计算机网络体系结构

短接技术-实验报告

指导教师 赖英旭

学号 S201861422 姓名 于泽群

学号 S201861442 姓名 詹 康

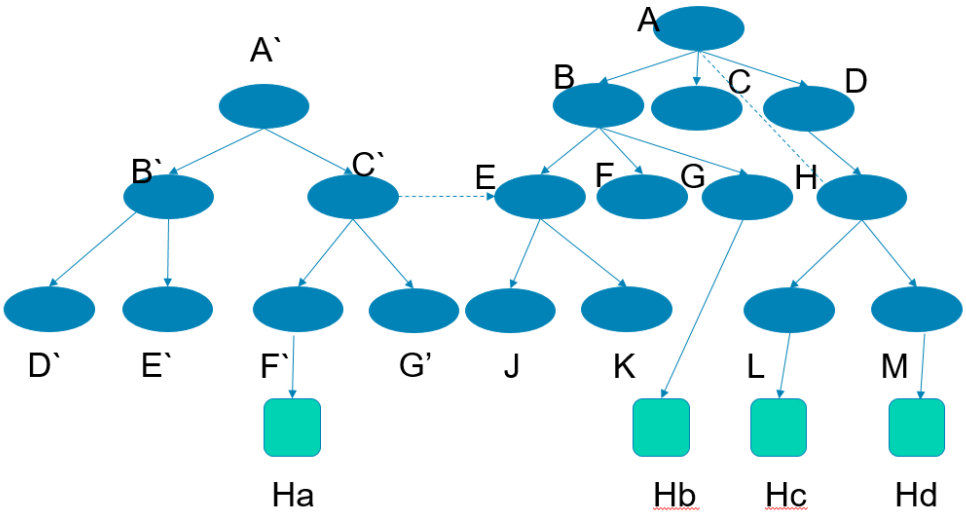
学号 S201861420 姓名 赵国帅

学号 S201861416 姓名 张梦隆

2018年10 月

1. 实验内容

1.1 层次网络结构



1.2 短接技术问题

1.2.1 短接通信问题之一——短接隧道

当 C’ 节点和 E 节点之间存在一条短接通路时，从源节点 Ha 到目的节点 Hb 之间的通信需要经过的路径是 Ha-> F’ -> C’ -> E -> B->G-> Hb；而进行反向通信时则会发生错误，因为从源节点 Hb 发送的数据包经过 Hb -> G -> B 之后到达 A 节点，A 节点会认为 Ha 节点的地址超出了本树状网的范围，而不能将数据包下行发送到 E 节点。

1.2.2 短接通信问题之二——重复路径

当 C’ 节点和 E 节点之间存在一条短接通路时，我们通过向 A 节点加入短接映射表表来解决前面短接隧道问题中反向通信的丢包问题，但是这样仍然有新的问题：从源节点 Ha 到目的节点 Hb 之间的通信需要经过的路径是 Ha-> F’ -> C’ -> E -> B->G-> Hb；而进行反向通信时需要经过的路径是 Hb-> G-> B-> A -> B -> E -> C’ -> F’ -> Ha，两次经过 A 节点的原因是只有将数据包通过 B 节点发送到 A 节点，才能从 A 节点的短接表中查到下一步应该经过 B 节点到达 E 节点进行短接通信，这样就导致了重复路径的产生。

1.2.3 短接通信问题之三——循环路径

当 A 节点和 H 节点之间存在一条上行短接通路时，源节点 Hc 到目的节点 Hd 之间的通信的路径是 Hc->L->H->A->D-> L->H->A->D……，出现了循环路径，Hc 到 Hd 无法通信。

当 A 节点和 H 节点之间存在一条短接通路时，源节点 Hc 到目的节点 Hd 之间的通信的路径是 Hc->L->H->A ->H->A……，出现了循环路径，Hc 到 Hd 无法通信。

2. 解决方案

2.1 短接隧道解决方案

在直接短接节点所在节点域的根节点中加入短接映射表，声明短接通路的存在。

2.2 重复路径解决方案

在间接短接节点域中保存短接信道对所有直接短接节点域和间接短接节点域的地址前缀到本侧直接短接节点域地址前缀的映射表项。

如果从直接短接节点域到间接短接节点域的路段上是单调上行路径，则沿路所有途经的节点域都要保存地址映射表项；如果该路段是非单调路径，则沿途节点域都不要保存地址映射表项.

2.3 循环路径解决方案

短接信道两侧的直接短接节点域之间，允许同一子树的单调下行路径，或者禁止上行短接信道。

3. 实验设计与实现

3.1 编程语言

JAVA

3.2 实验环境

JDK 1.8 +
IntelliJ IDEA 2018

3.3 层次网络底层的设计与实现

3.3.1 模型设计

3.3.1.1 地址前缀(AddressPrefix)

AddressPrefix 类
-List<Integer> aPrefix
+List<Integer> getaPrefix() +void setaPrefix(List<Integer> aPrefix) +void setaPrefix(AddressPrefix aPrefix) +String toString(String regex) +boolean isPrefixMatch(AddressPrefix addressPrefix) +boolean isPrefixEqual(AddressPrefix addressPrefix) +AddressPrefix getAddressPrefix(int len) +int getPrefixLength()

逻辑节点和数据包均含有地址前缀(AddressPrefix)信息,层次网络中地址前缀标识了逻辑节点的网络地址信息,其本身就包含有节点的部分地址信息.本实验采用整数+分隔符(比如 X.X.X,分隔符为“.”)形式的模拟地址前缀,而现实中可能将采用现有的 IPv6 地址.

该模型提供了地址前缀匹配(isPrefixMatch)和比较(isPrefixEqual)的方法,以及按长度截取地址前缀(getAddressPrefix 方法)和获取长度(getPrefixLength)等方法.

