中国科学技术大学
University of Science and Technology of China

Design, implementation and evaluation of congestion control for multipath TCP

Damon Wischik, Costin Raiciu, Adam Greenhalgh, Mark Handley

NSDI 2011

授课教师：赵功名

中国科大计算机学院

2025年秋·高级计算机网络

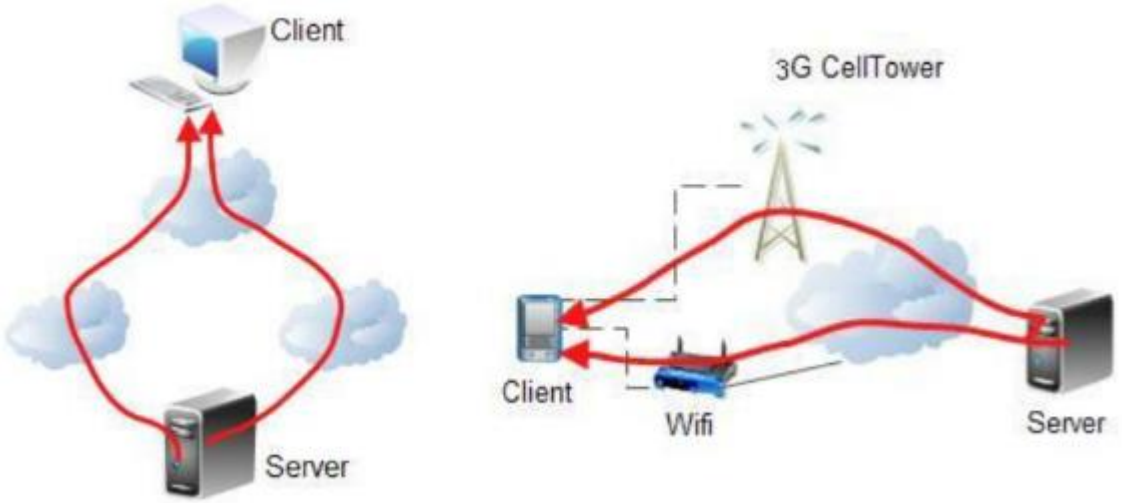
Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Introduction

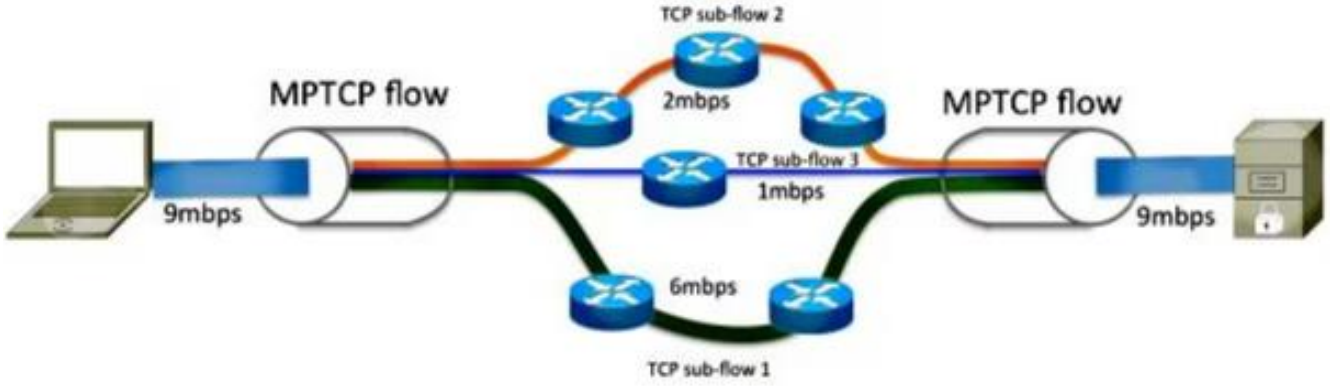
- 互联网终端的变化：从单宿主到多宿主
  - 过去： 计算机通常只有一个网络接口（例如以太网卡）
  - 现在： 移动设备（手机、平板、笔记本）普及，通常拥有多个接口（WiFi, 4G/5G, Bluetooth）
  - 数据中心： 服务器拥有多块网卡，网络拓扑（如 FatTree）提供多条冗余路径

问题： 标准 TCP 只能利用其中一条路径，浪费了大量的网络资源



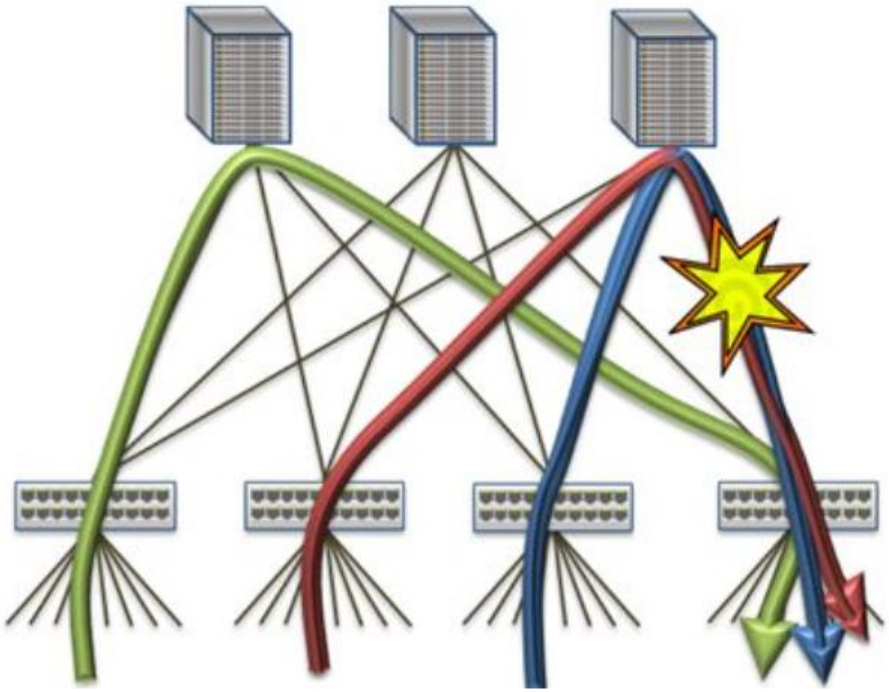
Introduction

- 什么是多路径TCP (MPTCP)?
  - 标准 TCP 仅允许数据流在两个端点之间通过单一路径传输
  - 多路径 TCP (MPTCP) 设计目标是：
    - 能够为单个连接使用多条网络路径
    - 能够像常规 TCP 一样利用可用网络路径，且不挤占（Starving） 常规 TCP 的资源
    - 对于现有应用来说，像常规 TCP 一样易用（无需更改终端主机上的应用代码）



Introduction

- 为什么要使用多路径?
  - 标准 TCP是面向连接的4层协议，它可以提供可靠传输，并且拥有拥塞控制机制。但是TCP的单路径传输的协议，即便网络上存在多条路径资源，也无法利用这些路径资源。因为TCP是通过四元组：{源IP，目的IP，源端口，目的端口} 唯一的确定一个连接
  - 数据中心存在大量的路径资源，使用 MPTCP 可以更加充分地利用带宽资源，且提供可靠传输，避免产生拥塞，动态的将数据更多的发送到负载低的链路上。有效的提高负载均衡性能



Introduction

➢ 使用多路径会带来哪些好处?

可靠性 (Reliability)

即使一条路径发生故障，连接依然保持。在不同接口间实现无缝故障转移

吞吐量 (Throughput)

聚合多条链路的带宽（例如 WiFi + 3G），从而实现更快的下载速度

移动性 (Mobility)

在不同网络间切换时（Handover），无需中断 TCP 连接

Introduction

- 核心挑战：拥塞控制
  - 多路径传输不仅仅是将数据分发到不同路径上，更关键的是如何分配流量
  - 如果在多条路径上简单地运行标准 TCP，会导致什么后果?
    - 对单路径用户是否公平?
    - 能否有效利用网络资源?
    - 流量是否会涌向拥塞的路径?
  - **需要在多种场景下验证MPTCP的性能效果，以及解决协议实现上遇到的实际挑战**

Outline

I. Introduction
II. **Design problem for MPTCP congestion control**
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Design problem for MPTCP congestion control

- 怎么设计好的拥塞控制算法呢?
  - 如果在每个子流上运行标准的TCP拥塞控制算法，只是简单地“叠加”TCP将导致极度不公平和效率低下，无法达到我们预期效果
  - 我们需要设计一个能同时解决以下矛盾的算法：
    - 共享瓶颈的公平性 (Fairness at Shared Bottlenecks)：MPTCP聚合后的总吞吐量，不能超过单路径TCP在最佳路径上的吞吐量
    - 路径选择的效率 (Efficient Path Selection)：流量应该自动从高丢包（拥塞）路径转移到低丢包（空闲）路径
    - 异构网络的鲁棒性 (Robustness to RTT Mismatch)：面对WiFi（低延迟高丢包）和3G（高延迟低丢包），算法不能被误导

Design problem for MPTCP congestion control

- 共享瓶颈处的公平性（Fairness at shared bottlenecks）
  - 无论 MPTCP 开启了多少条子流，如果它们都经过同一个瓶颈，那么这个 MPTCP 连接作为一个整体，

    获得的带宽**不应该超过一个标准的单路径 TCP 连接**

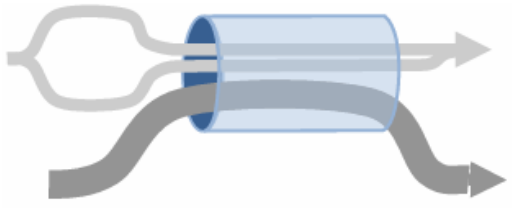


Figure 1: A scenario which shows the importance of weighting the aggressiveness of subflows.

