STMS: Improving MPTCP Throughput Under Heterogeneous Networks

专业：计算机技术硕 1805 班 姓名：杨向杰 学号：M201873143

1 背景

随着网络用户和应用的快速增长，例如云计算和虚拟现实等，对网络带宽有着强烈的需求。为了在网络中获得较高的吞吐量，已经引入了通过实现多路 TCP（MPTCP）协议[1]等方式来提高带宽。例如，许多无线的设备都有两个网络接口，一个到局域的 WiFi 网络，另一个到广域的蜂窝网络。在现实使用情况下，由于无线带宽是有限的，数据流可以通过两个网络路径来提高传输速率。但是 WIFI 和 LTE 网络两者是异构的，两者之间的 RTT（Round-Trip-Time）差别很大[2]。

2 MPTCP 原理和存在问题

MPTCP 原理

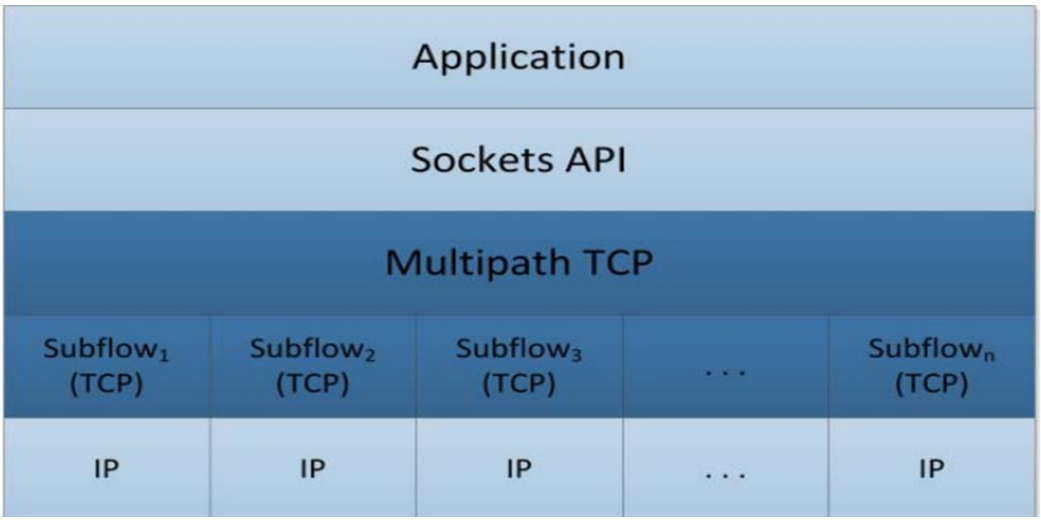

图 1 MPTCP 协议栈结构

Multipath TCP 是指与传统单路径 TCP 相比，在传输数据时使用多条传输路径去传输数据，以此来提高最大化资源利用率和吞吐量。MPTCP 默认的调度原理：在所有的可用传输路径中选择最快的路径发送数据包，由此带来的优势就是：

（1）明显提高了吞吐量，与传统 TCP 相比，由于 MPTCP 可以在两个端点之间同步的发送数据，因此明显提高了吞吐量。

（2）明显提高了网络垂直切换的速度，对于移动用户来说，由于用户可能移入或移出 WIFI 以及 LTE 网络,而 MPTCP 保持多种连接，因此 可以在异构的网络环境中快速垂直切换等。

（3）在数据中心环境下，MPTCP 与传统 Ethernet 相比，它可以通过多个接口来平衡一个单路的 TCP。

（4）失败连接时，有着更快速的反应。

MPTCP 存在问题

(1) 导致需要更大的 host buffer

MPTCP 默认从最快的可用路径发送数据包，那么就会导致从 slow-path 发送的数据包到达接收方会晚一些，在接收方便会导致数据包的无序到达（out of order），因此，需要更大的 host buffer 缓存先到达的数据包。如图 2 所示，T=3 时，从 slow-path 传输的数据包才会到达，而通过 fast-path 传输的数据包 2、4 早已到达接收方，于是形成乱序到达，所以需要更大的 receive buffer[3]。

