

B4: Google's Software-Defined WAN

Paper Reading

Log Creative

2021 年 11 月 7 日

论文

Chi-Yao Hong et al. “B4 and after: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google’s Software-Defined WAN”. In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM ’18. Budapest, Hungary: Association for Computing Machinery, 2018, pp. 74–87. ISBN: 9781450355674. DOI: 10.1145/3230543.3230545. URL: <https://doi.org/10.1145/3230543.3230545>



B4

Google 私有广域网后端

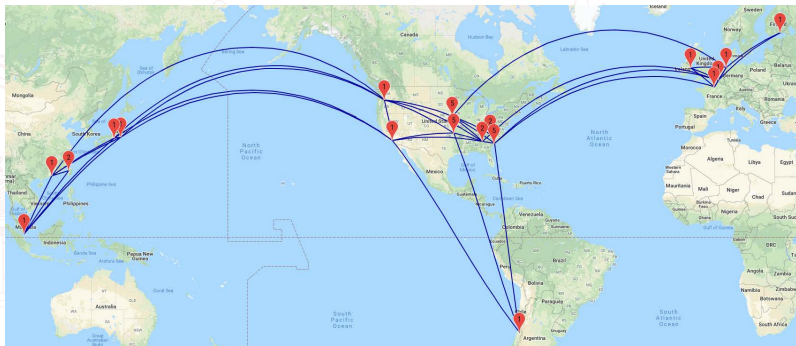
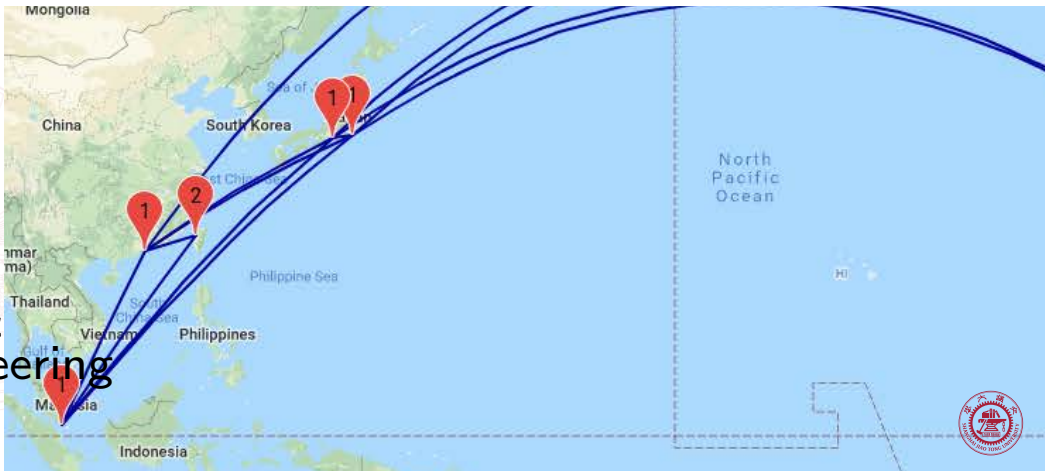


图: B4 全球网络



B4

Google 私有广域网后端

TE
Traffic
Engineering

SLO

Service Level Objectives 服务级别协议

表示 30 天滑动窗口内的网络连接可用性和带宽可用性。

服务级别	应用举例	SLO需求
SC4	搜索广告、DNS、WWW	99.99%
SC3	照片服务后端、邮件	99.95%
SC2	广告数据库拷贝	99.90%
SC1	搜索索引拷贝	99%
SC0	批量传输	

表: SLO



SLO

Service Level Objectives 服务级别协议

表示 30 天滑动窗口内的网络连接可用性和带宽可用性。

服务级别	应用举例	SLO需求
SC4	搜索广告、DNS、WWW	99.99%
SC3	照片服务后端、邮件	99.95%
SC2	广告数据库拷贝	99.90%
SC1	搜索索引拷贝	99%
SC0	批量传输	

表: SLO



SLO

Service Level Objectives 服务级别协议

表示 30 天滑动窗口内的网络连接可用性和带宽可用性。

服务级别	应用举例	SLO需求
SC4	搜索广告、DNS、WWW	99.99%
SC3	照片服务后端、邮件	99.95%
SC2	广告数据库拷贝	99.90%
SC1	搜索索引拷贝	99%
SC0	批量传输	