- 瓶颈链路：中间的那根管子代表网络中的瓶颈链路（带宽有限）
- 普通用户（Single-path TCP）：只有一条路径穿过这个瓶颈
- MPTCP用户：建立了两条子流（Subflows），这两条子流恰好都经过了同一个瓶颈链路

  - 竞争情况：瓶颈处共有 3条流 在竞争带宽（1条来自普通TCP用户，2条来自MPTCP用户），MPTCP 用户仅仅因为开启了多路径功能，就“欺负”了单路径用户，抢占了双倍的资源，这是不公平的

Design problem for MPTCP congestion control

- 共享瓶颈处的公平性（Fairness at shared bottlenecks）
  - **理想结果：**在右图中MPTCP 用户（整体）和普通 TCP 用户应该各获得 50% 的带宽

ALGORITHM: EWTCP

- For each ACK on path r, increase window w_r by a/w_r.
- For each loss on path r, decrease window w_r by $w_r/2$.

Here w_r is the window size on path r, and $a = 1/\sqrt{n}$ where n is the number of paths.

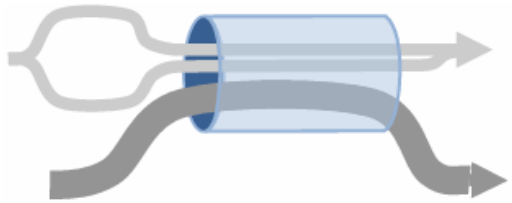


Figure 1: A scenario which shows the importance of weighting the aggressiveness of subflows.

- 论文提到了以上 EWTCP (Equal-Weighted TCP) 算法作为目前的基础思路【文中后来指出了它的缺陷】
  - 降低侵略性：MPTCP 的子流不能像普通 TCP 那样激进地增加窗口
  - 权重分配：如果一个 MPTCP 连接有 n 条子流，那么每条子流的增长速度应该降低
    - 具体来说，EWTCP 建议将窗口增加的系数设为 $\alpha = 1/\sqrt{n}$
    - 这样，多条子流的总行为加起来，就大约等同于一条标准 TCP 流的行为

Design problem for MPTCP congestion control

- 选择高效路径（Choosing efficient paths）

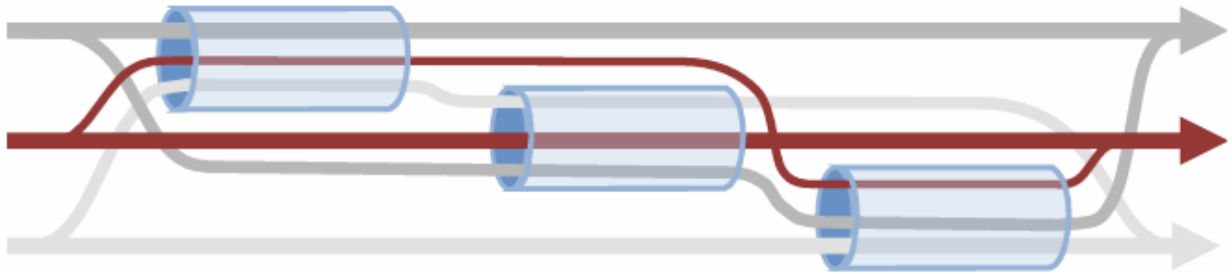


Figure 2: A scenario to illustrate the importance of choosing the less-congested path

- 考虑这样一个场景：有一个 MPTCP 连接，它可以走两条路到达目的地，假设所有链路的带宽容量都是 12Mb/s
  - 路径 1（下方）：直达，只经过 1 跳（1-hop）
  - 路径 2（上方）：绕路，经过 2 跳（2-hops）
- 如果不区分路径好坏（如 EWTCP）：算法只是简单地将流量平均分配（或者只是为了公平而降低整体速率），它可能会在两条路径上各发送一半流量；虽然用户可能获得了满意的带宽，但路径 2 占用了两条链路的资源，这是一种浪费

Design problem for MPTCP congestion control

- 选择高效路径（Choosing efficient paths）

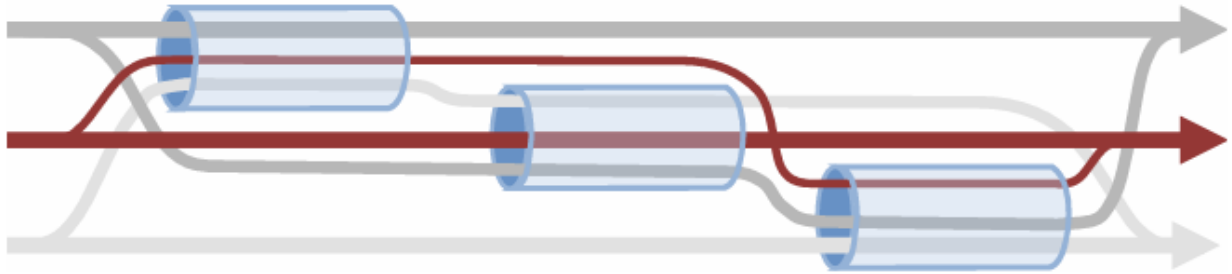


Figure 2: A scenario to illustrate the importance of choosing the less-congested path

- 根据 TCP 的特性，长路径（经过更多跳数）通常会有更高的累积丢包概率（拥塞概率）
- 理想的 MPTCP 行为：应该检测到下方路径（1跳）的丢包率更低（更不拥塞），从而将所有流量转移到下方路径
  - 用户依然获得了 12Mb/s 的带宽。
  - 上方那两条链路现在的带宽被释放出来了，可以留给网络中的其他用户使用
- 结论：**通过总是偏向更少拥塞的路径，MPTCP 实际上在做网络流量工程，把资源留给最需要的地方**

Design problem for MPTCP congestion control

➢ 选择高效路径（Choosing efficient paths）

- EWTCP 将多条路径视为独立的，只是简单地降低了每条路径的窗口增长速，它不会主动根据拥塞程度大幅调整流量比例，结果导致那条 5Mb/s 的链路被塞满了，而 12Mb/s 的链路可能还很空闲，拥塞不均衡，有的链路很挤，有的很空
- 在右图中通过COUPLED（耦合控制，效果好），算法会将所有子流的丢包率进行比较，只要某条路径的丢包率（拥塞程度）比其他路径高，就减少该路径的窗口，增加低丢包率路径的窗口，所有的链路拥塞程度最终趋于一致

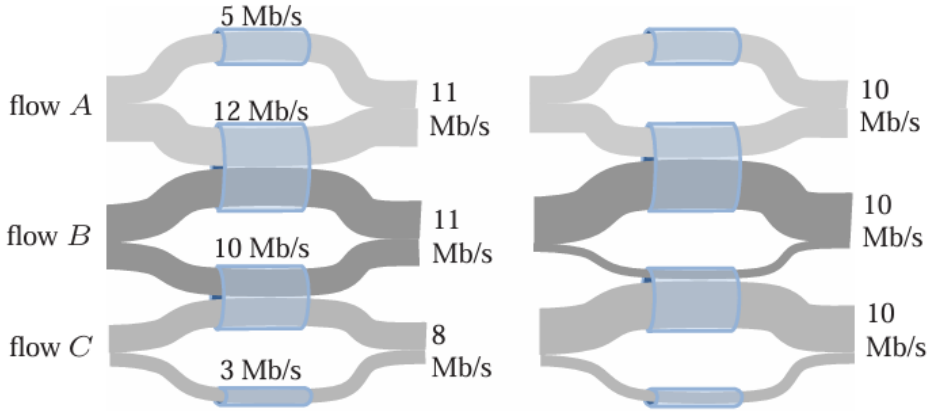


Figure 3: A scenario where EWTCP (left) does not equalize congestion or total throughput, whereas COUPLED (right) does.

Design problem for MPTCP congestion control

➢ 选择高效路径（Choosing efficient paths）

ALGORITHM: COUPLED

- For each ACK on path r, increase window w_r by $1/w_{\text{total}}$.
- For each loss on path r, decrease window w_r by $w_{\text{total}}/2$.

Here w_{total} is the total window size across all subflows. We bound w_r to keep it non-negative; in our experiments we bound it to be $\geq$ 1pkt, but for the purpose of analysis it is easier to think of it as ≥ 0.