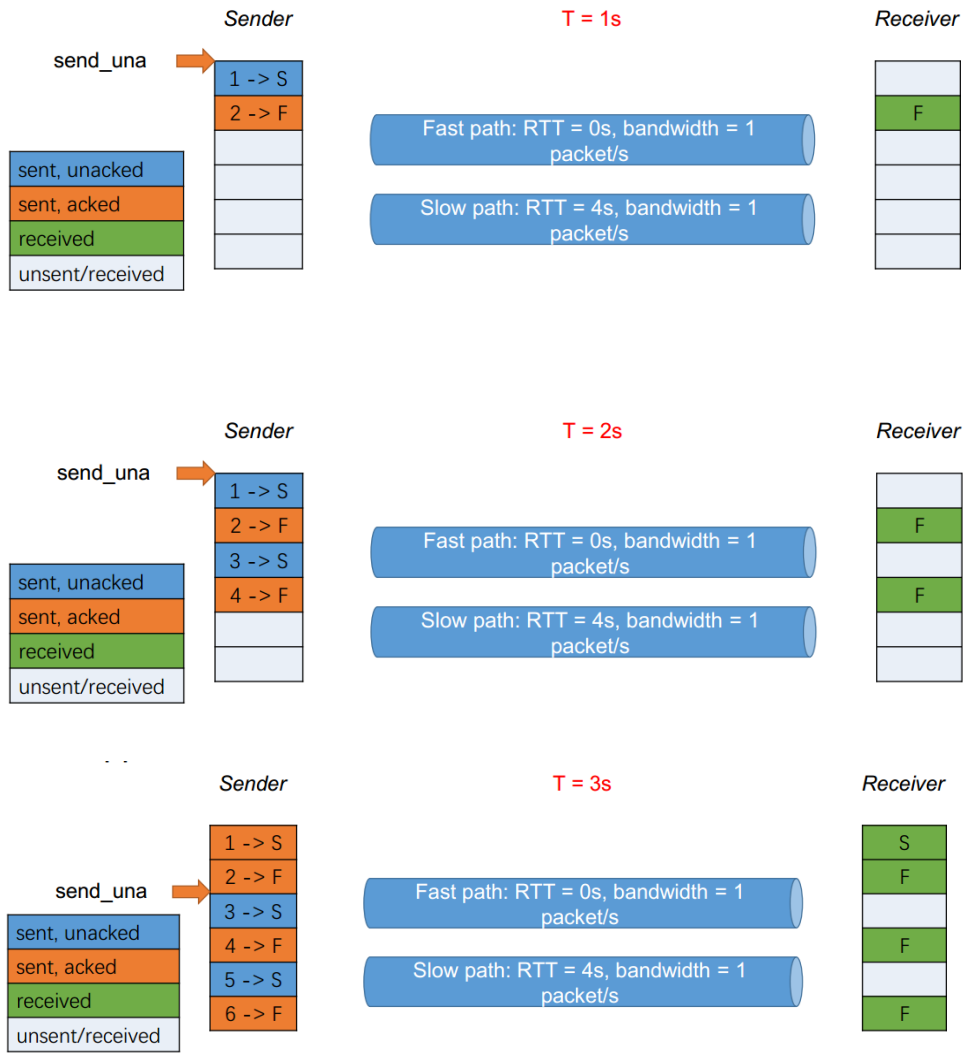

图 2 MPTCP 传输过程示意

(2) MPTCP 需要更大的 in-network buffer

在 MPTCP 中，当接收到有顺序的数据包时，每一个包都会有相应的 Data ACK 信号[4]。但是当接收无序的数据包时，直到接收从 slow-path 传输的数据包之后，Data ACK 将同时确认从 fast-path 到达的所有数据包，那么确认之后，fast-path 将有足够的空间同时发送多个数据包，这就会导致大量突发性（burst sending）的发送行为[5]。因此，

如果 network buffer 容量不够大的话，就会导致丢包或者堵塞等行为。如图 3、4 所示，为了达到与 single-TCP 相同的吞吐量，双路的 MPTCP 的缓冲区必须从 30KB 增加到 150KB。

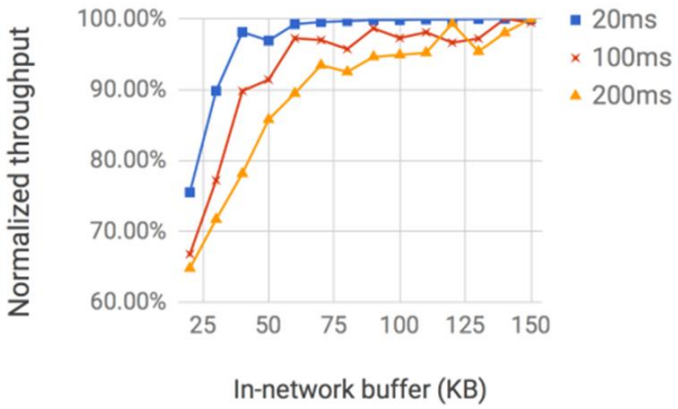

图 3 当 network buffer 有限时，MPTCP 吞吐量降低

in-network buffer/KB	observed loss rate	Fast path in MPTCP/Mbps	Single TCP/Mbps	Utilization
30	0.05%	12.1	28.4	42.76%
60	0.02%	20.8	28.4	73.50%
90	0.02%	25	28.4	88.34%
150	0.01%	26.5	28.4	93.64%