表: SLO



扁平结构

不利于扩展和可用性

之前的 B4 若想增加容量，需要在地理限界内增加站点。但这会带来：

- ① 增加了中央流量控制优化算法的运行时间。
- ② 对交换机有限的流表空间增加压力。
- ③ 使得容量管理变得复杂并给应用开发者造成麻烦。



扁平结构

不利于扩展和可用性

之前的 B4 若想增加容量，需要在地理限界内增加站点。但这会带来：

- ① 增加了中央流量控制优化算法的运行时间。
- ② 对交换机有限的流表空间增加压力。
- ③ 使得容量管理变得复杂并给应用开发者造成麻烦。

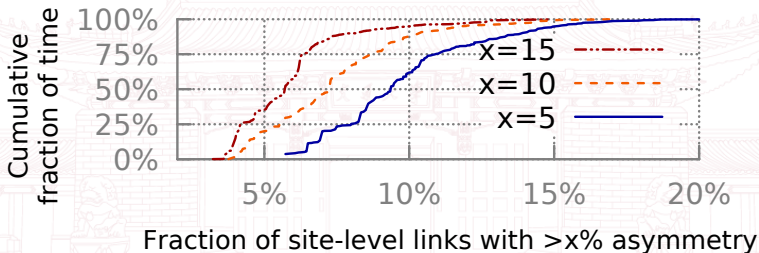
为了解决这个问题，引入 *supernode*（超级节点）和两层架构。



分层架构

容量不对等问题

B4 中 6–20% 的地理级连接仍然会在 $\geq 5\%$ 的时间内有容量不对等情形。



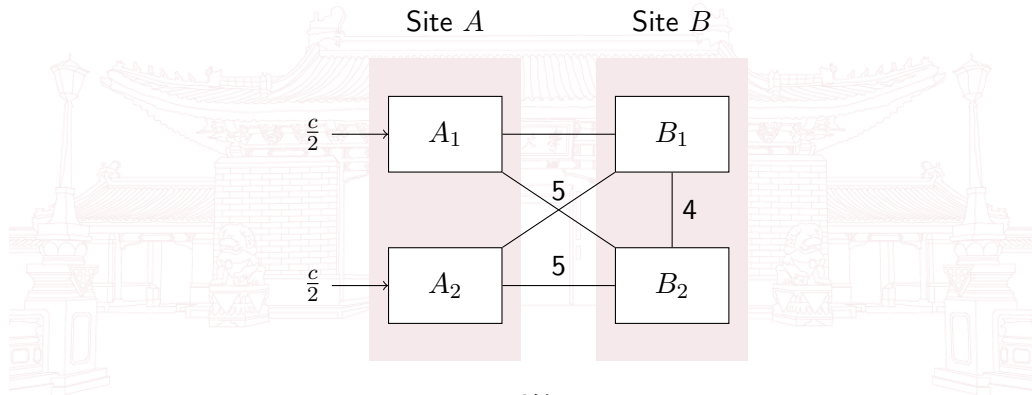
$$\frac{\text{avg}_{\forall i} C_i - \min_{\forall i} C_i}{\text{avg}_{\forall i} C_i}$$