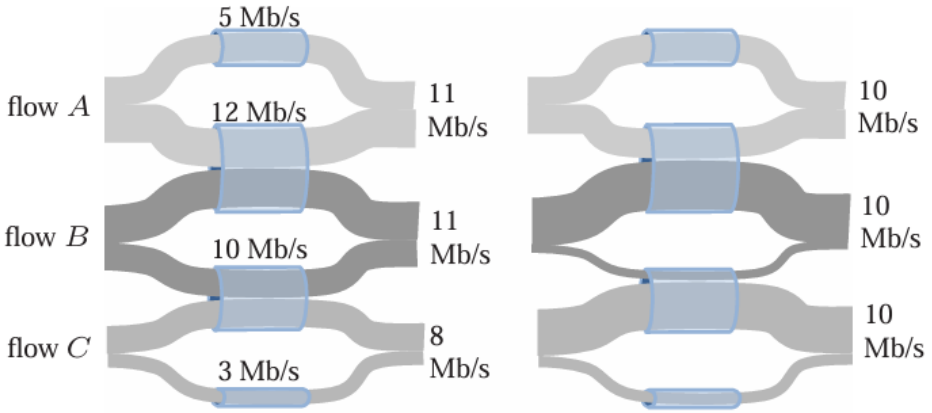


Figure 3: A scenario where EWTCP (left) does not equalize congestion or total throughput, whereas COUPLED (right) does.

- 论文提到了以上 **COUPLED（完全耦合）**算法作为目前的思路【文中后来也指出了它的缺陷】
  - 算法的核心思想是将所有子流（Subflows）看作一个整体（即一个虚拟的“大”TCP连接）来进行控制，w_total是所有子流拥塞窗口的总和
  - 收到路径 r 的 ACK 时，窗口 w_r 增加 $\frac{1}{w_{total}}$
  - 路径 r发生丢包时，窗口 w_r 减少$\frac{w_{total}}{2}$
  - **最终结果：**流量会完全转移到丢包率最低的那条路径上，其他路径的窗口降为 0（或 1）

Design problem for MPTCP congestion control

- 往返时间不匹配的问题（Problems with RTT mismatch）
  - 在此之前，论文假设不同路径的 RTT（往返延迟）大致相同。

    但在现实中，移动设备经常同时连接 WiFi 和 3G/4G，这两者的特性截然相反
  - WiFi：低延迟（RTT 短，约 10ms），高带宽
  - 3G：延迟很高（RTT 长，约 100ms），带宽通常较低。丢包率极低（例如 1%）。这是因为运营商为了保证连接不断，在基站设置了巨大的缓冲区（Buffer），数据包只是在排队，并没有被丢弃

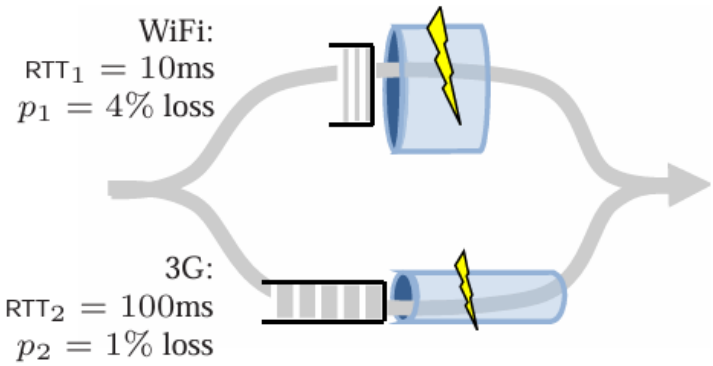


Figure 4: A scenario in which RTT and congestion mismatch can lead to low throughput.

Design problem for MPTCP congestion control

- 往返时间不匹配的问题（Problems with RTT mismatch）
  - 当往返时延（RTT）不相等时，EWTCP 和 COUPLED 算法都会出现问题
  - COUPLED 算法的原则是：“把流量都移到丢包率最低（最不拥塞）的路径上”，将导致流量全部转移到 3G，放弃 WiFi
  - EWTCP 不会像 COUPLED 那样极端地把流量全切到 3G，但它在分配流量时也没有考虑到 RTT 的差异，EWTCP 会在两边各跑一半流量

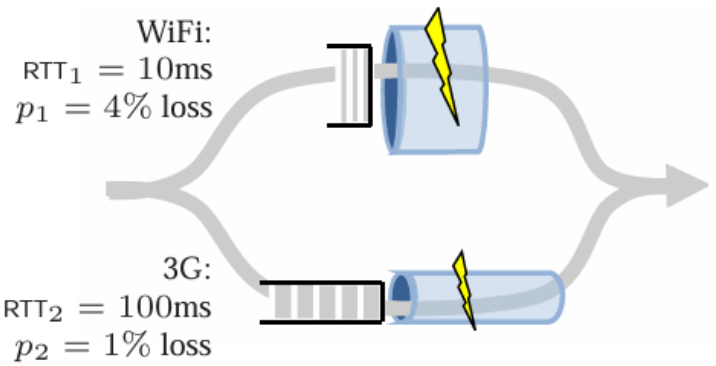


Figure 4: A scenario in which RTT and congestion mismatch can lead to low throughput.

- 结论：仅仅关注“丢包率（拥塞程度）”是不够的，如果不把 RTT纳入拥塞控制的考量范围，MPTCP 就会被那些“不丢包但延迟极高”的路径干扰，导致用户体验下降

Design problem for MPTCP congestion control

- 适应负载变化（Adapting to load changes）

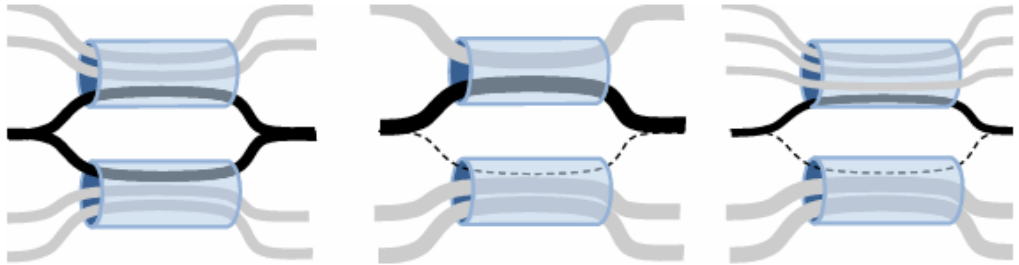


Figure 5: A scenario where multipath TCP might get 'trapped' into using a less desirable path.

- 阶段一：MPTCP 连接有两条路径可用（上路和下路）。假设一开始上路比较空闲（拥塞少），下路比较拥塞
- 阶段二：流量转移COUPLED 算法检测到上路好，于是把所有流量都切到了上路，MPTCP 完全停止了对下路的使用
- 阶段三：负载突变上路瞬间变得非常拥塞。同时，下路变空闲了，MPTCP 应该立即把流量切回下路，但是MPTCP 无动于衷。因为在阶段二，发送端不再往下路发包，收不到下路的 ACK ，算法永远不知道下路现在其实已经变好了

Design problem for MPTCP congestion control

- 适应负载变化（Adapting to load changes）

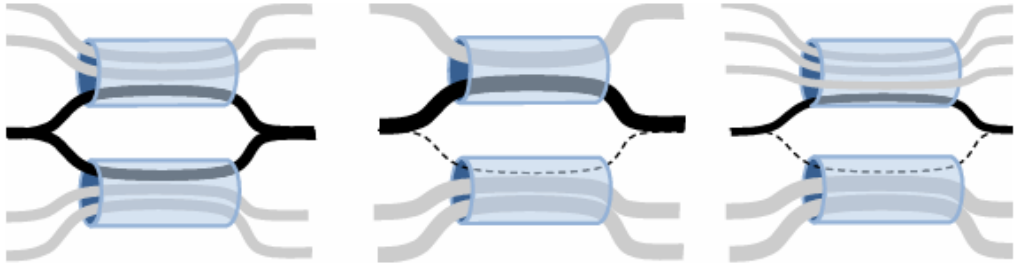


Figure 5: A scenario where multipath TCP might get 'trapped' into using a less desirable path.

ALGORITHM: SEMICOUPLED

- For each ACK on path r, increase window w_r by a/w_{total}.
- For each loss on path r, decrease window w_r by $w_r/2$.