图 4 不同 in-network buffer 下 fast-path 丢失率

3 改进 MPTCP 调度算法（STMS）

STMS 核心思想

本文实现的 STMS 调度算法的核心思想就是预分配数据包，对数据包进行标号，为 fast sub-flow 缓冲先到达的数据包，并将具有较大序列号的数据包分配给 slow sub-flow，使它们按顺序到达。

Algorithm 1 Slide Together Scheduler

```
1: procedure ST_SCHEDULE(unsentPackets)                ▷
   Scheduler runs when one of sub-flow is available
2:     if Fast sub-flow has space in send window then
3:         Fast sub − flow ←unsentPackets[0]
4:     elseSlow sub-flow has space in send window
5:         Slow sub − flow ←unsentPackets[Gap]
6:     end if
7: end procedure
```

图 5 调度算法核心思想

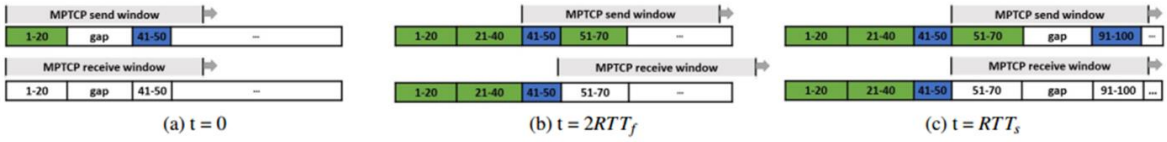

图 6 调度算法核心思想

本文所实现的 STMS 算法运行过程如图 5 和图 6 所示。当至少有一个路径可以用于发送分组数据包时，scheduler 开始运行。fast sub-flow 总是在缓冲区中发送具有最小序列号集的分组数据包。如图 6 所示，slow sub-flow 发送具有更大序列号的分组，这时候不是紧接着 fast path 右边序号继续发送，而是为将来 fast path 继续发送相应数据包留下一个序列 Gap。所以要确保当来自 slow path 的分组到达时，来自 fast path 的具有较小序列号的所有分组应该已经到达。这时候所有的数据包是按顺序到达，因此确保了正常的 ACK 时钟，fast path 也没有产生突发传输（burst sending），并且不会阻止发送/接收窗口。

STMS 调度算法的关键点

本文改进的 STMS 调度算法的关键点是 Gap 的计算，它是为从 fast path 发送预分配的数据包数。调度算法的效率取决于 Gap 的准确性。与真实值的任何偏差都将导致数据包的无序到达。

最简单的获得 Gap 值方法是测量网络路径状况。如果我们能够准确地测量网络状况，那么我们可以通过以下方式得出真实的 Gap 值。对于 Gap 中的所有数据包，通过 fast path 到达接收方需要 OWDf + Gap/Bf，其中 Bf 是 fast path 的带宽。这个时间应该等于 OWDs，以便来自 slow path 的第一个数据包与来自 fast path 的数据包同时到达。然后我们就可以得到真正的 Gap 值：

$$\text{True Gap} = B_f \times (OWD_s - OWD_f)$$

但是这个方法有两个缺陷：其一是单路延迟（one way dalay）测量不可能那么精确；其二是当 in-network buffer 被限制时，路径中带宽测量也不精确。因此，本文设计了基于反馈的 Gap 调整方案，以便更准确，更快速地调整 Gap。

Gap adjust 方法

Algorithm 2 Gap Adjustment Algorithm

```
1: procedure GAP_ADJUST(data_acked)          ▷ This function gets called when receiving Data ACK
2:     if data_acked > 2 then
3:         send_una ← left edge of MPTCP-level send window
4:         if send_una was sent from slow path then
5:             delta_gap = data_acked
6:         elsesend_una was sent from fast path
7:             delta_gap = −data_acked
8:         end if
9:     end if
10:    gap+ = EWMA(delta_gap,adjust_interval)
11: end procedure
```

图 7 Gap Adjust 方案

在从 slow-path 传输的数据包到达之后，将发送 Data ACK 一次性确认无序的数据包,Data ACK 的数量也就反映了乱序到达的程度，那么就可以根据 Data ACK 动态调整 Gap Value。

如图 7 所示，令 delta_gap 和 adjust_interval 是调整参数。那么当 Gap 小于 True Gap 时，来自 slow path 的数据包迟到，不会有 Data ACK，所以 MPTCP 级别的发送窗口将被 slow path 发送的数据包停止。因此，发送窗口的左边缘由来自 slow path 的未确认分组（其实就是 fast path 中没有被确认的数据包）确定。对称地，当 Gap 大于 True Gap 时,发送窗口的左边缘将由 fast path 发送的分组确定。每次我们收到 Data ACK 时，我们首先计算此 Data ACK 确认的数据包数量（数据已确认）。如果数据大于 2，我们将调整 Gap 值。我们检查发送窗口左边缘的数据包。如果数据包是从 slow path 发送的第一个数据包,则 delta_gap = data_acked;否则,delta_gap = -data acked。