图：地理级流量不对等



不对等的后果

大幅减少系统效率

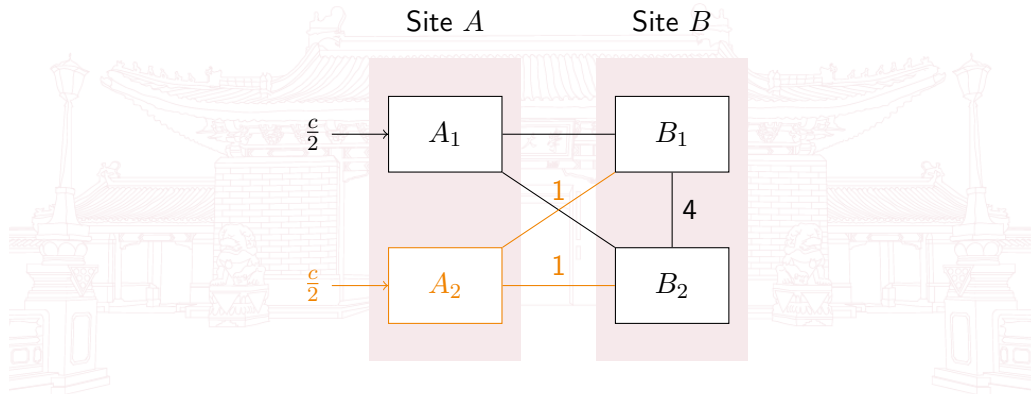


图：对等



不对等的后果

大幅减少系统效率

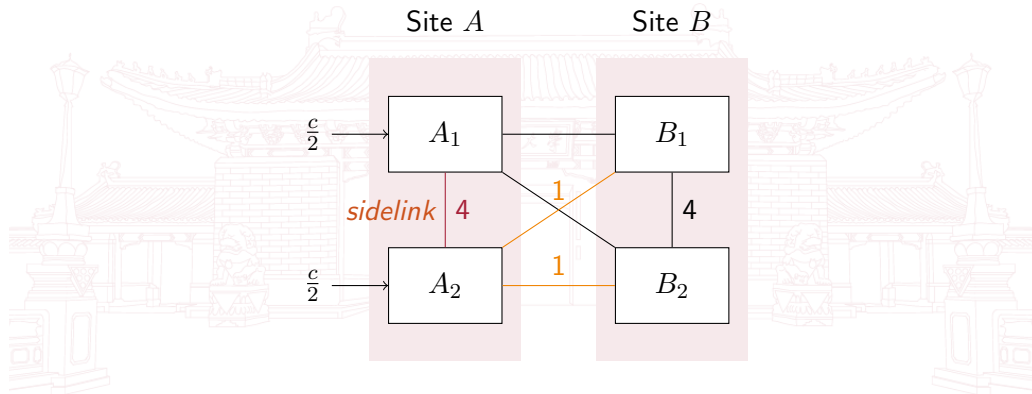


图：不对等示例 $c = 4$



不对等的后果

大幅减少系统效率



图：不对等示例 $c = 12$



使用 *sidelink* 可以提高不对等时的带宽利用率。但是仍然需要考虑相关的协议问题，比如有些数据不可分割、MAC 地址不可变化，后文通过预留MAC地址解决。以及死循环问题，转换隧道可能是原子操作，以任意顺序应用 TE 更新会导致这种死循环率上升，后文通过TSG排序算法解决。



高效交换规则管理

Merchant 交换机只支持有限的匹配和哈希规则。



Saturn

第一代 B4 网络结构

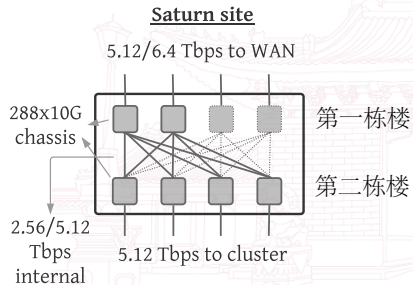


图: Saturn 站点

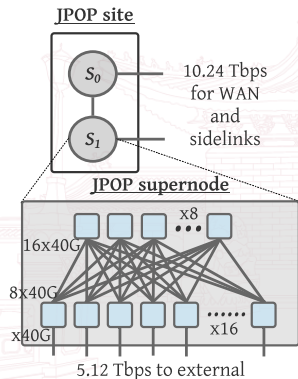
名称	Saturn
部署年	2010
类型	数据中心
交换机芯片	24x10G
每站点机箱数	6 / 8
站点容量 (Tbps)	5.12 EX 2.56 INTER
每站点交换机箱数	4
控制域数量	1

表: Saturn 站点



Jumpgate: JPOP

仅传输站点



名称	JPOP
部署年	2013
类型	POP
交换机芯片	16x40G
每站点机箱数	20
超级节点交换机数	24
站点容量 (Tbps)	10.24
每站点交换机箱数	4
控制域数量	2

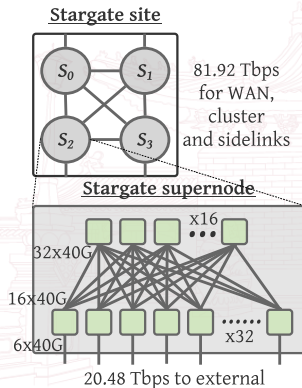
表: JPOP 站点



图：JPOP 站点

Jumpgate: Stargate

数据中心级



名称	Stargate
部署年	2014
类型	数据中心
交换机芯片	32x40G
每站点机箱数	192
超级节点交换机数	48
站点容量 (Tbps)	81.92
每站点交换机箱数	8
控制域数量	4