Here a is a constant which controls the aggressiveness, discussed below.

$$w_r \approx \sqrt{2a}\frac{1/p_r}{\sqrt{\sum_s 1/p_s}}.$$

- 因此“只使用最好的路径”这个原则必须被修正，即使某条路径非常拥塞，也不能“抛弃”，需要偶尔发包试探
- 为了解决这个问题，作者提出了 Semi-Coupled（即后来的 LIA 算法），即使路径很差，也保持微小的窗口确保持续收到 ACK 反馈，丢包规则改为：只减少发生丢包的那条路径的窗口，而不是像 COUPLED 那样惩罚总窗口
- 效果：**使得算法能够适应网络负载的动态变化**

Design problem for MPTCP congestion control

➢ 补偿往返时间不匹配（Compensating for RTT mismatch）

- 为了在数学上严谨地定义“什么是好的 MPTCP”，作者提出了两个必须同时满足的硬性要求
  - 原则一：MPTCP 连接获得的总吞吐量，必须至少等于该连接所有可用路径中，最好的一条路径单独使用TCP 时能获得的吞吐量
  - 原则二：MPTCP 在任何瓶颈链路上占用的资源，不应超过一个单路径 TCP 流在该链路上通过最佳路径组合所能获得的资源，保证 MPTCP 对互联网上的其他普通 TCP 用户是公平的

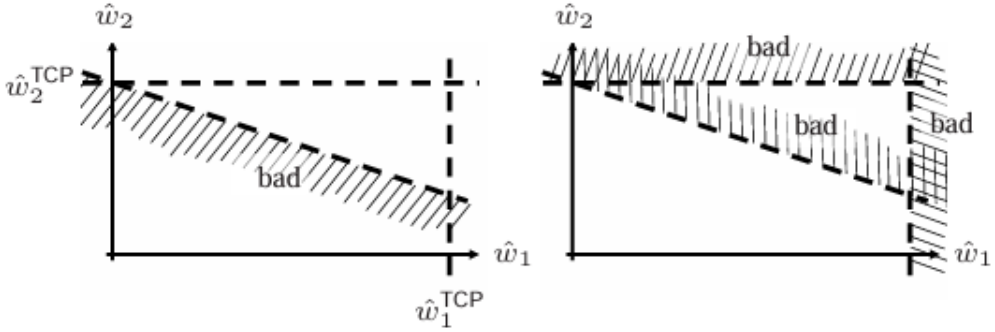


Figure 6: Fairness constraints for a two-path flow. Constraint (3) on the left, constraints (4) on the right.

- 对角线代表了理想状态：既满足了激励性（吞吐量够高），又满足了公平性（不抢占）。算法的目标就是通过调整参数，让操作点落在或者逼近这条对角线上

Design problem for MPTCP congestion control

- 补偿往返时间不匹配（Compensating for RTT mismatch）
  - 作者提出了最终的算法

$$\text{Increase } w_r \text{ by } \min\left(\frac{a}{w_{total}}, \frac{1}{w_r}\right)$$

  - Increase rule：同时保证公平性和吞吐量，在任何一条特定路径上，MPTCP 增加窗口的速度永远不会超过标准 TCP

$$\text{Decrease } w_r \text{ by } \frac{w_r}{2}$$

  - Decrease rule：只惩罚发生丢包的那条路径，保留了探测能力，适应网络的动态变化

ALGORITHM

- Each ACK on subflow r, increase the window w_r by $\min(a/w_{\text{total}}, 1/w_r)$.
- Each loss on subflow r, decrease the window w_r by $w_r/2$.

Here

$$a = \hat{w}_{\text{total}} \frac{\max_r \hat{w}_r/\text{RTT}_r^2}{(\sum_r \hat{w}_r/\text{RTT}_r)^2}, \quad (5)$$

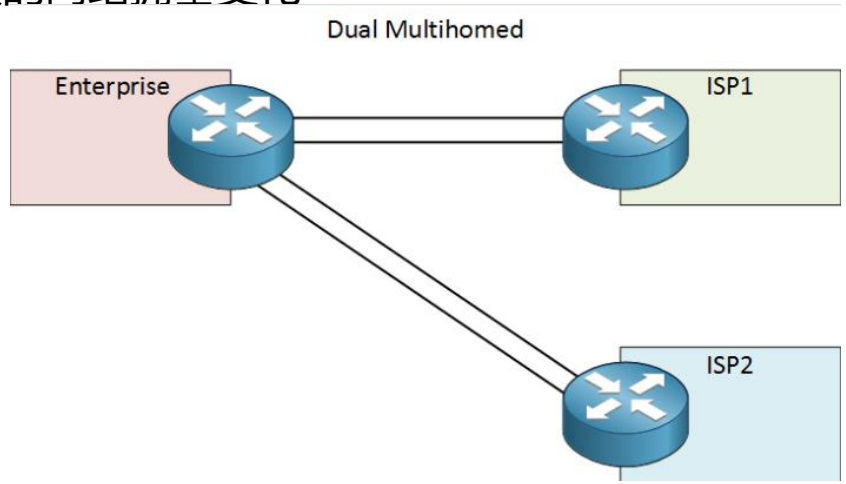
  - 该参数动态计算，它确保 MPTCP 获得的带宽总量大致等于在最佳路径上运行标准 TCP 的带宽

Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Congestion control at multihomed server

- Multihomed Server 指的是通过多个上游网络链路（例如连接到两个不同的 ISP，或者通过两条不同的物理线路）接入互联网的服务器
- 虽然多宿主服务器在物理上拥有多条通路，但在流量工程 (Traffic Engineering) 和 负载均衡 (Load Balancing) 方面存在长期未解决的痛点传统的流量调度主要依赖 BGP 技术，存在以下问题：
  - 粒度太粗 (Coarse-grained)： 只能针对 IP 前缀块进行调整，无法针对具体的数据流
  - 反应太慢 (Very slow)： BGP 的收敛和策略调整需要时间，无法应对秒级的网络拥塞变化
  - 系统压力 (Stress)： 频繁的路由变动会给全球路由系统带来压力
- 在这个场景下做 MPTCP 拥塞控制可以实现细粒度且快速的负载均衡



Congestion control at multihomed server

- 静态负载均衡实验模拟
  - 拓扑：这是一个环面（Torus）拓扑结构，包含 A、B、C、D、E 五条瓶颈链路
  - 流量：网络中有多个多路径流在跑，每条流使用两条路径
  - 变量：研究人员人为地降低链路 C 的带宽容量，使 C 成为最拥塞的瓶颈
  - 预期：理想的算法应该检测到 C 很堵，自动把流量转移到 B 和 D，进而影响 A 和 E，最终让所有链路的拥塞程度（丢包率）趋于一致

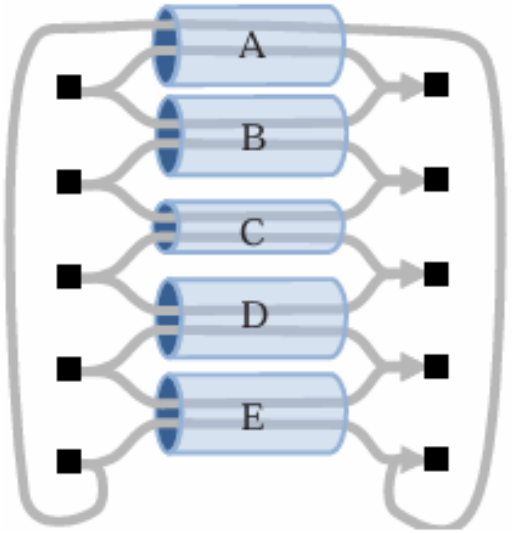


Figure 7: Torus topology. We adjust the capacity of link C, and test how well congestion is balanced.

Congestion control at multihomed server

- 对比效果：
  - **COUPLED**：表现最好。它的曲线紧贴着 0（接近比率 1），说明它能极其强力地将流量从 C 移走，实现完美的拥塞均衡
  - **EWTCP**：表现最差。当 C 容量变小时，比率飙升，说明它仍然死板地往 C 发数据，无法缓解拥塞。
  - **MPTCP**：表现居中，非常接近 COUPLED
  - 结论：这证明了作者设计的 MPTCP 算法虽然为了鲁棒性（避免死锁）牺牲了一点点纯粹的均衡能力，但其负载均衡效果依然非常出色，远好于 EWTCP

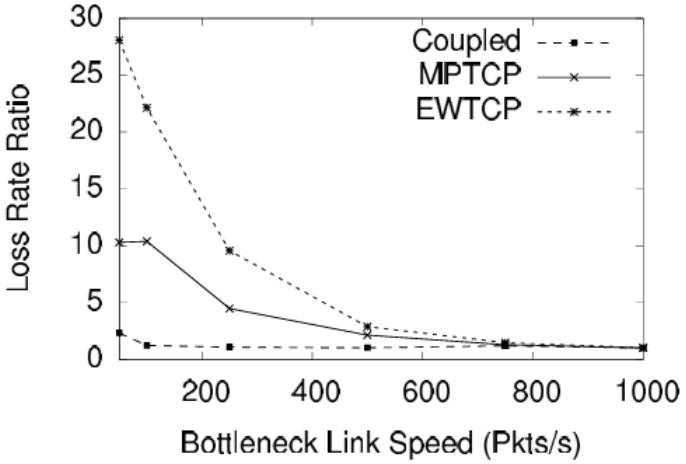


Figure 8: Effect of changing the capacity of link C on the ratio of loss rates p_C/p_A. All other links have capacity 1000pkt/s.

