1. **问题介绍：**

本节主要介绍了本项目需要解决的问题。将该实际问题转化为优化问题进行求解。我们需要寻找满足时延、丢包率和抖动三个约束的最小花费路径，建模如下：

其中，x表示两个节点间的所有路径；分别表示时延、丢包率和时延抖动量三个约束。

该项目同时要求算法实现如下功能：

**必选节点**：某数据流的传输要求必须经过节点，这些节点的先后顺序是事先给定的。

**必选链路**：某数据流的传输要求必须经过某些链路，这些链路的先后顺序是事先给定的。

**分流：**当前网络剩余带宽不能满足该数据流的分配时，根据数据流可分的最小拆分单位，将数据流拆分后再分配

**优先级抢占：**当前数据流无法被分配时，可以通过将当前网络上已经分配的低优先级的数据流释放掉，从而达到带宽要求来分配当前数据流。

在算法中我们需要注意如下的一些说明：

1. 数据流的属性：

KeyVec是一个K维的向量，K表示数据流的必经节点和必经链路总数，向量中的元素是1或者2分别表示含必经节点或必经链路；例如，KeyVec=[1,1,1,2,2] 表示该数据流有3个必经节点，2条必经链路。

2. 建模时需要把约束：时延、丢包率、抖动按照约束的由硬到软排列，也就是最严格的约束作为f1, f2次之，f3可以被一定程度地违反。

3. 约束说明：

；

；例如：路径上的时延总和 <= 时延阈值，

b) 丢包率约束的说明：路径的时延和抖动均是求和运算，而丢包率的

计算是1-各边到包率的乘积。因此为了算法设计的需要，这里对边上的到包率属性稍作变形这样路径总的丢包率的刻画即为：

这样到包率的乘积运算就可以转化为相应的取对数后的求和运算。

**二．前期算法**

本节主要介绍我们在项目初期的一个算法，该算法的基本思想是首先判断当前网络是否带宽可行🡪然后判断是否有满足约束的路🡪最后求解满足约束的最小花费的路径。 在此过程中，如果没有满足约束的路径，我们通过改变约束的阈值能够找到一条违反约束程度最小的路径。

通过该算法我们得到了一个事实，那就是当约束比较严格的时候，也就是没有满足约束的路径的时候，我们更改约束中的阈值（最小程序的更改阈值）后，通过该算法求解得到的满足更改后的约束的最短路，实际上就是我们在求解过程中判断出约束不可行的那条路径。这一事实为我们后期设计最终算法提供了一个依据，也就是这种情况下，我们根本不需要再启用求解模块，直接输出当前的最短路。该路就是违反约束最小的路径。

**算法流程图：**

**在更新的拓扑图上分配**

**带宽为M的数据流（A->B）**

* 1. **若含必经链路，判断链路带宽可行;**

**1.2）判断拓扑图带宽是否可行；**

**a）抢占优先级低的数据流带宽**

**b) 若数据流可分，分流当前数据流；**

**否则转至a) 抢占其他数据流；**

**对分流的数据流依次分配**

1. **判断是否带宽可行—算法1**

**否**

**是**

**否**

**是**

**2）**

**判断**

**是否**

**约束**

**可行**

**算法2.3**

**判断**

**算法2.2**

**判断**

**算法2.1**

**< 0**

**判断**

**是**

**是**

**更新**

**否**

**更新**

**否**

**否**

**3)**

**求**

**解**

**是**

**更新**

**算法3**

**判断是否带宽可行——**

**算法1**

S1. 如果数据流的传输要求含必经链路（ ）,判断必经链路上的剩余带宽是否大于M

若是，goto S2； 否则，返回 —1（抢占）；

S2. 删除当前网络图上剩余带宽小于M的边，记新图为 Gnew；

for k=0: K

判断节点Node k 到 Node k+1 是否连通；

若否，返回 —2（分流）

k=k+1；

end

返回 1（带宽可行）

*说明：Node 0 就是起点A，Node K+1 就是终点B；*

S3. 抢占：

从1到K的顺序对必经链路以此判断剩余带宽是否可行：对每条必经链路所涉及的数据流按照优先级

低开始逐一释放，直到该必经链路剩余带宽可行；

for k=1: K

if 第k条必经链路剩余带宽<M，

更新图；

将流经第k条必经链路的已分配的数据流按照优先级低逐个开始释放；

end

k=k+1;

end

goto S2;

S4. 分流: 根据数据流的可拆分最小单位对数据流拆分，若用户没有指定拆分方案，提供的缺省拆分方案是按最小单位拆分为若干条数据流；对拆分否的数据流依次重新调用整体算法。

**2）判断是否约束可行——**

**算法2.0 ：计算必经链路上的时延、log（1/到包率）、抖动以及花费的分别和记为：bj1，bj2，bj3，bj4：**

如果数据流的传输要求含必经链路（if ）,

令bj1=0；bj2=0；bj3=0；bj4=0；

if K>0

for i=1:K

if KeyVec(i)==2

bj1= bj1+ 第i条链路的时延；

bj2 = bj2 + log(1/第i条链路的到包率)；

bj3= bj3+ 第i条链路的抖动；

bj4= bj4+ 第i条

end

end

end

输出 bj1 ,bj2, bj3, bj4.