表: Stargate 站点

图: Stargate 站点



Jumpgate: Stargate

数据吞吐量大带来的好处



图: 交换机与交换机架

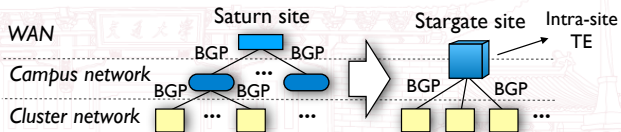


图: 减少 BGP 复杂度



Facebook Outage

BGP 相关新闻

Facebook 在 10 月 4 日爆发了服务大中断，其他网络无法与之形成正确的连接。

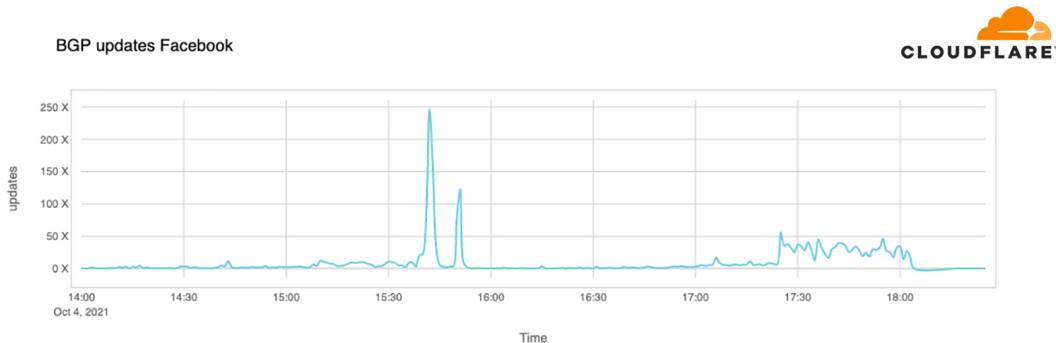


图: BGP 变更高峰¹

¹Celso Martinho and Tom Strick. *Understanding How Facebook Disappeared from the Internet*. Tech. rep. Cloudflare, Oct. 2021. URL: <https://blog.cloudflare.com/october-2021-facebook-outage/>

简单粗暴的提案

平面流量工程

方法 直接对站点内所有的超级节点直接应用流量控制。这种模型下，由一个中央控制器使用 IP-in-IP 封装对超级节点级隧道进行负载均衡。

缺点 高运行时间、花费大量的交换机流表空间、不可扩展。每个站点内有4个超级节点，那么站点到站点间的3跳转发会有 $4^3 = 64$ 条路径。



简单粗暴的提案

最短路转发

方法 超级节点级链路实现最短路转发。

优点 拥有扩展性、只需要一层封装、在广域网失效时能够通过**旁路链接**完成流量转移。

缺点 无法处理容量不对等情形，不是完全失效的情况下无法找到通过**旁路链接**得到的更长但容量更大的路径。

▶ 不对等的后果



分层流量工程结构

概念

SSG *Switch Split Group* 确定物理交换机所分割的流量。

TSG *Tunnel Split Group* 确定一个链路 *tunnel* 内如何分配两个站点超级节点间的流量分布。

TG *Tunnel Group* 通过 IP-in-IP 封装映射 FG 到一个链路集合。

FG *Flow Group* 〈源站点, 目标站点, 服务类别〉

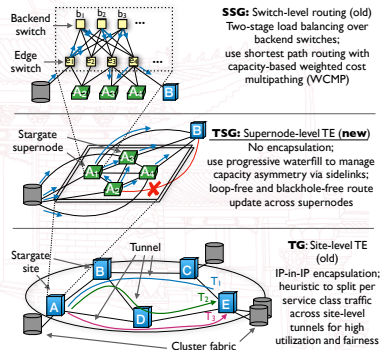


图: 不同的流量工程结构



分层流量工程结构

运行概览

Algorithm 1: B4 运行概览

- 1 域控制器通过聚合可用的物理链路容量来计算超级节点间的连接;
 - 2 中央控制器根据上述结果计算 TSG 来分配每一条站点级的出口连接;
 - 3 **if 对等 then** 不使用旁路连接;
 - 4 **else** 通过旁路连接重新分配TSG;
 - 5 使用上述 TSG 结果计算站点级每条链路的有效容量, 生成 TG;
 - 6 生成 TE 操作的无环依赖图;
 - 7 通过生成 SSG 分割规则, 按照次序对 FG, TG, TSG 编程;
-



