B4: Google's Software-Defined WAN

Paper Reading

李子龙　张　露　李婉婷　田F元

2021 年 11 月 8 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

第 1 节

简介

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

第 2 节

背景与动机

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，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。

第 3 节

站点拓扑的进化

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

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]

[1] Celso 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/.

第 4 节

分层流量工程

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

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;
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;
if 对等 **then** 不使用旁路连接;
else 通过旁路连接重新分配 TSG;
使用上述 TSG 结果计算站点级每条链路的有效容量，生成 TG;
生成 TE 操作的无环依赖图;
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)

[2] 不关闭的后果

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： 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)

```
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 $C_{Remaining}(S_i, D) > 0$ **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。

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;

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

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 $S_i \to D$ 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。 □

第 5 节

高效交换规则管理

FG 匹配

最初 ACL 设计

在最初的设计中，FG 匹配是通过 ACL[2]实现的

$$size_{ACL} \geq num_{Sites} \times num_{Prefixes/Site} \times num_{ServiceClasses}$$

ACL大小限制=3K

集群前缀平均长度≥16

支持的服务类型=6

计算后可知，当站点个数达到 32 左右，将会超过 ACL 的大小限制。

[2] Access Control List，访问控制列表，完全匹配。

FG 匹配

对比

采用层级 FG 匹配可以大幅增大支持的站点数量。

图: 最大支持站点数量： 32

图: 最大支持站点数量： 1920

FG 匹配

层级 FG 匹配

由于 LPM[3] 表项远多于 ACL 表，但是 LPM 表并不能直接与 DSCP 标签匹配却可以与虚拟路由转发（VRF）标签匹配，所以首先采用虚拟转发平面（VFP）生成 VRF。

Algorithm 8: 层级 FG 匹配

1 将 DSCP 标签先通过 VFP 匹配到一个 VRF 以确定相应的服务等级;

2 结合 VRF 对集群前缀匹配于 LPM 表;

(b) After

[3] Longest Prefix Matching，最长前缀匹配。

FG 匹配

其他特性

当 TE 发生某些致命错误时，可以在源站点禁用 TE，暂时退回到 BGP/ISIS。在入口站点的交换机上删除 TE 转发规则，数据包可以匹配较低优先级的 BGP/ISIS 转发规则，而无需封装。

分流

最初设计

最初，仅在入口边缘交换机实现划分

$$size_{ECMP} \geq num_{Sites} \times num_{PathingClasses} \times num_{TGs} \times num_{TSGs} \times num_{SSGs}$$

1. $num_{PathingClasses}$ 是共享共同路径约束的服务类的数量，其值为 3