3.3.1.2 数据包(Data)

Data 类
- Stack<AddressPrefix> aPrefixStack - final AddressSuffix aSuffix - final String dataStr
+Stack<AddressPrefix> getaPrefixStack() +AddressSuffix getaSuffix() +String getDataStr() +void setCurrentAddressPrefix(AddressPrefix addressPrefix)

+AddressPrefix getCurrentAddressPrefix() +boolean isOriginData() +Data DataCode(AddressPrefix addressPrefix) +Data DataDecode()

数据包(Data)主要包含三种重要信息,分别是包头地址前缀(aPrefixStack),地址后缀(aSuffix)和数据信息(dataStr). 数据包的包头地址是可以动态扩展的,以便实现短接隧道技术,同时该模型提供了数据包包头解析(DataDecode)和包头扩展(DataCode)等方法.

3.3.1.3 逻辑节点(LogicalNode)

LogicalNode 类
-String alias -AddressPrefix aPrefix -ShortTable sTable -ShortTableMap sTableMap -Data data
+AddressPrefix getaPrefix() +String getAlias() +String setAlias() +ShortTable getsTable() +ShortTableMap getsTableMap() +void setsTable(ShortTable sTable) +void setsTableMap(ShortTableMap sTableMap) +void run() +synchronized void receive() +synchronized void confirm() +synchronized void drop() +synchronized void send() +AddressPrefix checkShortTable() +synchronized AddressPrefix checkShortTableMap() +synchronized void checkAddressPrefix() +synchronized void deliverUp() +synchronized void deliverDown() +synchronized void dataCode(AddressPrefix addressPrefix) +synchronized void dataDecode()

逻辑节点是构成层次网络的重要组成部分,每个逻辑节点都是独立的,均具有唯一确定的地址前缀,同时所有的逻辑节点又是统一的,均具有一些相同的处理流程. 逻辑节点(LogicalNode)具有别名(alias),地址前缀(aPrefix),短接表(sTable),短接映射表(sTableMap)以及获取转发的数据包(Data)字段,其中逻辑节点的地址前缀在初始化后不允许修改,其余属性均可动态变化.

逻辑节点具有接收数据包(receive 方法),确认数据包(confirm 方法),丢弃数据包(drop 方法),检查逻辑节点与数据包地址前缀(checkAddressPrefix 方法),上行转发数据包(deliverUp 方法),下行转发数据包(deliverDown 方法),数据包包头解析(dataDecode 方法),数据包包头扩展(dataCode 方法),发送数据包(send 方法),检查短接表(checkShortTable 方法),检查短接映射表(checkShortTableMap 方法)等方法, 这些处理方法(函数)实现了逻辑节点对数据包的接收-处理-转发过程,是整个层次网络运行时最基本的过程.

在实验中,每一个逻辑节点由独立的线程模拟运行,线程具有逻辑节点的全部功能,并由线程池统一管理(ScheduledExecutorService),每个线程唤醒的时间周期设定为 50 毫秒.

3.3.1.4 短接表(ShortTable)

ShortTable 类
- HashMap<String,List<AddressPrefix>> directShortPrefix - HashMap<String, Integer> shortStatus
+ void addShort(AddressPrefix aPrefix, List<AddressPrefix> extendedShortPrefixs) + void setShortStatus(AddressPrefix aPrefix, int status) + void removeShort(AddressPrefix aPrefix, Boolean isLogicalDelete) + AddressPrefix isShortMatch(final AddressPrefix addressPrefix, final AddressPrefix iPrefix) + AddressPrefix isShortMatch(final AddressPrefix addressPrefix)

短接表是实现短接通信的重要组成部分,其主要由对侧直接短接节点和信道状态字段构成.由于逻辑节点可能存在多条短接信道,因而其配置的短接信息也可能存在多个.为了实现数据包对对侧全部逻辑节点的通信,该短接表支持扩展地址前缀,即在保存直接短接节点地址前缀的前提下,可以将对侧层次网络的根节点地址前缀保存在短接表中,保证了所有逻辑节点的通信.

3.3.1.5 短接映射表(ShortTableMap)

ShortTableMap 类
- HashMap<String, List<AddressPrefix>>directShortPrefix - HashMap<String, Integer> shortStatus
+ void setShortTableMap(List<AddressPrefix> directShortPrefix, List<AddressPrefix> mappedPrefixs) + AddressPrefix isShortMatch(AddressPrefix addressPrefix)

短接映射表是实现短接隧道的重要组成部分,也是解决短接通讯中数据包返回问题的重要技术.短接映射表由对侧直接短接节点和信道状态字段构成.由于层次网络间存在多条短接信道,因而其配置的短接映射信息也可能存在多个.为

了实现数据包对对侧全部逻辑节点的通信,该短接映射表支持扩展地址前缀,即在保存直接短接节点地址前缀的前提下,可以将对侧层次网络的根节点地址前缀保存在短接映射表中,保证了所有逻辑节点的通信.

3.3.1.6 数据包交换(TransmitSimulation)

TransmitSimulation 类
- static volatile AddressPrefix currentAddressPrefix - static volatile AddressPrefix nextAddressPrefix - static volatile Data data - static volatile boolean transmiting
+ static AddressPrefix getCurrentAddressPrefix() + static void setCurrentAddressPrefix(AddressPrefix currentAddressPrefix) + static AddressPrefix getNextAddressPrefix() + static void setNextAddressPrefix(AddressPrefix nextAddressPrefix) + static boolean isTransmiting() + static void setTransmiting(boolean transmiting) + static Data getData() + static void setData(Data data) + static void clear()

由于本实验采用线程模拟逻辑节点转发数据包,因此实现线程间数据的通讯就代表了逻辑节点间的数据通讯.本实验设计了模拟转发类(TransmitSimulation)实现线程间通讯,并采用线程锁从而实现了并发控制,模拟转发类包含当前逻辑节点地址前缀(currentAddressPrefix),下一跳逻辑节点地址前缀(nextAddressPrefix)以及数据包(data)和数据包状态(transmiting),值得注意的是数据包状态将用于标识数据包是否在传输过程中,这也将用于 GUI 实现的过程.