TSG 生成算法

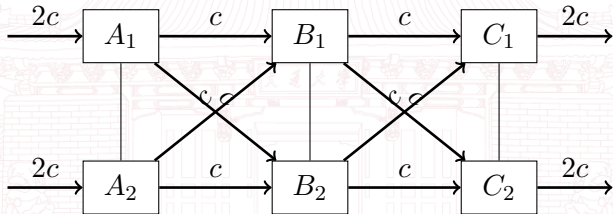
问题描述

假设进入隧道的流量对于源站点的所有超级节点来说都是平均分割的。对于沿着隧道的每个站点内的每一个超级节点计算 TSG，使得能够最大化利用对应超级节点连接容量的上限。使用正整数表示输出链路上的每一连接的相对权重，并要求每一个 TSG 的权重总和不能超过一个阈值 T ，因为交换器哈希表键值数量限制。



TSG 生成算法

例子

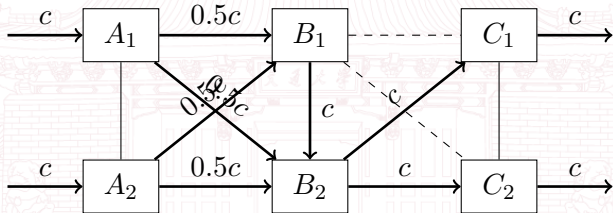


图：对等网络 $TC = 4c$



TSG 生成算法

例子

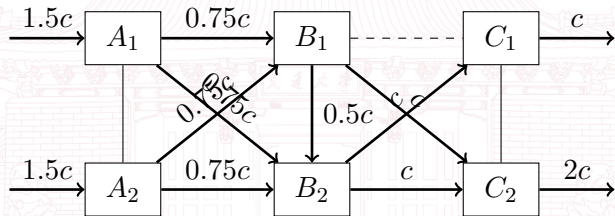


图：不对等网络 $TC = 2c$



TSG 生成算法

例子



图： 不对等网络 $TC = 3c$



TSG 生成算法

例子（罕见）

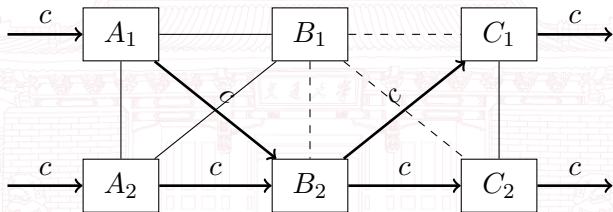


图: $TC = 0$ (B_1 故障)² 或 $TC = 2c$ (B_1 关闭)

TSG 生成算法

例子（罕见）

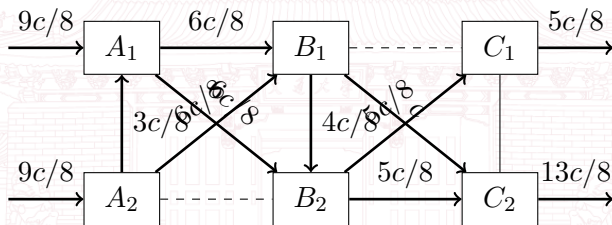


图: $TC = 9c/4$ (次优解)



TSG 生成算法

例子（罕见）

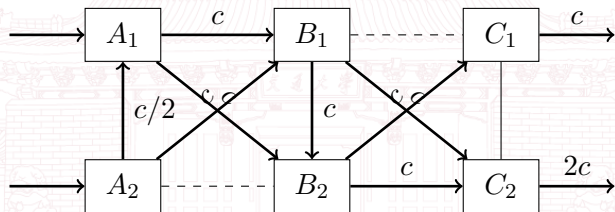
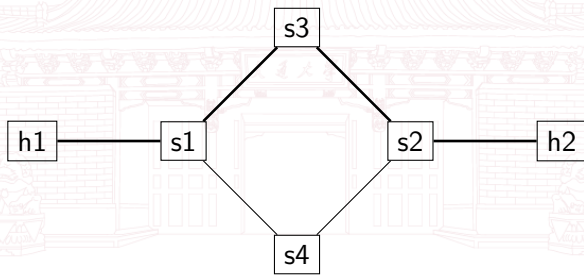


