 D 现已通过节点C接收了相同的链路状态数据包。因此,节点D不需要再次广播此数据包。您必须实施一种机制来减少这种不必要的广播。这可以通过几种方式实现。您可以选择任何方法来实现此目的。你必须在书面报告中描述你的方法。

4.1 测试拓扑

已为您提供 6 个路由器的示例拓扑的配置文件。使用此拓扑可增量测试实现。给出了路由器 A 的正确输出,在路由器 D 发生故障之前和之后。

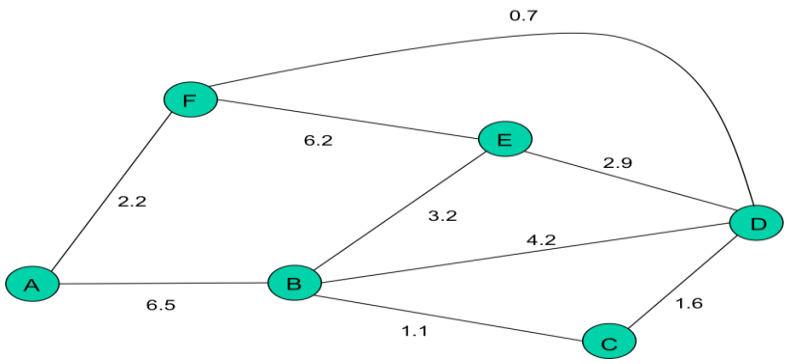

图 1:苔丝　t 拓扑

链接旁边的数字表示链接成本。6个节点的配置文件可从分配网页下载。在这些配置文件中,我们假定了以下端口分配:A 在 5000,B 在 5001,C 在 5002,D 在 5003,E 在 5004 和 F 在 5005。但是,请注意,在 CSE 服务器上测试实现时,某些端口可能由与您登录到同一 CSE 计算机的另一个学生使用。在这种情况下,相应地更改所有配置文件中的端口分配。

以下是路由器 D 发生故障前后路由器 A 的输出。

所有路由器都正常工作的输出:

我是路由器 A
路由器 C:AFDC 的最小成本路径,成本为 4.5
路由器 B:AFDCB 的最小成本路径,成本为 5.6
到路由器 E:AFDE 的最小成本路径,成本为 5.8
路由器 D:AFD 的最小成本路径,成本为 2.9
路由器 F:AF 的最小成本路径,成本为 2.2