3.3.2 接口设计

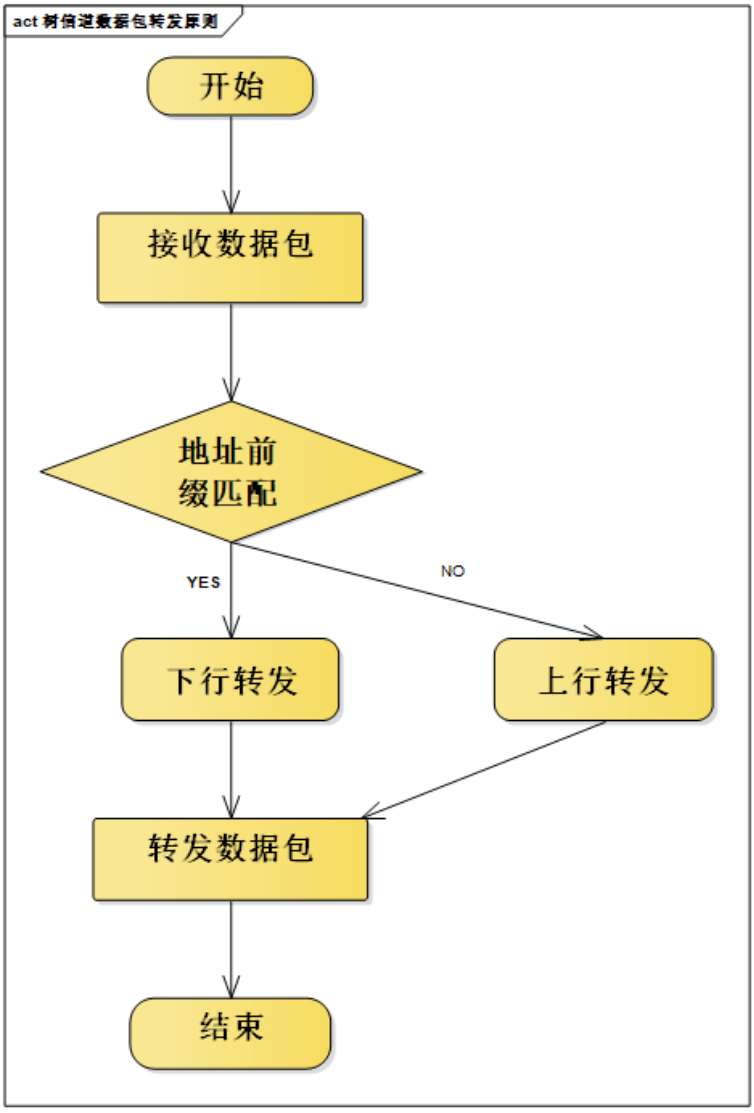
数据包在逻辑节点中经历了接收-处理-转发的过程,其中包含了最基本的数据(树信道)转发原则,即树信道的向上/向下转发,也包含了直接短接节点转发原则与间接短接节点转发原则.本实验涉及三种数据包转发原则,并由两类接口控制,分别为 TreeChannelTask, ShortChannelTask.

3.3.2.1 树信道数据转发接口(TreeChannelTask)

TreeChannelTask 接口
+ void checkAddressPrefix() + void deliverUp() + void deliverDown()

```
+ void dataCode(AddressPrefix addressPrefix)
+ void dataDecode()
```

树信道数据转发接口指明了数据包在包含树信道上的转发原则,包括检查逻辑节点与数据包地址前缀(checkAddressPrefix 方法),上行转发数据包(deliverUp 方法),下行转发数据包(deliverDown 方法),数据包包头解析(dataDecode 方法),数据包包头扩展(dataCode 方法). 下图展示了数据包在包含树信道上的转发过程.