图: $TC = 3c$ (最优解)



不关闭的后果

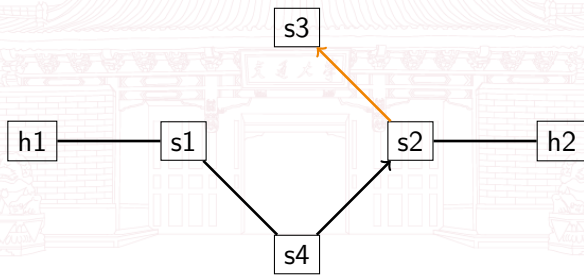
最近的一次 lab 含有环路的网络，代码写的不好的话，如果切断一个链路，而没有告知另一侧的路由，将会导致另一侧的路由仍然会向错误的链路发包，导致不可达。



图：正常链路

不关闭的后果

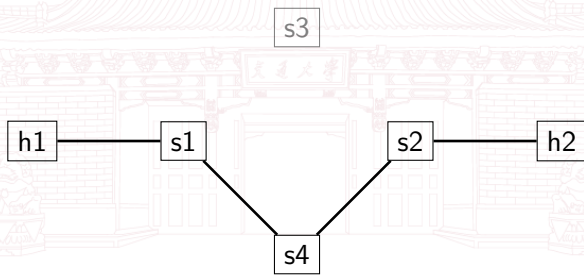
最近的一次 lab 含有环路的网络，代码写的不好，如果切断一个链路，而没有告知另一侧的路由，将会导致另一侧的路由仍然会向错误的链路发包，导致不可达。



图：不可达

不关闭的后果

最近的一次 lab 含有环路的网络，代码写的不好的话，如果切断一个链路，而没有告知另一侧的路由，将会导致另一侧的路由仍然会向错误的链路发包，导致不可达。



图： 关闭后

网络流算法

回顾

Algorithm 2: Augment(f, c, P)

```
1  $\delta \leftarrow$  bottleneck capacity of augmenting  
   path  $P$ ;  
2 foreach  $e \in P$  do  
3   if  $e \in E$  then  
4      $f(e) \leftarrow f(e) + \delta$ ;  
5   else  
6      $f(e^R) \leftarrow f(e^R) - \delta$ ;  
7 return  $f$ ;
```

Algorithm 3: Ford-Fulkerson Algorithm

Input: $G = (V, E), c, s, t$

```
1 foreach  $e \in E$  do  
2    $f(e) \leftarrow 0$ ;  
3  $G_f \leftarrow$  residual graph;  
4 while there exists augmenting path  $P$   
   do  
5    $f \leftarrow$  Augment( $f, c, P$ );  
6   update  $G_f$ ;  
7 return  $f$ ;
```

TSG 生成算法

算法描述

对于每一个有向站点级链接作为独立的网络流问题。

$G_{TSG} = (V, E)$ 有向图。包含

- 源站点的超级节点 $S = \{S_i | 1 \leq i \leq N\}$
- 目标站点 D

$C(u, v)$ 边 (u, v) 的容量, $\forall (u, v) \in E$ 。两种边:

- 源站点间超级节点间的连接边 $\forall i, j \neq i : S_i \leftrightarrow S_j$
- 源站点超级节点与目标站点超级节点的连接边 $\forall i : S_i \leftrightarrow D$

PG Path Groups 来满足从源站点每一个超级节点向目标节点的无限需求

- PG_{1-hop}
- PG_{2-hop}



Algorithm 4: TSG 生成算法

Data: $G_{TSG} = (V, E), C(u, v)$

Result: $TSG(u, v)$

```

1   $\forall (u, v) \in E \ C_{\text{Remaining}}(u, v) := C(u, v);$ 
2  repeat
3       $\forall (u, v) \in E \ \text{weight}(u, v) := 0;$ 
4       $\text{frozen\_flow} := \text{NULL};$ 
5      foreach  $S_i \in S$  do
6           $PG_{2\text{-hop}}(S_i) := \{[(S_i, v), (v, D)] | v \in S, C_{\text{Remaining}}(S_i, v) > 0, C_{\text{Remaining}}(v, D) > 0\};$ 
7          if  $C_{\text{Remaining}}(S_i, D) > 0$  then
8               $\text{weight}(S_i, D) += 1;$ 
9          else if  $PG_{2\text{-hop}} \neq \emptyset$  then
10             foreach  $\text{path } P \in PG_{2\text{-hop}}(S_i)$  do
11                 foreach  $(u, v) \in P$  do
12                      $\text{weight}(u, v) += \frac{1}{|PG_{2\text{-hop}}(S_i)|}$ 
13             else  $\text{frozen\_flow} := S_i$  break;           // Stop when any flow failed finding

