B4: Google's Software-Defined WAN

Paper Reading

Log Creative

2021 年 10 月 19 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

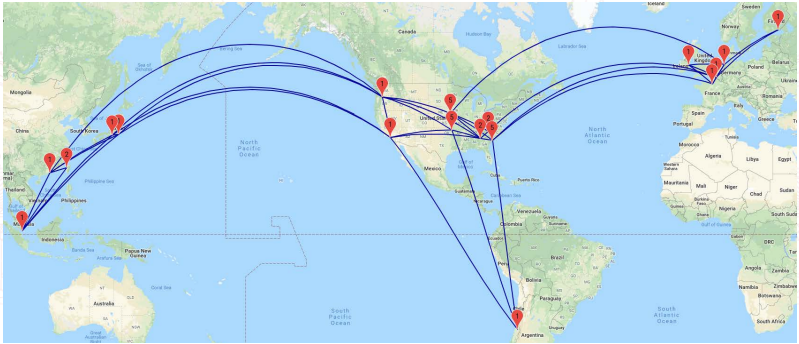
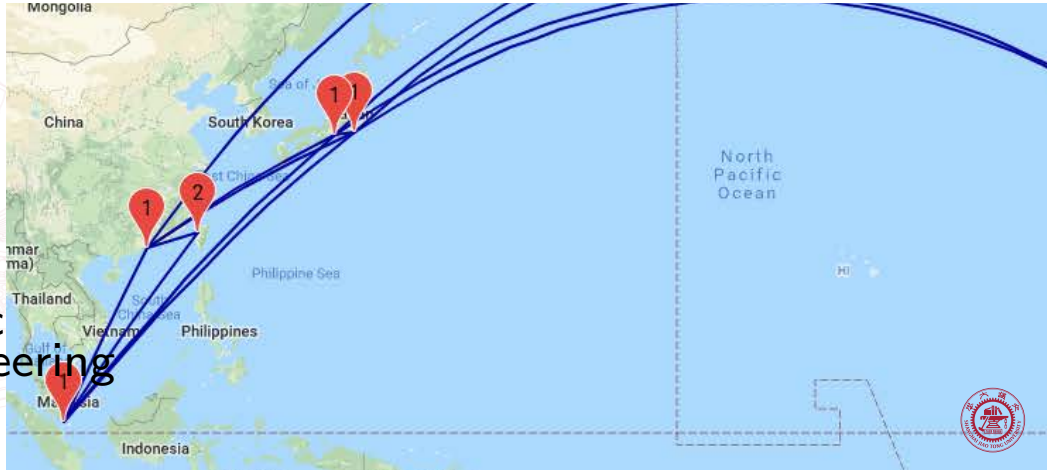


图: B4 全球网络

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：

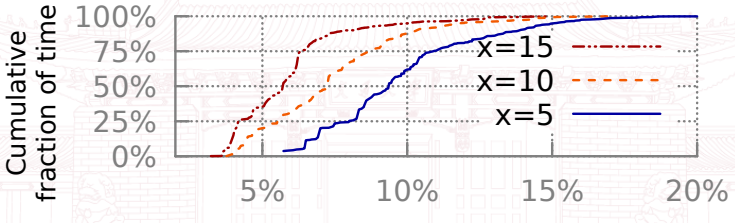
1. 增加了中央流量控制优化算法的运行时间。
2. 对交换机有限的流表空间增加压力。
3. 使得容量管理变得复杂并给应用开发者造成麻烦。

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。



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

图: 地理级流量不对等

不对等的后果

大幅减少系统效率

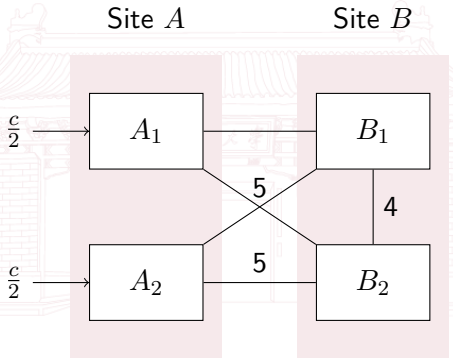
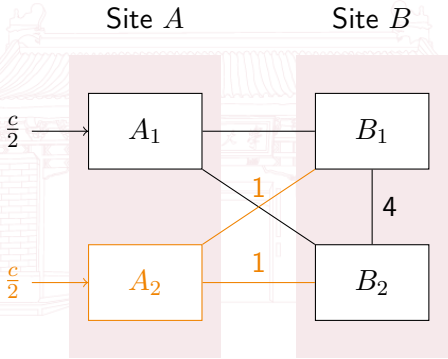