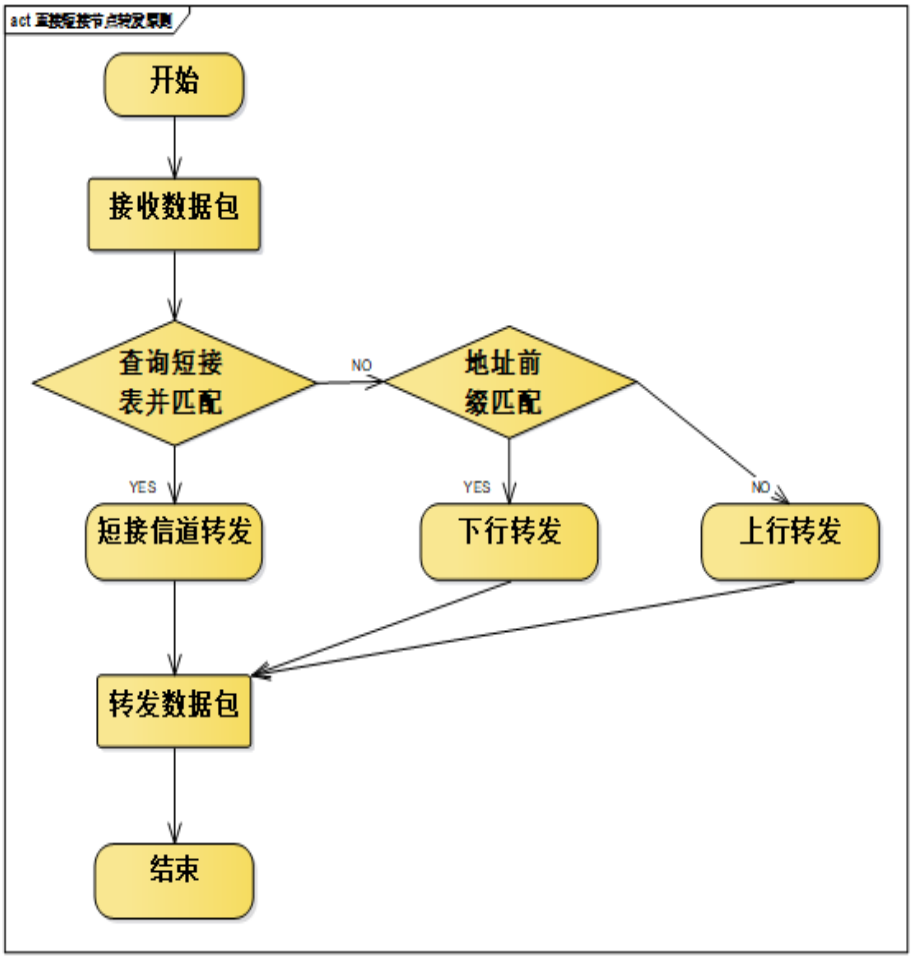
这个过程中最重要的步骤就是地址前缀匹配,它决定了数据包是上行转发还是下行转发. 如果数据包地址前缀与当前逻辑节点地址前缀匹配成功,则下行转发,否则上行转发.

3.2.2.2 短接信道数据转发接口(ShortChannelTask)

ShortChannelTask 接口
+ AddressPrefix checkShortTable() + AddressPrefix checkShortTableMap()

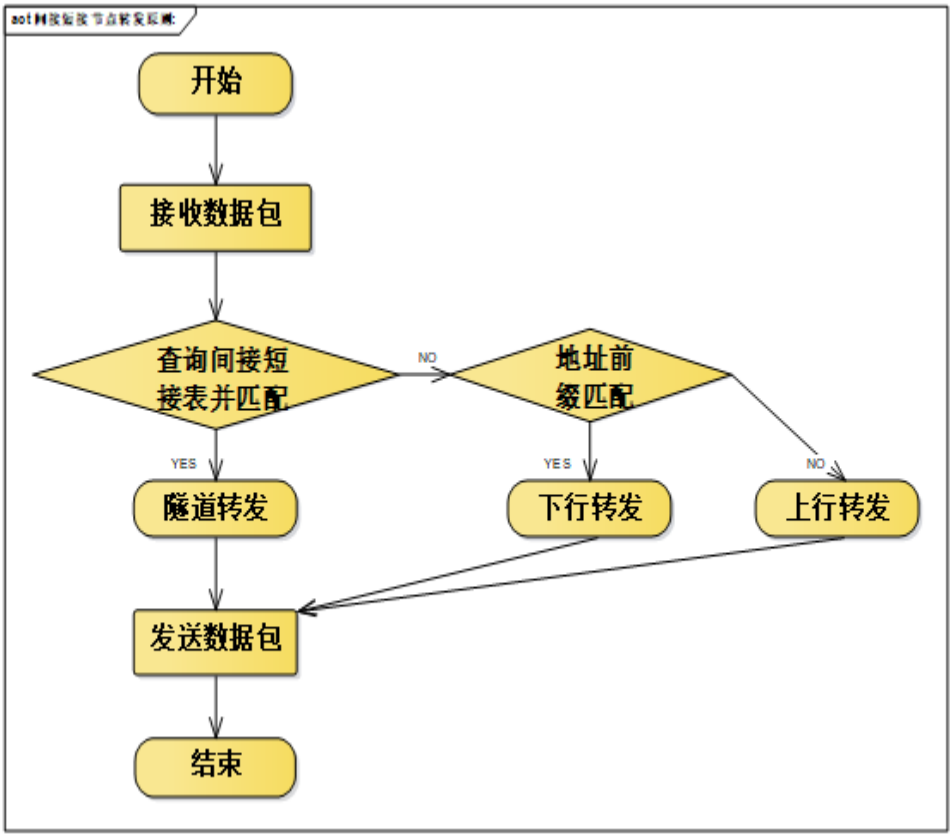
短接信道数据转发接口指明了数据包在包含短接信道上的转发过程,包括检查短接表(checkShortTable 方法),检查短接映射表(checkShortTableMap 方法)的方法. 实际上该接口定义了直接短接节点转发原则与间接短接节点转发原则,下面将详细解释这两个过程.

直接短接节点转发原则



如果逻辑节点是直接短接节点,该节点的转发流程则如上图所示.逻辑节点收到数据包并转发时首先查询短接表,如果短接表存在,数据包地址前缀与短接表前缀记录匹配成功且信道状态为 ON(即信道可通)则进行短接信道转发,否则将实现 TreeChannelTask 接口定义的转发过程.