- 纵轴表示 p_C/p_A（C链路丢包率 / A链路丢包率），横轴表示链路 C 的容量（越往左越拥塞）

Congestion control at multihomed server

- 动态负载均衡实验模拟
  - **场景**：两条平行链路（Top 和 Bottom）
  - **干扰**：上方链路（Top）会遭到突发性 CBR 流量的攻击（模拟网络突然变卡），攻击模式：发 10ms（堵死），停 100ms（空闲）
  - 理想情况：当 CBR 攻击开始时，MPTCP 应该迅速把流量切到底部链路；当 CBR 攻击停止时，MPTCP 应该迅速切回上方链路，利用两条路的带宽

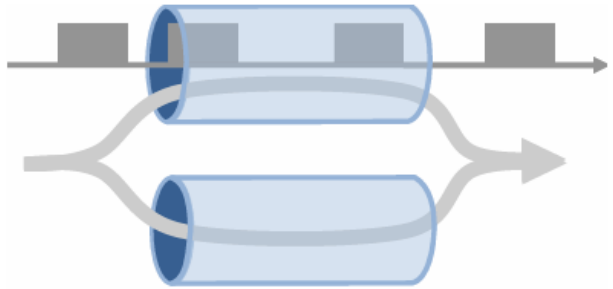


Figure 9: Bursty CBR traffic on the top link requires quick response by the multipath flow.

Congestion control at multihomed server

- 对比效果
  - COUPLED 表现最差，当上方堵死时，COUPLED 切到底部；但当上方变空闲时，COUPLED 因为之前把上方窗口减到了 0，不再发包探测，导致它不知道上方已经变好了。结果就是总吞吐量很低（被困在了单条路径上）
  - MPTCP 实现了良好的平衡，它在 CBR 攻击时能避开拥塞，攻击结束时能迅速恢复双路传输。因为它保留了探测流量（Semi-Coupled 机制）

	top link	bottom link
EWTCP	85	100
MPTCP	83	99.8
COUPLED	55	99.4

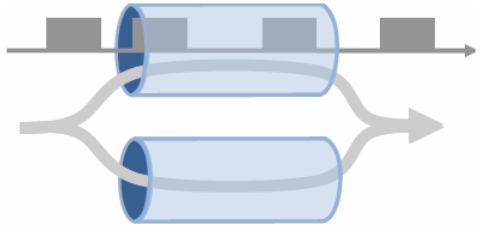


Figure 9: Bursty CBR traffic on the top link requires quick response by the multipath flow.

Congestion control at multihomed server

- 真实环境测试 (Linux Testbed)
  - 硬件拓扑：一台双宿主（Dual-homed）服务器，拥有两条 100Mb/s 的上行链路
    - Link 1（轻负载）： 只有 5 台客户端连接，运行常规的 Linux 2.6 NewReno TCP 流量
    - Link 2（重负载）： 有 15 台客户端连接，同样运行常规 TCP 流量

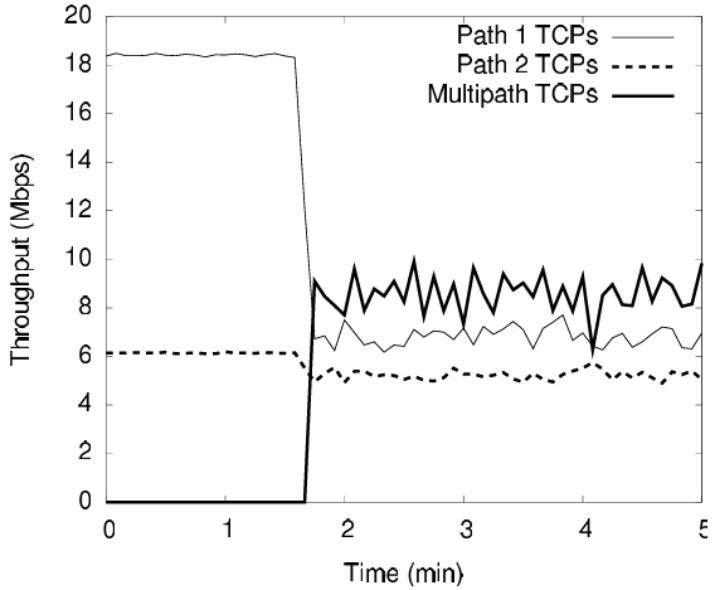


Figure 10: Server load balancing with MPTCP

- **第一阶段（0 - 1.5 分钟）**：Path 1 TCPs 的吞吐量很高（约 18 Mbps），因为 Link 1 上只有 5 个流争抢带宽；Path 2 TCPs 的吞吐量很低（约 6 Mbps），因为 Link 2 上有 15 个流在拥塞，此时网络负载极不均衡

Congestion control at multihomed server

- 真实环境测试 (Linux Testbed)
  - **第二阶段（1.5 分钟后）**：MPTCP 流加入
    - MPTCP 曲线（实线）上升： 获得了约 6-9 Mbps 的吞吐量
    - Path 1 TCPs 曲线下降： 从 18 Mbps 骤降至 6-8 Mbps 左右
    - Path 2 TCPs 曲线基本不变： 维持在 5-6 Mbps 左右
  - **结论**： MPTCP 并没有平均地分配流量到两条链路，而是智能地将绝大部分流量导向了原本空闲的 Link 1，从而自动平衡了服务器两条上行链路的负载

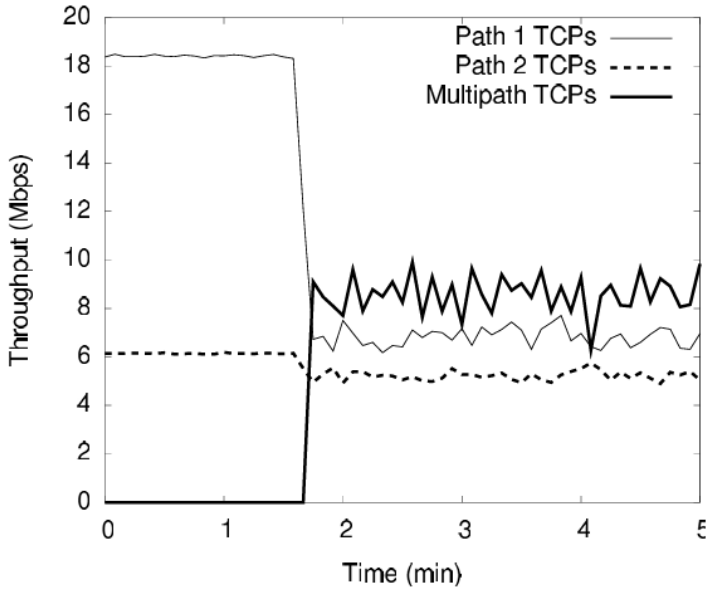


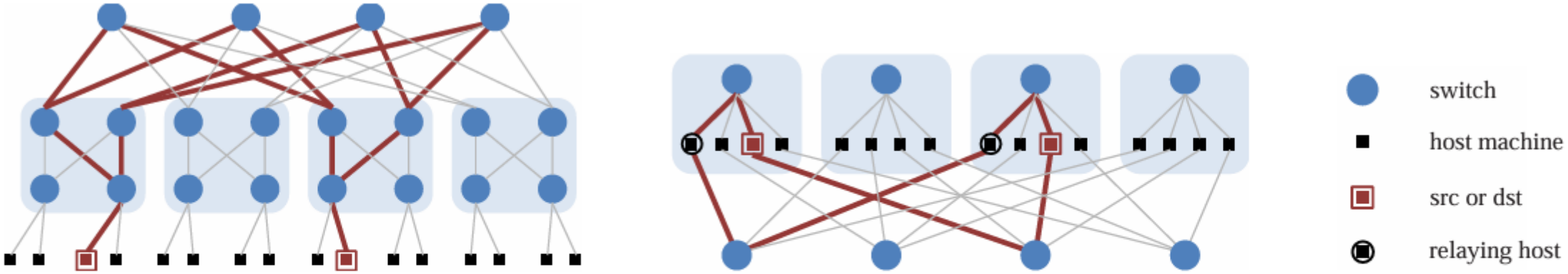
Figure 10: Server load balancing with MPTCP

Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Congestion control in data center

- 谷歌、微软和亚马逊等公司的云应用不断增长，催生了大型数据中心
- 在这些数据中心中，大量流量在机器之间传输，而不仅仅是传向互联网；为支持这一点，研究人员提出了新的架构（FatTree，Bcube），其互连密度远高于传统实现方式。互连的高密度意味着任意两台机器之间存在许多可能的路径
- 面临的挑战是：无论流量模式如何，我们如何确保负载得到有效分配?



(a) FatTree (b) BCube

Figure 11: Two proposed data center topologies. The bold lines show multiple paths between the source and destination.