图: 对等

不对等的后果

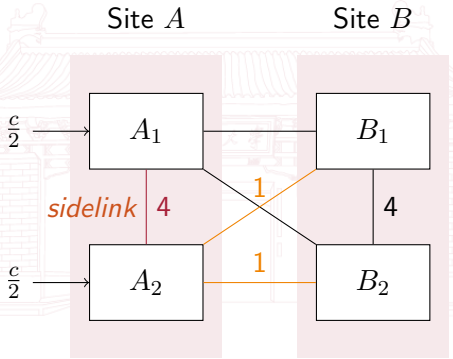
大幅减少系统效率



图： 不对等示例 $c = 4$

不对等的后果

大幅减少系统效率

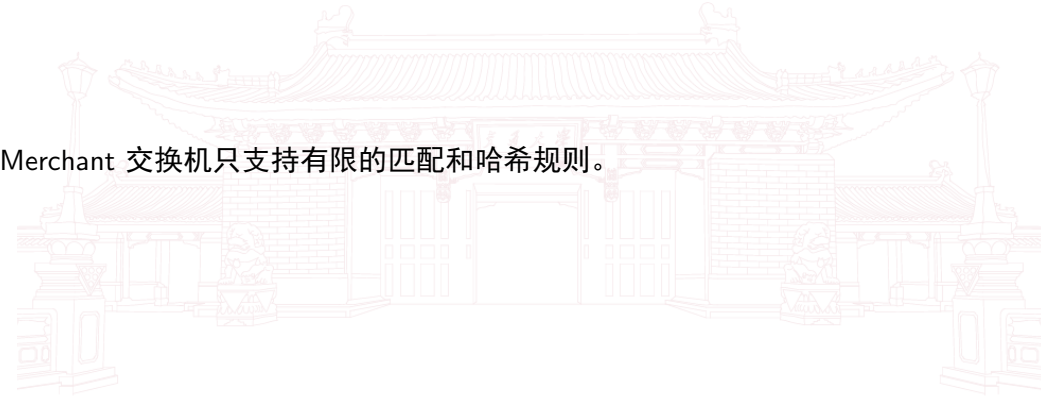


图： 不对等示例 $c = 12$

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，
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，