间接短接节点转发原则



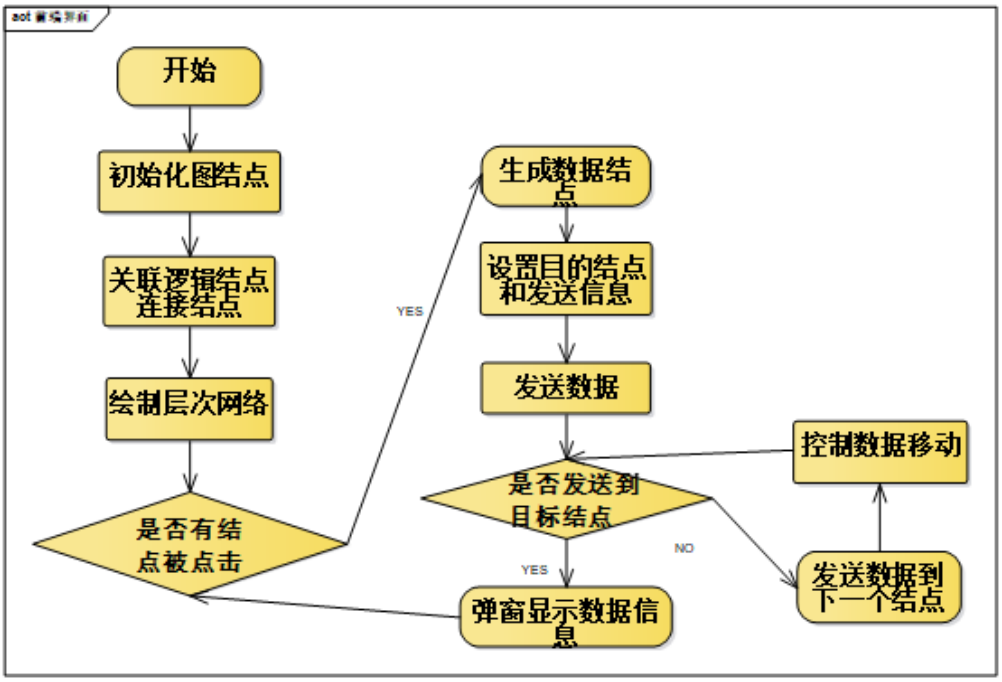
如果逻辑节点是间接短接节点,该节点的转发流程则如上图所示. 逻辑节点收到数据包并转发时首先查询短接映射表,如果映射表存在,数据包地址前缀与映射表前缀记录匹配成功且信道状态为 ON(即信道可通)则进行隧道转发,否则将实现 TreeChannelTask 接口定义的转发过程.

本实验采用了扩展包头的方法实现了隧道转发,而没有将原始数据包封装成新的数据包.数据包包头的扩展和解析接口定义在 TreeChannelTask 中,具体函数分别是 dataCode 与 dataDecode.

3.4 层次网络界面的设计与实现

该部分使用 java swing 技术绘制图形用户界面，使得对网络上的数据传输有一个直观的可视化的效果。

界面部分的程序设计主要分了三个包：Mode 包中负责记录视图模型的信息以及与后端逻辑结点相关联,Frame 包负责窗口和图形的绘制,Controller 包负责与后端逻辑网络的数据传输进行配合，并将网络数据传输过程中的数据流动绘制在窗口上。流程如下图所示.



3.4.1 视图模型

在此次的设计中主要有三种视图模型，网络节点视图、信道视图、数据点视图，位于程序的 mode 包中。

3.4.1.1 网络节点视图类(NetNode)

NetNode 类
- String name - int x - int y - int r - boolean isFIll - Color color - LogicalNode logicalNode;
+ String getName() + Color getColor()

网络节点的制图模型中记录了节点所需要绘制的坐标、半径、颜色等信息。程序初始化其实体化对象时会通过 logicalNode 属性与后端逻辑节点相关联，之后根据其记录的信息进行网络节点绘制。

3.4.1.2 信道视图类

Line 类
- NetNode form - NetNode to - Color color - float width = 4.0f - boolean isBrokenLine
+ void setColor(Color color) + Color getColor() + Line(NetNode form, NetNode to)

信道视图类描叙了信道的链接。其中 form 和 to 两个属性表示一条信道所链接的两个网络节点,color 和 width 记录信道绘制时的颜色和宽度。isBrokenLine 表示是否绘制虚线。在程序中实线表示树信道，虚线表示短接信道。

3.4.1.3 数据视图类

DataNode 类
- double x - double y - int r - NetNode form - NetNode to - Color color - boolean isMoving - Data data - float speed
+ void setMove(NetNode form, NetNode to) + void move() + boolean isMoving()

数据视图类描述数据可视化的显示信息，以及控制数据的移动。x, y, r, color 分别描述了数据点的坐标、半径和颜色。form 和 to 记录实例后数据点对象当前正在哪两个网络节点之间移动以及移动的方向。data 用于关联后端数据点，isMoving 记录数据视图点是否在移动中。setMove 方法用于设置数据点的移动起点和终点，move 方法控制数据视图点的移动，isMoving 方法用于判断数据视图点是否在移动之中。