Congestion control in data center

- 仿真实验设置
  - 作者使用包级模拟器对比了 MPTCP、EWTCP 和 单路径 TCP (Single-Path TCP + ECMP)
  - FatTree: 128 台主机，80 个交换机，MPTCP 为每对主机随机选择 8 条路径
  - BCube: 125 台主机（每台 3 个接口），25 个交换机
- 流量模式 (Traffic Patterns - TP)：
  - TP1 (Random Permutation): 随机一对一通信，这是测试网络能否达到全二分带宽（Full Bisection Bandwidth）的基准 。
  - TP2 (Locality/Replicas): 每个主机向 12 个目的地发送流量（模拟分布式文件系统写副本），这是对流量局部性的压力测试 。
  - TP3 (Sparse): 只有 30% 的主机发送流量，这是测试在网络轻载下能否充分利用主机带宽

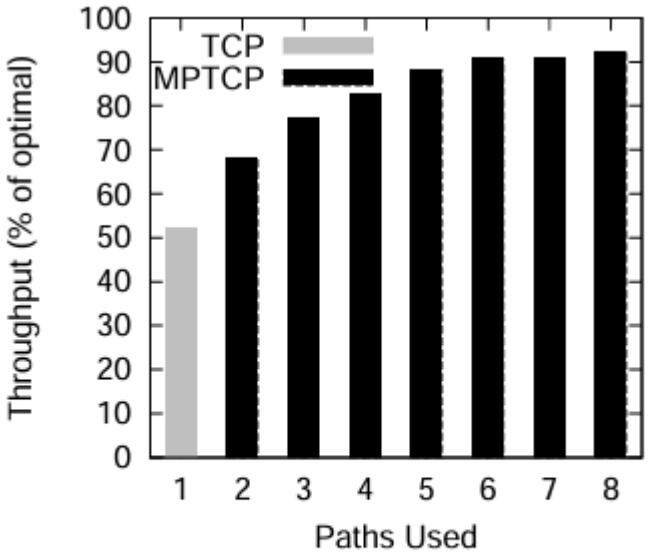


Figure 12: Multipath needs 8 paths to get good utilization in FatTree

Congestion control in data center

- FatTree 拓扑下的表现
- 极高的网络利用率：在 TP1（随机排列）模式下，单路径 TCP 只能达到约 51% 的吞吐量，而 MPTCP 可以达到 95%，接近物理链路极限 ,这证明 MPTCP 能有效消除 ECMP 哈希冲突带来的热点
- 子流数量的选择（关键结论）：作者研究了需要多少个子流才能跑满带宽。结果显示，8 个子流足以在 FatTree 中获得 90% 的网络利用率
- 公平性优势：与 EWTCP 相比，MPTCP 不仅平均吞吐量高，而且在流之间的公平性（Fairness）更好，丢包率分布更均匀

	TP1	TP2	TP3
SINGLE-PATH	51	94	60
EWTCP	92	92.5	99
MPTCP	95	97	99

- 在不同流量模式（TP）下的带宽大小，单位Mb/s

Congestion control in data center

- BCube 拓扑下的表现
- 多接口利用：在 TP3（稀疏流量）中，MPTCP 能够利用主机的全部 3 个物理接口，吞吐量（135Mb/s）远超单路径 TCP（78Mb/s），后者受限于单接口瓶颈
- 路径选择的智能性：由于 BCube 中存在不同跳数的路径，长路径更容易拥塞。MPTCP 比 EWTCP 更能有效地将流量从拥塞的长路径转移到短路径上
- 在 TP2 模式下，单路径 TCP 的吞吐量（297Mb/s）略高于 MPTCP（272Mb/s）。单路径 TCP 默认只选最短路径；而 MPTCP 为了探测拥塞，会强制保留少量流量在非最短（更拥塞）的路径上，导致整体效率微弱下降

	TP1	TP2	TP3
SINGLE-PATH	64.5	297	78
EWTCP	84	229	139
MPTCP	86.5	272	135

- 在不同流量模式（TP）下的带宽大小，单位Mb/s

Congestion control in data center

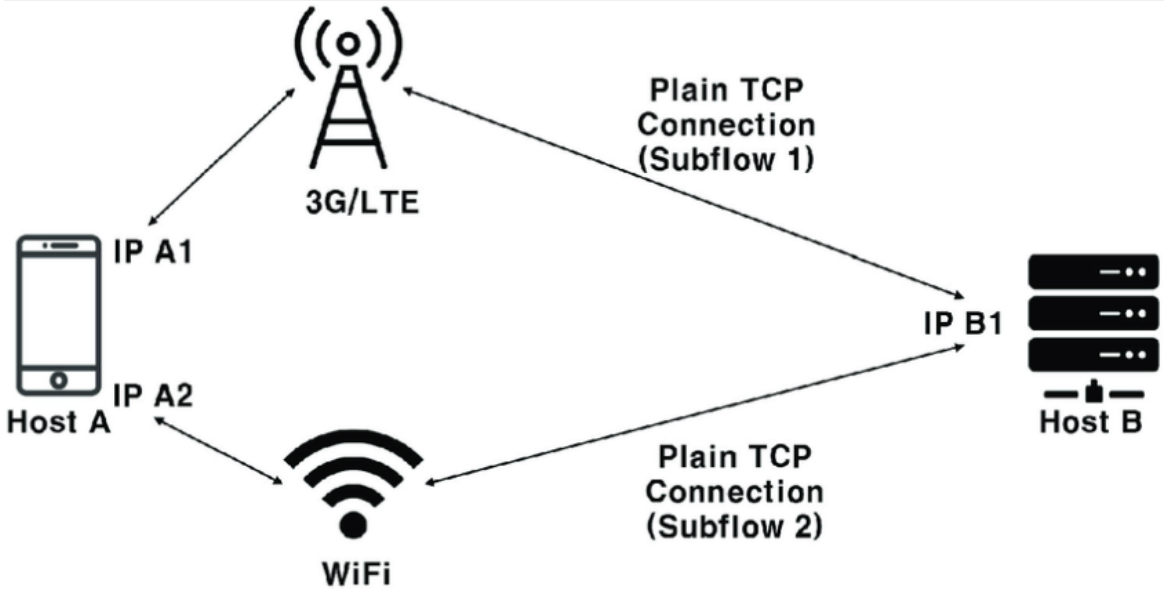
- 与集中式调度 (Centralized Scheduling) 的对比
  - SDN 风格的集中式流调度（如 Hedera）要达到与 MPTCP 相当的性能，集中式调度器可能需要每 100ms 就重新计算一次路由
  - 这在大规模网络中会带来巨大的控制平面开销和扩展性问题。相比之下，MPTCP利用拥塞控制作为隐式的反馈回路，具有更好的扩展性
- 数据中心网络中，MPTCP 可以被视为一种高效、分布式的负载均衡机制。它不需要修改交换机硬件或核心路由协议，仅仅通过端主机的拥塞控制改进，就能在 FatTree 和 BCube 等复杂拓扑中获得接近最优的吞吐量和公平性

Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Congestion control in wireless network

- 主要通过实验验证了 MPTCP 算法在 异构无线网络环境（同时使用 WiFi 和 3G）下的表现。
- 现代移动电话和设备拥有多个无线接口，但在任何特定时间，只有其中一个接口用于数据传输。多路径技术可以通过允许同时使用这两种接口来改善移动用户的体验
  - WiFi: 低 RTT，高带宽，但由于干扰导致丢包率较高
  - 3G: RTT 很长（通常超过1秒，因为缓冲区过大），但丢包率极低



Congestion control in wireless network

- 静态单流实验 (Single-flow experiment)
  - 目的： 验证在没有竞争流量的情况下，MPTCP 能否聚合两条链路的带宽
  - 设置： 笔记本电脑位置固定，同时通过 3G 和 WiFi 连接服务器
    - 单路径 WiFi 吞吐量：14.4 Mb/s
    - 单路径 3G 吞吐量：2.1 Mb/s
    - MPTCP 吞吐量：17.3 Mb/s
- 结论： MPTCP 成功实现了带宽聚合，其总吞吐量大约等于两条链路带宽之和

Congestion control in wireless network

- 竞争流实验与 RTT 补偿 (Competing-flows experiment)
  - 目的：验证了当存在竞争流量且链路 RTT 差异巨大时，算法是否还能保持“公平”且“高效”
  - 设置：如右图，MPTCP 流与两条单路径 TCP 流（一条在 WiFi 上，一条在 3G 上）竞争带宽
  - Link 1 (WiFi): 模拟 WiFi 接口，特点是 RTT 短，但由于干扰丢包率相对较高
  - Link 2 (3G): 模拟 3G 接口，特点是 RTT 长（由于缓冲区大），但丢包率低
    - 流 S_1: 一个标准的单路径 TCP 流，仅运行在 Link 1 上
    - 流 S_2: 一个标准的单路径 TCP 流，仅运行在 Link 2 上
    - 流 M: 一个 MPTCP 流，同时使用 Link 1 和 Link 2

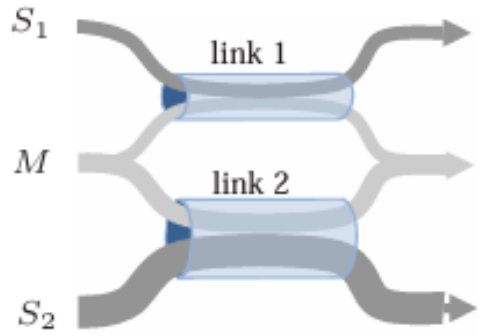


Figure 14: A multipath flow competing against two single-path flows

Congestion control in wireless network

➢ 竞争流实验与 RTT 补偿 (Competing-flows experiment)

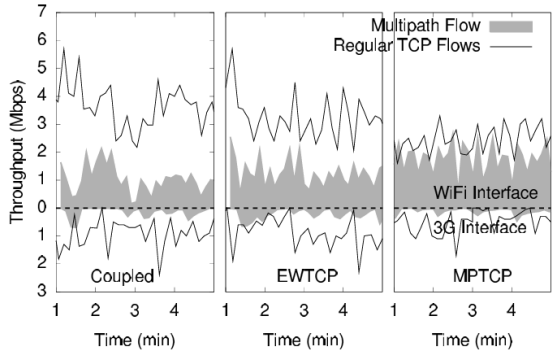


Figure 15: Multipath TCP throughput compared to single-path, where link 1 is WiFi and link 2 is 3G.