```

/* More procedure in the next page



Algorithm 5: TSG 生成算法 (续)

```
1 repeat
  /* ... Continued from the previous page */
2  if frozen_flow ≠ NULL then break;
3   $E' = \{(u, v) \in E \mid \text{weight}(u, v) > 0\};$ 
4   $\forall (u, v) \in E' \text{ fair\_share}(u, v) := \frac{C_{\text{Remaining}}(u, v)}{\text{weight}(u, v)};$ 
5   $BFS := \min_{(u, v) \in E'} \text{fair\_share}(u, v);$  /* Bottleneck Fair Share */
6  foreach  $(u, v) \in E'$  do
7     $C_{\text{Remaining}}(u, v) - = BFS \times \text{weight}(u, v);$ 
8  until True;
9  foreach  $(u, v) \in E'$  do
10    $TSG(u, v) = \frac{C(u, v) - C_{\text{Remaining}}(u, v)}{\sum_{(u, v') \in E} (C(u, v') - C_{\text{Remaining}}(u, v'))};$ 
```

贪婪灌水算法通过最大最小公平规则 *Max-Min Fairness* 以迭代分配瓶颈容量。



定理 1

生成 TSG 无环

生成的 TSG 不会产生任何环路。

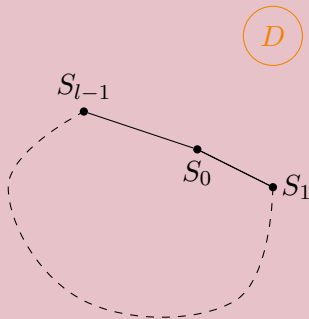


定理 1

生成 TSG 无环

生成的 TSG 不会产生任何环路。

证明



D 是目标节点。
假设生成的 TSG 含有环路，首先这个环路不会包含目标节点。

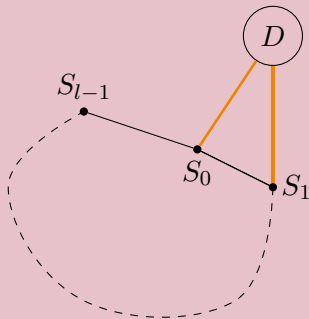


定理 1

生成 TSG 无环

生成的 TSG 不会产生任何环路。

证明



由于寻路算法最多只允许两跳，所以贪婪算法告诉我们，对于 S_0 而言，如果选择了含有 S_1 节点的路径，那么 $\langle S_0, D \rangle$ 一定比 $\langle S_1, D \rangle$ 更拥挤。

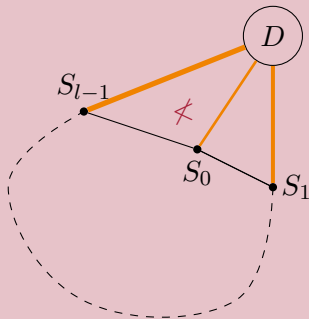


定理 1

生成 TSG 无环

生成的 TSG 不会产生任何环路。

证明



这样进行下去会导致 $\langle S_{l-1}, D \rangle$ 应当比 $\langle S_0, D \rangle$ 更拥挤，但是环形的不等式告诉我们这件事情是矛盾的。□



TSG 排序算法

Algorithm 6: TSG 排序算法

Input: G_{TSG} 和由算法4产生的TSG

Output: TSG更新顺序

- 1 建立依赖图，定理1指示其不会有环路，因此有对于 DAG 的拓扑排序；
 - 2 以反向拓扑排序顺序更新 TSG；
-

该算法与 IGP (Interior Gateway Protocol) 的更新算法类似。



TSG 排序算法

Algorithm 7: TSG 排序算法

Input: G_{TSG} 和由算法4产生的TSG

Output: TSG更新顺序

- 1 **建立依赖图**，定理1指示其不会有环路，因此有对于 DAG 的拓扑排序；
 - 2 以反向拓扑排序顺序更新 TSG；
-

$\langle S_i, S_j \rangle$ 被添加进依赖图中 $\leftarrow \begin{cases} S_j \text{ 是 } S_i \text{ 的下一跳} \\ S_j = D \text{ 而且 } S_i \text{ 向下一个站点传输任何流量} \end{cases}$