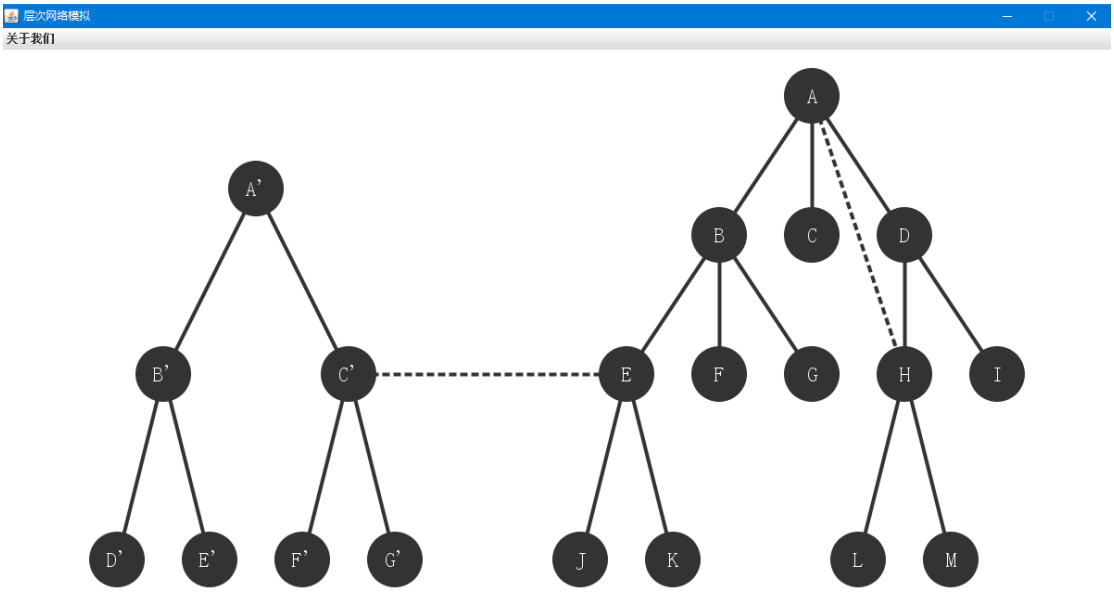
3.4.2 数据点的流动控制

3.4.2.1 可视化处理过程

采用单独地线程进行图像的绘制，数据传输过程中每 20ms 重绘一次图像实现动画效果。如果数据点在两节点传输过程中直接控制数据点移动即可。当数据点移动到某一节点之中，即数据从上一个逻辑结点传输到了当前逻辑结点，此时由后端算法计算返回数据点所需要传输到的下一个结点，然后再控制数据点移动。重复以上步骤，直到数据点传输到目的结点。