**算法2.1**

计算

Step0. 全局变量： MaxN,eps ,KeyVec,K; bj1, bj2, bj3, bj4。

Step1. 初始化: 输入初始目标函值cf；

Step2. f=cf;

for k =0：K+1

计算 Node(k) 到Node(k+1) 之间的最短路；最优值记为fk；

f=f+fk;

end

S3. 输出, 以及最优值f.

**算法2.2**

**计算**

**作**

**转化为 利用算法2.1 求解 (2.2 - 1)**

Step 0. 全局变量： bj1, bj2, bj3, bj4；

Step 1. 初始化 令；令初始最优解x\*=x1\* , 输入初始目标函值cF；

Step 2. 令 取初始函数值 cf=作为算法2.1的初始目标值输入， 用算法2.1 求解（2.2-1）得

if 输出，退出；else

Step 3. 若, 输出x\*, 退出；

Step 4. 否则取 取初值cf=作为算法2.1的初始目标值输入，用算法2.1求解（2.2-1）得到

**算法2.3 ：**

**计算**

作 ;

转化为 利用算法2.2 求解

Step 0. 全局变量： bj1, bj2, bj3, bj4；

Step 1. , 令初始最优解 x\* = x2\* ，

输入初始目标函值cG

Step 2. 令 取初值作为算法2.2的初始目标值输入; 用算法2.2求解（2.3-1）得到最优解,

if 输出；else

Step 3. 若 ，输出**最优解 后退出**；

Step 4. 否则取 取初始函数值 作为算法2.2的初始目标值输入; 用算法2.2求解（2.3-1）得

**判断 ?**

**算法2.4：**

作 ; 转化为求解

**算法2.4：**

Step 0. 全局变量： bj1, bj2, bj3, bj4; ， **，；**

Step 1. 初始化： ，令初始最优解 x\* = x3\* (x3\*是算法2.3求得的最优解); 令目标函数的初始值 cH=bj4.

Step 2. 取 取初始值cG = 作为算法2.3的初始目标值输入;

用算法2.3 求解（3-1）得到最优解, if ,令，输出x\* 退出；

else

Step 3. 若 输出 后退出；

Step 4. 取 初始函数值cG = 作为算法2.3的初始目标值输入;

用算法2.3求解（3-1）得

**主算法3:**

**注1：为书写方便， 有时侯用 g1(x), g2(x), g3(x)表示；有时侯用T1，T2, T3表示。**

**算法3：**

1. 全局变量赋初值： 输入初始的约束上限值T1，T2，T3， 输入初始链路属性值bj1=bj2=bj3=bj4=0.
2. 令约束上限值的改变记录值J1=J2=J3=0，.
3. 调用算法2.0，计算更新全局变量bj1, bj2, bj3, bj4 ;
4. 取f(x)=g1(x),cf=bj1，调用算法2.1得到最优解X1\*和最优目标值f\*。 将X1\*设置为全局变量，先计算更新J1=max{0, f\*-T1+eps1}, 后计算更新全局变量T1=T1+J1。
5. 取F(x)=g2(x), cF=bj2，调用算法2.2得到最优解X2\*和最优目标值F\*。将X2\*设置为全局变量，先计算J2=max{0, F\*-T2+eps1}, 后计算更新全局变量T2=T2+J2。**注意:** 此处实际调用算法2.1时的目标函数为f(x)=g2(x)+g1(x), 初始目标值cf=bj2+bj1。
6. 取G(x)=g3(x), cG=bj3，调用算法2.3得到最优解X3\*和最优目标值G\*。将X3\*设置为全局变量，先计算J3=max{0, G\*-T3+eps1}, 后计算更新全局变量T3=T3+J3。

**注意:** 此处实际调用算法2.1时的目标函数为

f(x)=g3(x)+g2(x) ]+g1(x),

初始目标值bj3+bj2(x) ]+bj1 .

1. 取H(x)为链路费用函数，调用算法2.4得到最优解X4\*和最优目标值H4\*，获得最优费用链路流。

**注意:** 此处实际调用算法2.1时的目标函数为

f0(x) + v g3(x)] +g2(x) } +g1(x),

初始目标值cf=bj4 + v bj3(x)] +bj2(x) } +bj1(x).