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

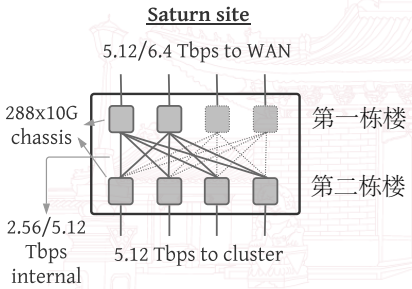


图: Saturn 站点

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

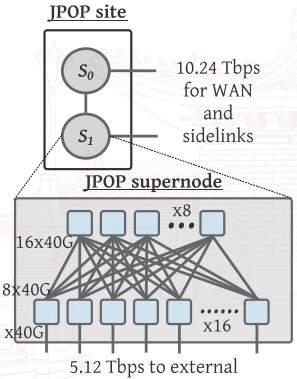


图: JPOP 站点

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

Jumpgate: Stargate

数据中心级

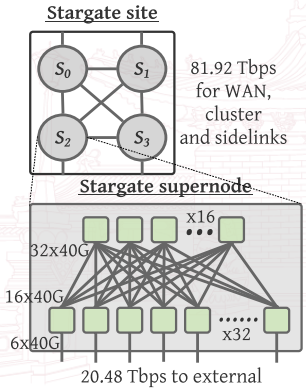


图: Stargate 站点

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

Jumpgate: Stargate

数据吞吐量大带来的好处

图: 交换机与交换机架

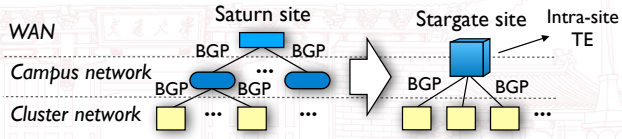


图: 减少 BGP 复杂度

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⟩

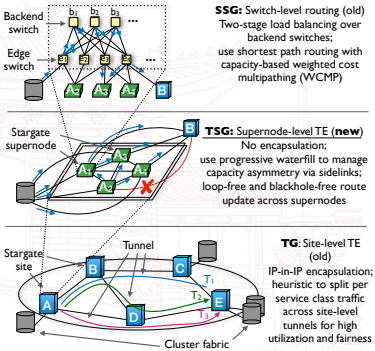


图: 不同的流量工程结构

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

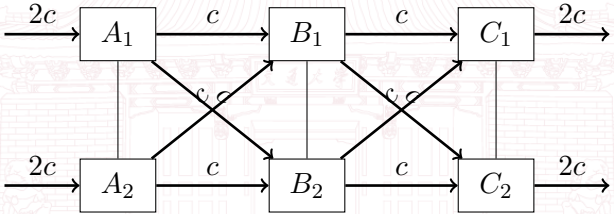


图: 对等网络 $TC = 4c$

TSG 生成算法

例子

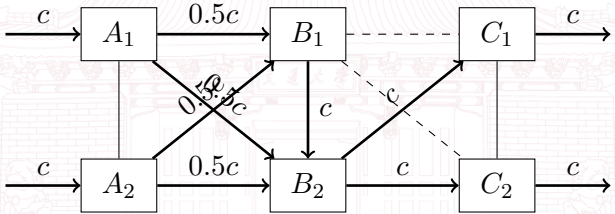


图: 不对等网络 $TC = 2c$

TSG 生成算法

例子

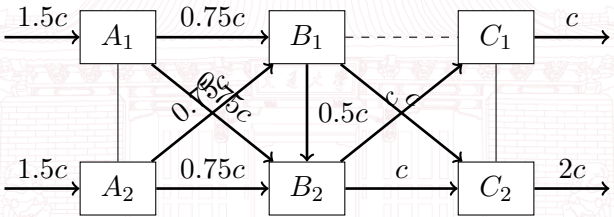
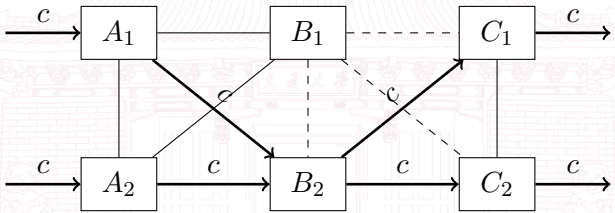


图: 不对等网络 $TC = 3c$

TSG 生成算法

例子（罕见）



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

TSG 生成算法

例子（罕见）

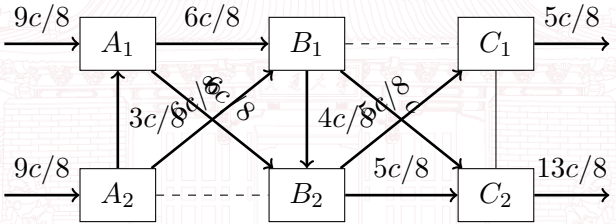


图: $TC = 9c/4$ (次优解)

TSG 生成算法

例子（罕见）

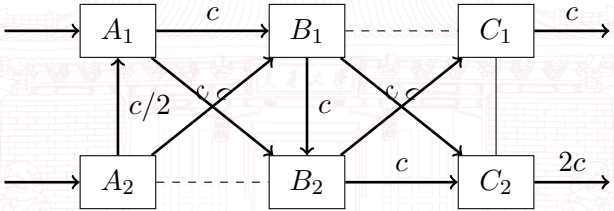


图: $TC = 3c$ (最优解)

网络流算法

回顾

Algorithm 2: Augment(f, c, P)

```
δ ⇐ bottleneck capacity of augmenting path P;
foreach e ∈ P do
    if e ∈ E then
        f(e) ← f(e) + δ;
    else
        f(e^R) ← f(e^R) − δ;
return f;
```

Algorithm 3: Ford-Fulkerson Algorithm

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

```
foreach e ∈ E do
    f(e) ← 0;
G_f ← residual graph;
while there exists augmenting path P do
    f ← Augment(f, c, P);
    update G_f;
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: $TSG(u, v)$

$\forall (u, v) \in E \ C_{\text{Remaining}}(u, v) := C(u, v);$

repeat

  $\forall (u, v) \in E \ weight(u, v) := 0;$

  $frozen\_flow$:=NULL;

  foreach $S_i \in S$ **do**

    $PG_{\text{2-hop}}(S_i) := \{[(S_i, v), (v, D)] | v \in S, C_{\text{Remaining}}(S_i, v) > 0, C_{\text{Remaining}}(v, D) > 0\};$

    if $C_{Remaining}(S_i) > 0$ **then**

      $weight(S_i, D) += 1;$

    else if $PG_{\text{2-hop}} \neq \varnothing$ **then**

      foreach *path* $P \in PG_{\text{2-hop}}(S_i)$ **do**

        foreach $(u, v) \in P$ **do**

          $weight(u, v) += \frac{1}{|PG_{\text{2-hop}}(S_i)|}$

    else $frozen\_flow := S_i$ **break**; `// Stop when any flow failed finding`

  `/* More procedure in the next page...`

Algorithm 5: TSG 生成算法（续）

repeat
  /* ... Continued from the previous page */
  if *frozen_flow* $\neq$ *NULL* **then break**;
  $E' = \{(u,v) \in E | weight(u,v) > 0\}$;
  $\forall (u,v) \in E' fair\_share(u,v) := \frac{C_{\text{Remaining}}(u,v)}{weight(u,v)}$;
  $BFS := \min_{(u,v)\in E'} fair\_share(u,v)$; /* Bottleneck Fair Share */
  foreach $(u,v) \in E'$ **do**
    $C_{\text{Remaining}}(u,v) -= BFS \times weight(u,v)$;
until *True*;
foreach $(u,v) \in E'$ **do**
  $TSG(u,v) = \frac{C(u,v) - C_{\text{Remaining}}(u,v)}{\sum_{(u,v')\in E}(C(u,v') - C_{\text{Remaining}}(u,v'))}$;

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。

谢谢