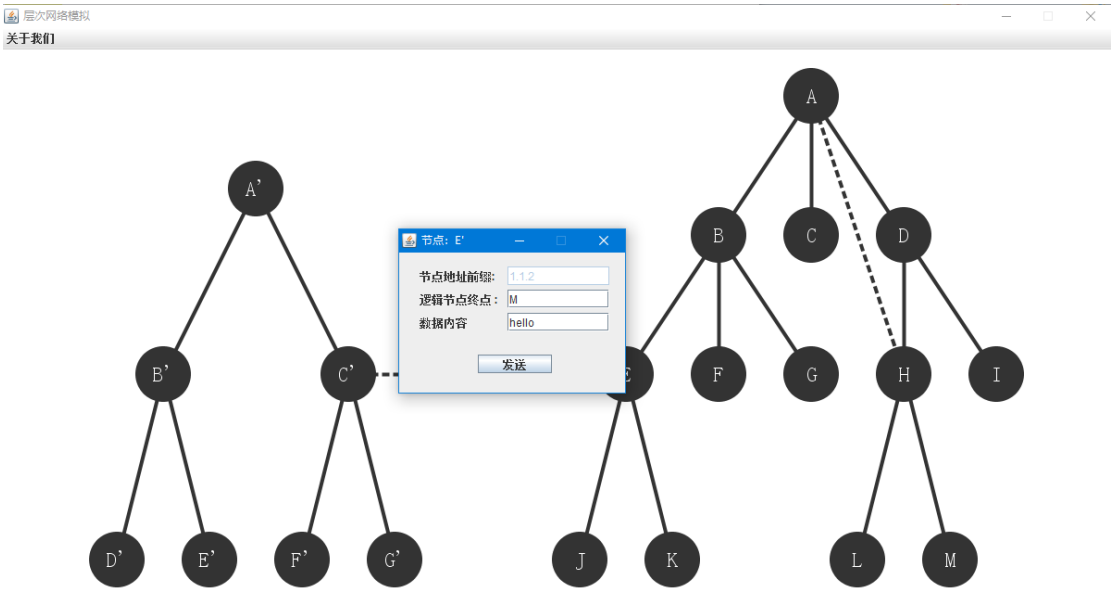
4. 实验结果

4.1 层次网络拓扑图



由上图可见网络节点的名称，连接关系，层次关系等，图中实线表示树信道，虚线表示短接信道。

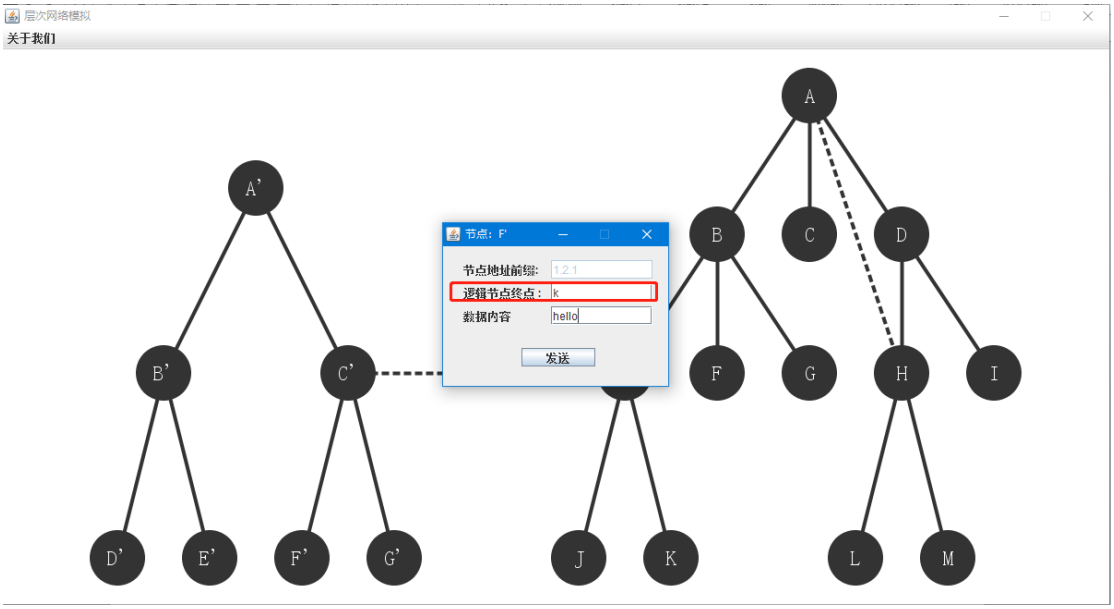
4.2 基本操作



点击任意一个节点可弹出对话框，可查看该节点的地址前缀，选择发送信息的目的节点，以及填写发送的内容，填写完成后点击发送即可开始发送数据。

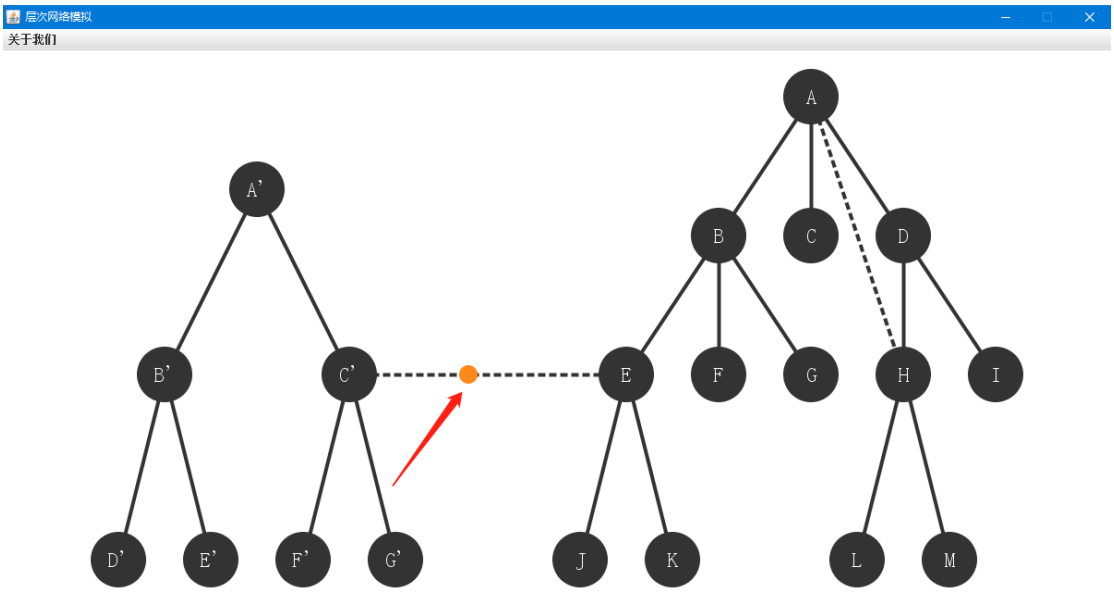
数据流动传输的过程会在图上动态的展现出来。

4.3 短接问题-短接隧道

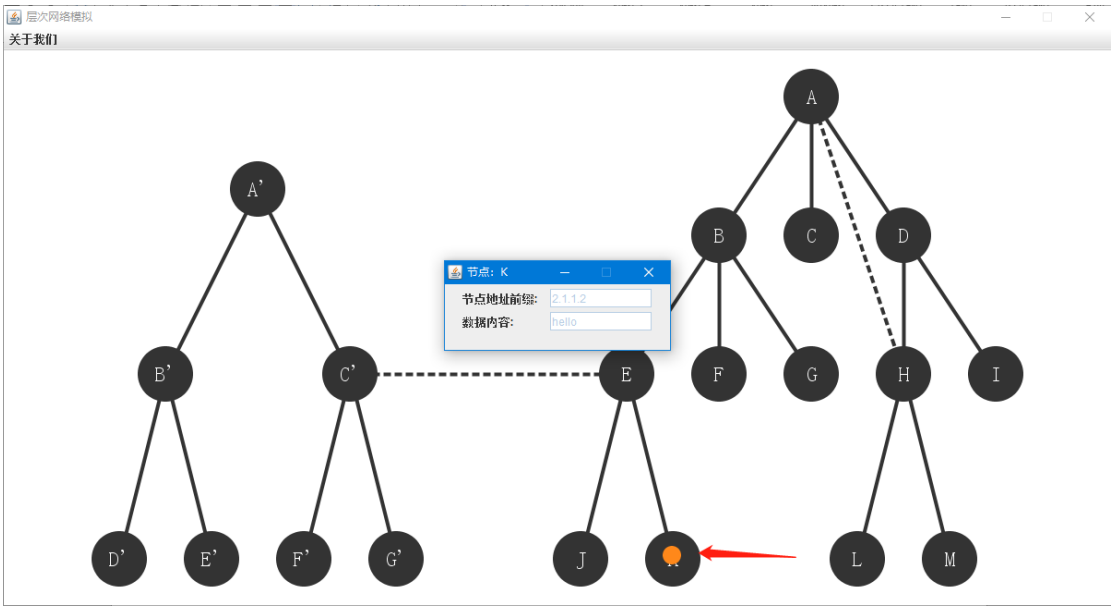


点击 F’，设置发送终点为 K，点击发送。

4.4 传输过程

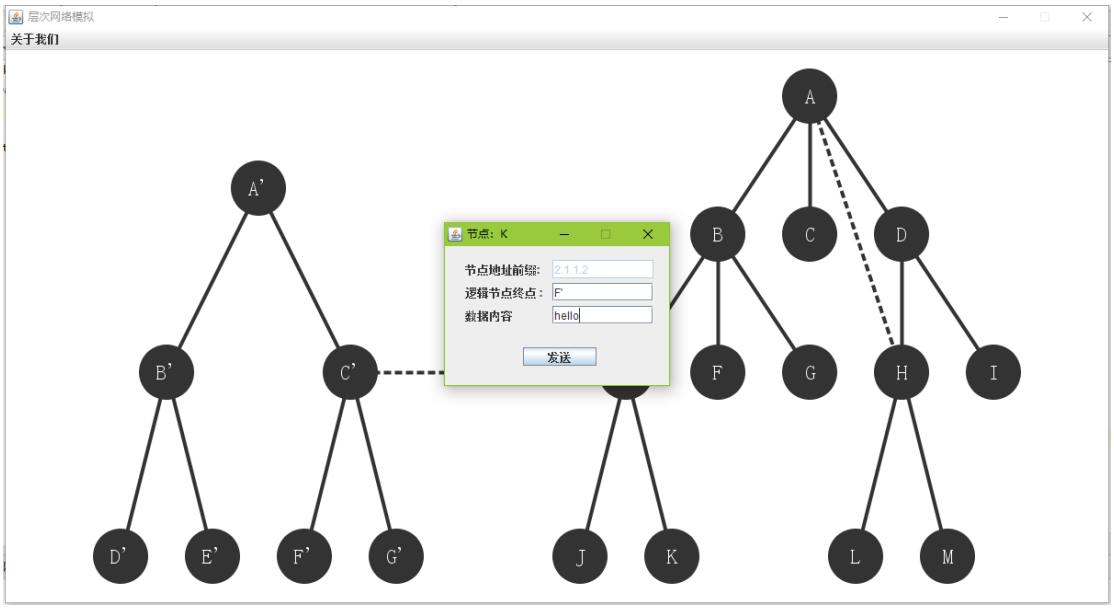


4.5 数据包到达骨干网终点



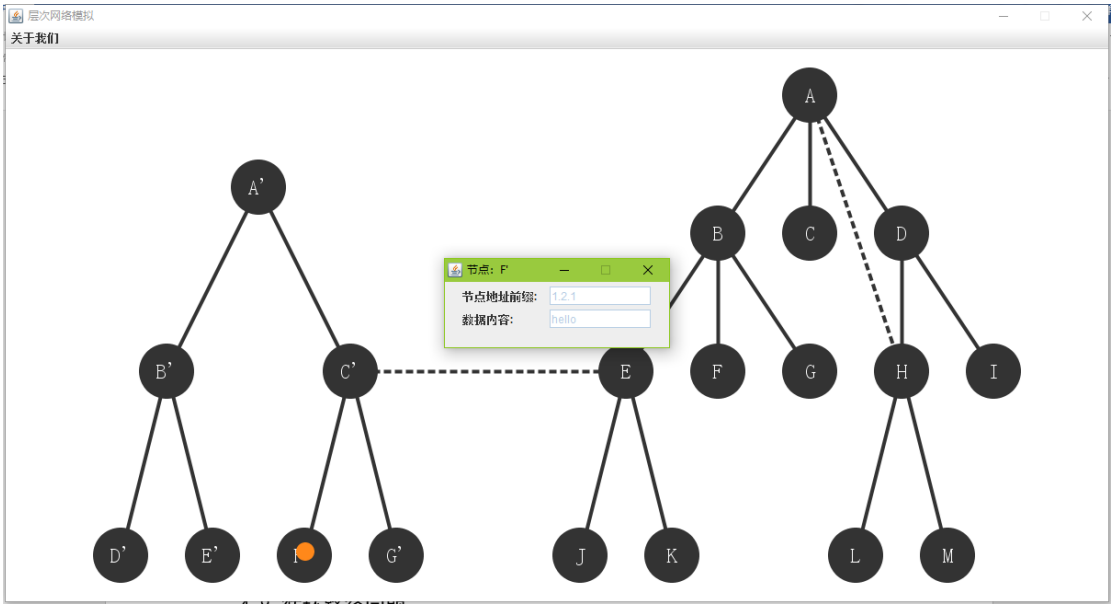
节点 K 收到数据包，骨干网数据转发成功

4.6 反向发送数据包



点击 K, 设置发送终点为 F’, 点击发送

4.7 反向数据包到达骨干网终点



传输成功, F’收到数据包, 隧道问题解决

4.8 重复路径

```
start F’-->C’-->E-->J
start J-->E-->C’-->F’
```