1. 求出网络图各边费用的最大值MF，对X4\*上每条边的费用为原费用加上MF。取H(x)为链路费用函数，调用算法2.4得到最优解X5\*和最优目标值H5\*，将X4\*上每条边的费用降为原费用后重新计算H5\*，获得最优费用备用链路流。

**三．改进算法**

本节介绍另一个求解满足约束的最小花费的路径算法。本算法是在前一节算法的基础上改进的。该算法的特点是在数据流在分配时，如果需要抢占低优先级的数据流，那么本算法能够保证尽量少抢占数据流，也就是能够保证抢占的数据流的数量尽可能少。同时本算法的求解时间较之上一节的算法大大减少了。下面详细介绍本算法：

算法设计思路：

1. 执行该算法之前，待分配的数据流若能不怎么损失传输质量能够分流的先行分流；
2. 我们不得已才会去分流，也就是当前网络图上释放掉比它优先级低的数据流后，剩余带宽还不能满足该数据流，只能分流解决；
3. 所以，在无法直接分配数据流的时候，我们优先抢占；
4. 首先，我们假性释放优先级低于当前数据流的所有数据流，判断带宽是否可以满足，若不满足说明即使真的抢占所有低优先级的数据流也不能分配该数据流，此时启动分流模块；若带宽满足，执行求解最短路模块；
5. 求解最短路模块中，我们采用经济杠杆自动调节，使得抢占的数据流尽量少，具体的：所有优先级低的数据流所选择的路，每条链路的占用部分的带宽花费增加1个单位（该单位是所有链路的最大费用记为maxcost），这样做的结果是拓扑图上每个链路的cost是个阶梯价格，那么为了降低花费，算法自动会避开那些热点的链路，从而达到被抢占数据流尽量少。
6. 求解得到解以后，判断该解是否满足约束，若满足，输出结果；若不满足，由于我们前期的大量数值试验以及双方讨论的结果，这时候当前的路径就是违反约束最小的路径，因此输出该路径，并输出该路径时延、丢包率、抖动三个属性（也就是对于不满足的约束，阈值需要调整为的数值）。
7. 求出解后，真正被抢占的那部分数据流，加入到待处理队列P中，注意该队列P是优先级高在前低在后；
8. 继续处理队列P中的数据流，直到P为空，恢复为被抢占的数据流；
9. 分流模块：采用最小拆分粒度的二分法拆分数据流。下面给出详细的分流算法：

**分流算法：**

S0. 对每个数据流flow增加属性qiujie，flow.qiujie=0表示该流并非所求数据流（既为被抢占的数据流），flow.qiujie=1表示该流为所求数据流（源求解数据流或其分流产生的流）。既所有flow.qiujie=1的数据流的起点终点相同，约束条件相同（带宽可能不同）。对每个数据流flow增加属性div表示已分流次数，初始为0队列Q表示所有待求数据流，初始为待分配的新数据流

S1. 队列Q的队首元素作为当前处理数据流，记作M（同时出队）若Q为空，退出程序，输出结果

S2. 假释所有低于M优先级数据流

S3. 判断此时M带宽是否满足（能找到一条满足带宽约束的路径），若不满足，记录fenliu\_flag=1用分流解决。若满足fenliu\_flag=0直接求解。

S4. 若fenliu\_flag=0，M不分流可找到解，直接求解，并分配路径

S5. 若fenliu\_flag=1，M需要分流。恢复所有假释数据流，判断M.bandwidth/2是否大于最小粒度，若不满足输出“已满足最小粒度，无解”退出程序。若满足，判断M.div是否大于最大分流次数，若不满足输出“已超过最大分流次数”退出程序。构造新的数据流t1=M,t2=M,且进行带宽分流t1.bandwidth=M.bandwidth/2; t2.bandwidth=M.bandwidth-t1.bandwidth

并令t1.div++;t2.div++增加已分流次数。Q=[Q t1 t2]，返回步骤S1

S6. 恢复假释数据流，对不能恢复的数据流加入队列Q。（同时其删除在数据流列表上原有的信息，防止已被抢占在外的数据流再次被抢占），返回步骤S1

**算法流程图：**

带宽为M的数据流到达

定义待处理数据流队列P=[M]

假释放全部优先级低的数据流

更新待处理的数据流队列P

优先级由高到低的顺序

取P中第一条数据流，记为X

P是否为空

带宽满足？

求解满足约束的最短路

输出该路径以及相应的约束值，删除P中的X

被抢占的数据加入P中

约束满足？

抢占发生？

恢复未被占用数据流 stop

输出结果

删除P中的X

分流：二分的数据流加入到P，

直到**分流到第三层**或者**分到最小粒度**，带宽不满足，输出分流及其是否分配成功等情况stop

恢复未被占用数据流 stop

是

是

是

否

是

否

否

否