2. num_{TGs} 是隧道分割的颗粒度，其值为 4
3. num_{TSGs} 是每个超级节点的分割粒度，其值为 32
4. num_{SSGs} 是 Stargate 后端阶段可供分配的交换机个数，其值为 16

分流

最初设计

最初，仅在入口边缘交换机实现划分

$$198000 \geq 33 \times num_{PathingClasses} \times num_{TGs} \times num_{TSGs} \times num_{SSGs}$$

当 num_{sites} 取 33 时，$size_{ECMP}$ 至少为 198K，但实际上，交换机只支持 $size_{ECMP}$ 不超过 14K，可以通过降低数据分流的颗粒度来降低所需的 ECMP 条目，但是这会填平 TE 带来的优势。

分流

基于划分的高效流量哈希

Algorithm 9: 基于划分的高效流量哈希

1 在一个 CLOS 集群上，边缘交换机决定采用哪个隧道（TG 分流）并判断入端数据流应被发送到哪个站点（TSG 分流第 I 步）。用一特殊的 MAC 标识本地/下一跳结果;

2 后端交换机判断该数据包应被发送到哪个超级节点上（TSG 分流第 II 步），并判断哪些出边缘交换机与该超级节点有链接（SSG 分流）;

并对可用的后端交换机添加 LAG（Link Aggregation Group）以进一步减少入口边缘交换机分配规则。

表: 分流判断负载对比

	Before	After
边缘	TG, TSG, SSG	TG, TSG(I)
后端		TSG(II), SSG

第 6 节

评估

规模

总流量

图：归一化到最小值的流量总量

- 五年内，总流量增长了两个数量级；平均每 9 个月，流量就增加一倍
- 相较于面向互联网的广域网，B4 提供了更多的流量，并且增长速度更快

规模

站点与控制域的数量

图: 站点和控制域的数量

- 基于 Saturn 时，二者数量一致；从 2013 年部署 JPOP fabrics 以后出现分歧
- 2015 年，为了提高可拓展性，大大减少了 Saturn fabrics 的站点数量，将其替换为 Stargate。但由于在迁移期间，Saturn 和 Stargate fabrics 会暂时在一个站点内共存，这反而带来了可拓展性问题。

规模

Flow Group

图: 每个站点的 FG 数

- 过去的五年中，平均每个源站点的 Flow Group 增加了六倍
- 2015 年，停止向交换机和 supertrunk 端口分发 IP 地址，这使 Flow Group 减少了约 30

规模

Tunnel Group

图: 每个站点的 TG 数

- 2013 年初，开始支持多个服务级别，这使得每个站点的 TG 数翻了一番
- 此后，随着站点数和服务级别的增加，每个站点的 TG 数持续增长

规模

隧道数

图: 每个站点的传输隧道数

- 每个站点的传输隧道数由中央控制器的配置调控，这有助于避免配置过多的交换规则，但也限制了备份隧道的数量
- 2016 年，随着交换规则管理的改进，提高了可安装备份隧道的数量

可用性

基于带宽的有效性

$$\frac{allocation}{min\{demand, approval\}}$$

其中，demand 是基于短期内的历史试用情况评估的，approval 是每个 SLO 所允许的最小带宽，allocation 指当前带宽，它受到网络容量和公平性的约束。

基于连通性的有效性 对于每一个服务级别，通过一个网络测量系统对每个集群对进行测量。每一个集群对的有效性定义如下：

$$\begin{cases} 1, & loss_rate \leq \alpha \\ 10^{-\beta \times (loss_rate - \alpha)}, & otherwise \end{cases}$$

其中，$\alpha = 5\%$ 是一个灵敏度阈值，它过滤掉了大部分的瞬态损失，同时捕获了会影响有效性的更大的损失。超过阈值后，随着传输损失率的提高，有效性成指数级下降，衰减因子 $\beta = 2\%$。

可用性测量结果

图: 每个服务级别的测量到的和目标需要的不可用性

- 取上述两个指标的最小值作为网络有效性
- 对于每一个服务类，B4 都达到了目标有效性

设计折衷

拓扑抽象化

为了更少的容量损失进行拓扑抽象。

图: 由分层拓扑中的容量不对等造成的容量损失

设计折衷

带有两级散列的分层结构 TE

图：吞吐量和平均运行时间比较

- 分层结构 TE 的运行时间仅为 2-3 秒，比扁平结构 TE 快 188 倍。
- 当最大流量分割间隔设置为 1 时，不论是分层结构还是扁平结构性能都很差，其吞吐量分别少于对应结构最大吞吐量的 91% 和 46%。
- 在最初的 8 路分流中，分层结构 TE 只实现了其最大吞吐量的 94%。

设计折衷

TSG 排序

- 在超过 99.7% 的 TSG 操作中，TSG 排序只需要一或两个“步骤”。
- 每个步骤由一个或多个 TSG 更新组成，其中它们的相对顺序不被强制执行。

设计折衷