定理 2

中间过程不含环路

TSG 升级时，如果原始和目标 TSG 都不包含环路，那么TSG的任何中间过程也都不包含环路。

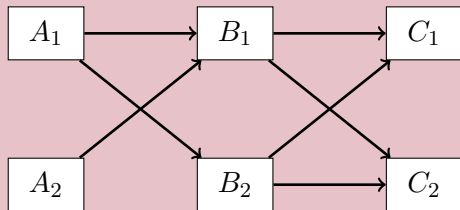


定理 2

中间过程不含环路

TSG 升级时, 如果原始和目标 TSG 都不包含环路, 那么TSG的任何中间过程也都不包含环路。

证明



原始 TSG 不包含环路。

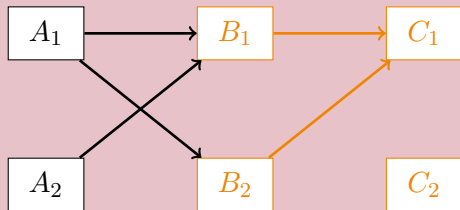


定理 2

中间过程不含环路

TSG 升级时, 如果原始和目标 TSG 都不包含环路, 那么TSG的任何中间过程也都不包含环路。

证明



TSG 中间过程有两种状态的节点:

- 解析过的节点
- 未解析的节点

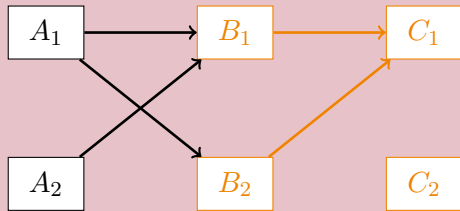


定理 2

中间过程不含环路

TSG 升级时, 如果原始和目标 TSG 都不包含环路, 那么TSG的任何中间过程也都不包含环路。

证明



如果存在环路:

- ✗ 不会只出现在解析过的节点中, 原始TSG不包含环路
- ✗ 不会只出现在未解析的节点中, 目标TSG不包含环路
- ✓ 环路中至少一个解析过的节点和一个未解析的节点

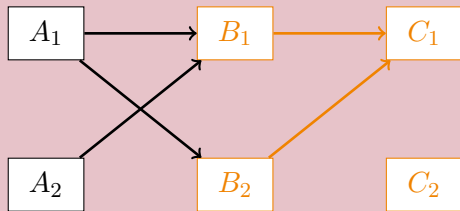


定理 2

中间过程不含环路

TSG 升级时, 如果原始和目标 TSG 都不包含环路, 那么TSG的任何中间过程也都不包含环路。

证明



那么这个环路应当存在由解析过的节点→未解析的节点的链路。然而, 由于使用的是反向拓扑顺序, 这是不可能的。□



定理 3

中间过程无黑洞流

考虑对于给定的 TSG 下载流下的一个黑洞流。假设原始的TSG可能包含黑洞流，目标TSG不会有黑洞流，下载流集合没有发生改变的情况下，如果在原始TSG中不是黑洞流，所有TSG中间过程对应的流都不是黑洞流（中间过程不会增加黑洞流）。



定理 3

中间过程无黑洞流

考虑对于给定的 TSG 下载流下的一个黑洞流。假设原始的TSG可能包含黑洞流，目标TSG不会有黑洞流，下载流集合没有发生改变的情况下，如果在原始TSG中不是黑洞流，所有TSG中间过程对应的流都不是黑洞流（中间过程不会增加黑洞流）。

证明

假设 $S_i \rightarrow D$ 是中间过程中的一个黑洞流，但在原始TSG中不是黑洞流。那么至少一个节点被解析，黑洞流只会出现在这个解析时或解析后。

然而由于相同的原因，解析后的节点不会向未解析的节点输送流量，那么黑洞流只会出现在解析后的节点中，然而目标 TSG 不含有黑洞流，所以产生了矛盾。□





谢谢