上图可知, F’发送数据到 J，然后又从 J 传送数据到 F’的路径记录，可以看到往返路径完全相同没有走多余的路径，重复路径问题得到解决。

4.9 循环路径问题

```
start L-->H-->M
start M-->H-->L

start L-->H-->D-->A
start A-->H-->L
```

如上图所示数据的发送分别是从 L 到 M、M 到 L、L 到 A、A 到 L。可以看到在设置单向短接信道后，虽然 L 到 A 的往返路径会不一致，但是解决了循环路径问题。

5 总结

计算机网络体系结构是一门理论性比较强的专业课，许多理论知识只有通过实验验证才能加深理解并真正掌握。在短接技术大作业中我们通过查阅相关资料与小组讨论得出了三种问题的解决方案，在实验过程中对层次网络体系结构有了更深入的理解，对短接技术有了深刻的体会，通过软件模拟层次网络的数据交换过程,锻炼了研究和实践能力.

在实验过程中,我们在第三个问题的分析上出现了一些分歧,由于数据包回路的产生需要一定条件,我们最开始默认认为两个直接短接节点都相互配置短接表,因此无法重现拓扑图中 L-A-D-H-A-D-H…回路现象.经过小组深入的讨论我们终于清楚了短接回路产生的原因和条件,并在此基础上提出我们自己的解决方案,最终解决短接回路问题.

我们在实现基本功能的基础上,添加了 GUI 功能,同时提出了一些我们自己的解决方案,这可能不是最完美的解决方案,但是经过我们小组的共同努力最终完成了本次大作业,体会到了这门课程的意义.