TSG 排序

(b)

- 在 99.7 百分位时，延迟增加了 2 倍，最坏情况下增加 2.63 倍。
- TSG 排序算法的运行时相对于编程延迟可以忽略不计。

设计折衷

TSG 排序

- 没有排序：可用容量因出现黑洞/循环降至零的可能性约为在 3%。
- 有排序：这个可能性将被降低一个数量级。

设计折衷

TSG 排序

- 在没有排序的情况下，TSG 运行时，有超过 2% 的中间状态中形成转发循环，这导致超过超过 38% 的入口流量将被丢弃。

第 7 节

运营经验和待解决问题

简化管理

1. 升级前，需要手动计算由于容量不对称而造成的容量损失，以保证降级的站点容量仍然满足流量需求；升级后，需要人工干预之处大大减少，因为 TE 系统能够自动且高效的处理不对称容量。
2. 限制操作范围至一次一个域，以减小潜在的影响；使用影响分析工具对网络情况进行监控，通过结果判断某些请求是否被接受。
3. 由于 B4 网络的大规模，由命令行控制网络“排水”操作是不现实的，这一操作由管理工具调用，这些工具通过管理 RPC 来协调网络操作。

确定旁路容量

确定应部署的最佳旁路容量以满足带宽保证是一个非常困难的问题，Google 正在研究一个日志驱动的统计分析框架以解决该问题。

处理不平衡输入流量

TSG 算法默认入端流量是平衡的，优势在于，管道可以共享 TSG 规则，而不用对每个管道都去分配一套规则（这样做会超过硬件规则极限），但缺点在于，对不平衡的流量处理能力较差。Google 尝试开发计算每对相邻站点级链接的 TSG 规则的算法进行优化。

相关工作

- Software-defined WAN 一些公司也已经将 SDN 用于数据中心间的连接，如 Facebook、微软 SWAN 等。Google 认为他们发展 B4 的经验也可以应用于这些工作。

相关工作

- Software-defined WAN 一些公司也已经将 SDN 用于数据中心间的连接，如 Facebook、微软 SWAN 等。Google 认为他们发展 B4 的经验也可以应用于这些工作。
- Decentralized, hierarchical network abstractions 分层路由的概念由来已久，近年来也有不少去中心化的、分层式的 SDN 路由架构被提出。B4 是以这些概念为基础构建出来的，Google 认为其创新点在于克服了将这些众所周知的概念应用于全球规模时所面临的实际挑战。

相关工作

- Software-defined WAN 一些公司也已经将 SDN 用于数据中心间的连接，如 Facebook、微软 SWAN 等。Google 认为他们发展 B4 的经验也可以应用于这些工作。
- Decentralized, hierarchical network abstractions 分层路由的概念由来已久，近年来也有不少去中心化的、分层式的 SDN 路由架构被提出。B4 是以这些概念为基础构建出来的，Google 认为其创新点在于克服了将这些众所周知的概念应用于全球规模时所面临的实际挑战。
- TE and WAN resource scheduling 诸如 Tempus, Pretium, SOL, FFC 等创新性的概念是可以被运用到 B4 中以优化其控制和数据层面的。

相关工作

- Software-defined WAN 一些公司也已经将 SDN 用于数据中心间的连接，如 Facebook、微软 SWAN 等。Google 认为他们发展 B4 的经验也可以应用于这些工作。
- Decentralized, hierarchical network abstractions 分层路由的概念由来已久，近年来也有不少去中心化的、分层式的 SDN 路由架构被提出。B4 是以这些概念为基础构建出来的，Google 认为其创新点在于克服了将这些众所周知的概念应用于全球规模时所面临的实际挑战。
- TE and WAN resource scheduling 诸如 Tempus, Pretium, SOL, FFC 等创新性的概念是可以被运用到 B4 中以优化其控制和数据层面的。
- Network update and switch rule management B4 中的 TSG 排序不需要数据包标注即可解决限制性网络使用案例，这与同领域其他工作是不同的；解耦转发规则这一概念在很多过往的工作中已被应用，B4 中，这一方案的独特之处在于，揭示了当规模足够大时，扩大交换机规则管理方面面临的挑战。

第 8 节

总结

总结

该论文介绍了本文介绍了 Google 团队将 B4 从一个以99%的可用性为目标的批量传输、内容复制网络发展到支持需要99.99%可用性的交互式网络服务的网络的历程和经验。主要是通过将 B4 重新架构为分层拓扑结构，同时开发 TE 算法和交换机规则管理机制来进行升级。高可用性、带宽充足的网络对众多云服务有着重要贡献，Google 团队也正努力继续优化 B4 网络，使其可用性再次得到提高。

分工

- 李子龙：演讲、第 6 节
- 张　露：第 1、2 节
- 李婉婷：第 3、4 节
- 田$\boxed{\text{F}}$元：第 5、7 节

谢 谢