- 上半部分（正向）： 显示 WiFi 接口（Link 1）的吞吐量。
- 下半部分（反向）： 显示 3G 接口（Link 2）的吞吐量（为了方便对比，Y轴向下延伸）。
- 灰色区域： 代表多路径流（Flow M）在两个接口上的总吞吐量。
- 黑色实线： 代表单路径竞争流（S1 或 S2）的吞吐量

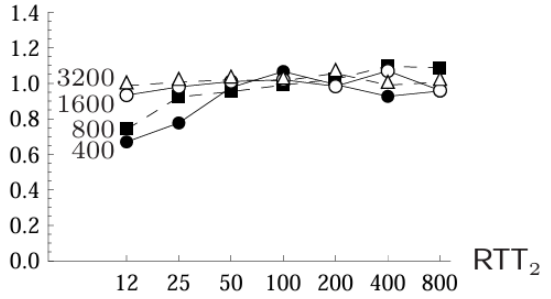


Figure 16: The ratio of flow M's throughput to the better of flow S_1 and S_2, as we vary link 2 in Fig.14.

- 横轴 (RTT2): Link 2 的往返时延
- 纵轴 (Ratio): MPTCP 流的吞吐量与 最好的单路径流（S1 或 S2）吞吐量之比

- COUPLED 算法的设计原则是将流量移向“丢包率最低”的路径，平均吞吐量1.41 Mb/s；EWTCP 只是简单地在每条路径上运行加权的 TCP，平均吞吐量1.66 Mb/s
- MPTCP引入了 **RTT 补偿机制，**最高的总吞吐量（平均 2.21 Mb/s），这证明了它既能聚合带宽，又不会被低速链路“拖后腿”

- 在几乎所有的 RTT 设置下（即便是 RTT 差异巨大的 800ms），比值都保持在 1.0 或略高 的位置
- 这意味着，无论链路的 RTT 差异有多大，MPTCP 算法总是能获得至少等于最佳单路径的吞吐量，平均提升约为 15%

- 结论：在各种带宽和 RTT 组合下，MPTCP 获得的吞吐量总是比“只使用最好的一条链路”要高，平均提升约 15%

Congestion control in wireless network

- 移动性与切换实验 (Mobile experiment)
  - 目的：验证在真实移动场景下，MPTCP 处理信号衰减和网络切换的鲁棒性
  - 过程： 用户拿着笔记本在楼层间走动
    - 初始阶段： 有 WiFi 覆盖。MPTCP 发现 3G 虽拥塞较低但速率慢，因此限制 3G 使用，主要流量走 WiFi，保证不抢占其他 3G 用户
    - 移动阶段（9分钟时）： 用户走到楼梯间，WiFi 信号丢失。MPTCP 无缝切换，利用 3G 保持连接继续传输
    - 恢复阶段： 用户离开楼梯间，连接到新的 WiFi 基站。MPTCP 迅速发现新路径并恢复高吞吐量传输

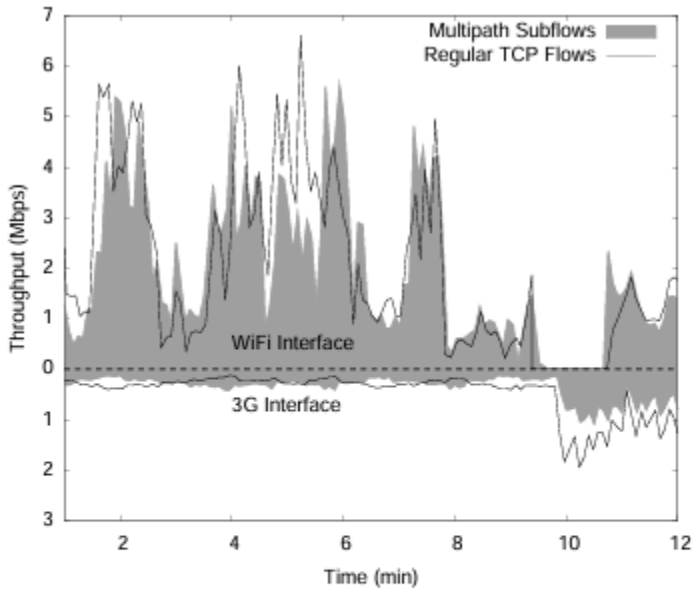


Figure 17: Throughput of multipath and regular TCP running simultaneously over 3G and WiFi. The 3G graph is shown inverted, so the total multipath throughput (the grey area) can be seen clearly.

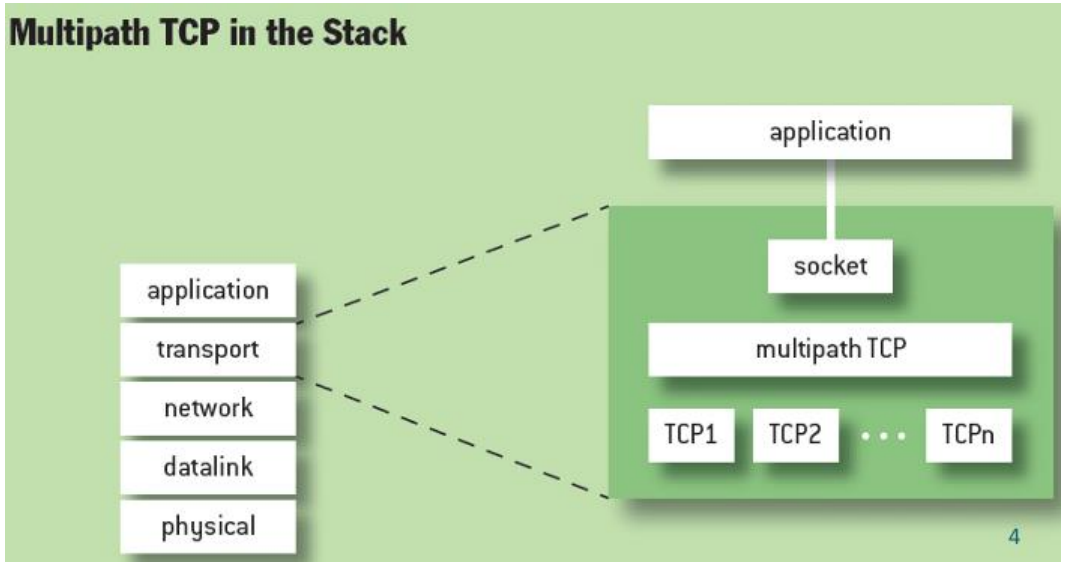
- 图中灰色区域（MPTCP 总吞吐量）始终保持连接，证明了 MPTCP 相比传统“断开-重连”机制的优越性

Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Protocol implementation

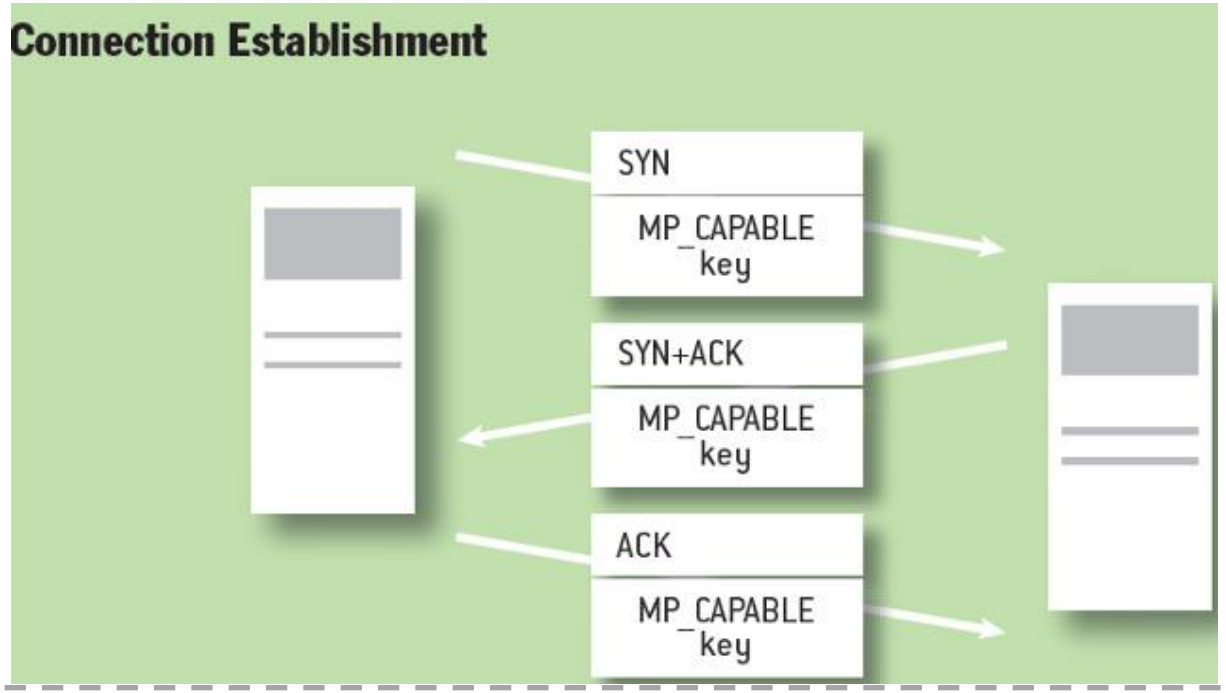
- 算法层面的优越性已得到验证，但在真实互联网中部署还面临系统级挑战……
- 算法只解决了“发多少”(How much to send)，但协议栈需要解决“怎么发”(How to send safely)：
  - 中间设备干扰 (Middleboxes)： 防火墙修改序列号会导致无法重组数据流?
  - 死锁风险 (Deadlocks)： 多路径接收缓存满了会导致互相等待?
  - 状态同步 (Signaling)： 如何在不造成死锁的情况下确认数据接收?



Protocol implementation

➢ MPTCP 的三次握手

- ➢ MPTCP依然按照正常的TCP进行三次握手，只是在握手过程中增加了MPTCP特有的信息
- ➢ 左边客户端发送的第一个SYN包携带有客户端自身的KEY，右边发送SYN/ACK的时候携带了自身的KEY，而最后左边的客户端发送最后一个ACK的时候携带着双方的KEY



Protocol implementation

- 子流建立 (Subflow Establishment)
- MPTCP 需要在两个端点之间建立和管理多个 TCP 子流
- 握手机制： 采用 TCP 选项（TCP Options）进行协商
  - 在第一个子流的 SYN 包中，携带选项以协商是否使用 MPTCP
  - 后续建立的新子流，其 SYN 包中携带关联选项（Joining Option），以便接收端将其绑定到现有的 MPTCP 连接上
- 路径获取： 当前实现依赖于多 IP 或多接口来创建不同的路径（由标准路由机制决定具体路由），暂未涉及何时启动新子流的策略研究

Protocol implementation

- 序列号分离 (Loss Detection and Stream Reassembly)
  - 问题背景： 标准 TCP 使用一个序列号空间（Sequence Space）同时完成两件事：**丢包检测** 和 **数据流重组**
  - MPTCP 的问题所在：如果为了重组方便，将一个序列号空间条带化（Striping）分配给不同子流，如果某个子流的序列号被中间件修改，接收端就无法还原整个数据流
  - 解决方案：双层序列号设计
    - 子流序列号 (Subflow Sequence Number, SSN)： 保留在 TCP 头部的标准序列号字段中。仅用于该特定子流上的链路级丢包检测和重传
    - 数据序列号 (Data Sequence Number, DSN)： 添加在 TCP 选项中。用于应用层数据流的端到端重组，告诉接收端这段 Payload 属于整个文件的哪个位置

Protocol implementation

- 流量控制与接收缓存 (Flow Control & Buffering)
  - 问题：如何通告接收窗口（Receive Window）以避免死锁?
  - 方案 A（被弃用）： 每个子流有独立的接收 Buffer
    - 死锁风险： 假设子流 1 丢包停滞，但子流 2 的 Buffer 填满了。由于数据按序提交给应用层，应用层在等子流 1 的数据，无法读取子流 2 的数据。此时子流 2 Buffer 满，不仅无法接收新数据，甚至可能无法接收用来“救急”的重传数据，导致死锁
  - 方案 B（最终采用）：单一共享接收 Buffer (Single Shared Buffer)
    - 所有子流共享一个大的接收池
    - 接收窗口（Receive Window）是相对于 数据序列号 (DSN) 的连续接收位置来通告的，而不是针对每个子流单独通告

Protocol implementation

- 显式数据 ACK (Explicit Data ACK)
- 问题：不能简单地通过“子流 ACK”来推断“数据 ACK”
- 例如，发送端在慢速链路（Link 1）发了 Pkt 1，在快速链路（Link 2）发了 Pkt 2
  - 接收端先收到 Pkt 2，发回子流 ACK
  - 发送端收到 ACK，知道数据 2 到了，但数据 1 没到
  - 由于路径延迟不同（Reordering），导致发送端很难准确推断接收窗口的尾部（Trailing Edge）。如果推断错误，发送端可能会发送超出接收端缓存能力的数据，导致不必要的丢包
- 解决方案：必须在 TCP 选项中增加显式的数据确认字段 (Data ACK)，直接告诉发送端数据流层面的确认进度，而不能依赖子流层面的 ACK 推导

Protocol implementation

- 编码与死锁避免 (Encoding & Control Deadlock)
- 问题：关于 DSN 和 Data ACK 应该放在哪里（TCP 选项还是 Payload）？
- Payload 方案的死锁风险：
  - 如果将 Data ACK 作为一种特殊的控制块（类似 SSL 记录）放在 Payload 中，那么 ACK 本身也会受到流量控制的限制
  - 死锁场景： A 的接收 Buffer 满了（应用层没读），A 想给 B 发送数据。B 接收了数据想回发 Data ACK。但由于 A 的 Buffer 满了，B 发不出来 Data ACK。A 收不到 ACK 就无法释放发送 Buffer，导致 A 的发送 Buffer 也满了。最终 A 和 B 互相等待，形成死锁
- 解决方案：为了彻底避免流控死锁，所有的控制信号（DSN 和 Data ACK）都必须放在 TCP 选项 (TCP Options) 中，因为 TCP 选项不受接收窗口大小的限制

Outline

I. Introduction
II. Design problem for MPTCP congestion control
III. Congestion control at multihomed server
IV. Congestion control in data center
V. Congestion control in wireless network
VI. Protocol implementation
VII. Review

Review

- MPTCP 的优点在哪，它是如何工作的?
- Concerns of MPTCP congestion control
  - Fairness with TCP flow
  - Discover and make use of less congested path
  - Load change
  - RTT mismatch
- 文中提到的EWTCP、Coupled算法的不足，体会作者是如何一步步修正并得出最终的算法
  - EWTCP: subflows are independent, one flow congested, other subflows will not be more aggressive
  - Coupled: only use the least congested path, can not discover other paths when they are good

Review

➢ Related Work

- ➢ 本文的核心是拥塞控制而非协议设计，现有多数方案仅考虑了共享瓶颈问题，而未考虑全面
  - ➢ pTCP 、SCTP 上的并发多路径传输（CMT）、M/TCP ： 在每条路径上采用独立拥塞控制，在一般场景下对竞争单路径流量不公平
  - ➢ mTCP ： 同样采用独立拥塞控制，通过计算不同子流上快速重传间隔的相关性检测共享瓶颈，但该检测器的鲁棒性尚不明确
  - ➢ R-MTP：针对无线链路设计，定期探测每条子流的可用带宽并调整速率；通过数据包到达间隔和抖动检测拥塞（抖动增加表明拥塞加剧），但仅在无线链路为瓶颈时有效

Review

- 网络层多路径
  - 等价多路径路由（ECMP）在流级别实现负载均衡，无需终端系统参与 —— 同一流的所有数据包沿同一路由传输，避免终端系统接收乱序数据包
  - ECMP 与 MPTCP 互补：MPTCP 可利用 ECMP，在无需多宿主终端的情况下获得多条路径；同一多路径连接的不同子流具有不同的五元组（至少一个端口不同），ECMP 会将其映射到不同路径。这种协同作用在数据中心中尤为实用（数据中心存在多条路径，且广泛部署 ECMP）

Review

- 应用层多路径
  - BitTorrent 是应用层多路径的典型示例：从不同对等节点下载同一文件的不同数据块，提升吞吐量。BitTorrent 以数据块为粒度，仅优化吞吐量（从更快的服务器下载更多数据块），其行为类似独立多路径拥塞控制（尽管路径的端点不同）。尽管独立拥塞控制无法平衡流速率，但考虑到流大小的差异，通过非拥塞子流获得更高吞吐量，减少拥塞子流的数据量，仍能实现一定程度的负载均衡