我们使用 delta_gap 的指数加权移动平均线（EWMA）来调整 Gap 值。adjust_interval 是一个可调参数，用于确定 Gap 调整对网络状况变化的反应速度。将其设置得太小会导致 Gap 振荡，因为之前的 Gap 调整尚未生效。但是，将其设置得太大会导致收敛时间变慢。

STMS 算法两种变形

我们实现了两种 STMS 变形：STMS-C 和 STMS-A。它们都为 fast sub-flow 预分配数据包，以便数据包可以在接收方按顺序到达。他们的不同在于如何计算 Gap value。每次从 slow path 发送分组时，STMS-C 从子流 TCP 的算法中提取平滑的 RTT 的带宽估计并计算 Gap 值（假设上行链路和下行链路延迟是对称的）。对于 STMS-A，则根据 Gap adjust 算法计算 Gap 值。

4 实验评估

本文在 linux 平台上基于 MPTCP 0.92 版本[6]实现 STMS 算法，证明了在不同条件下 STMS 算法都有不同程度的改善。

在真实环境中 STMS 表现

实验中，Server 部署在阿里云[7]上，client 在校园中，client 通过 WIFI 和 LTE 连接 server（在现实情况下，WiFi 的延迟小于 LTE，如图 8 所示），下载的文件为 200MB，结果如图 9 所示。

	Bandwidth(Mbps)	Latency(ms)
WiFi	40	50
LTE	30	70

图 8 WIFI 和 LTE 的带宽以及延迟

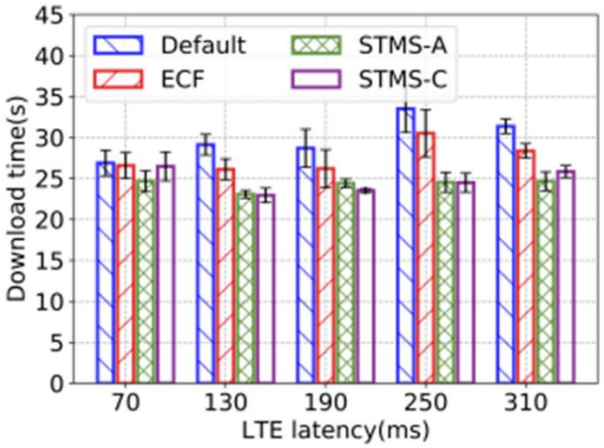

图 9 STMS 在现实网络状况下结果

不同 in-network buffer 和 host buffer

In-network buffer 变化时

如图 10 所示，当限制 in-network buffer 时，大约有 25%提高。

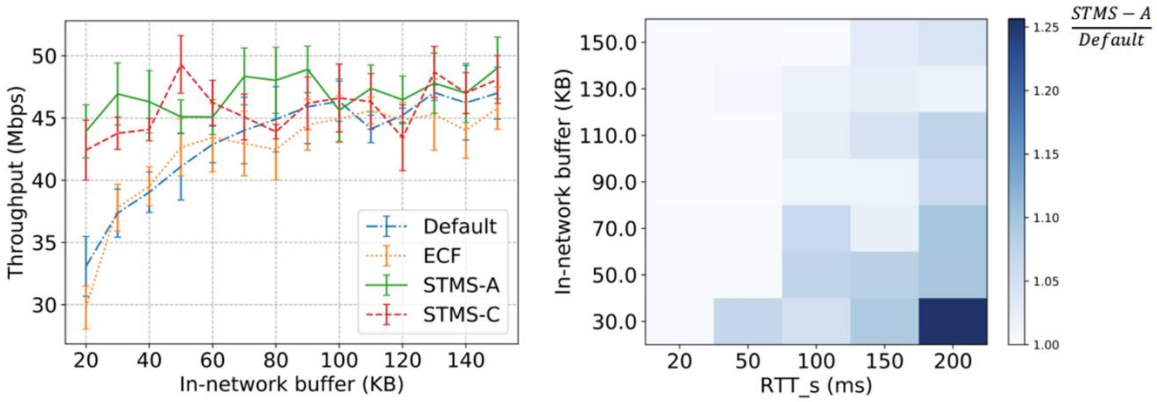

图 10 STMS 在 in-network buffer 变化时的表现

host buffer 变化时

结果如图11所示当限制receive/send buffer时，也大约提高了25%。

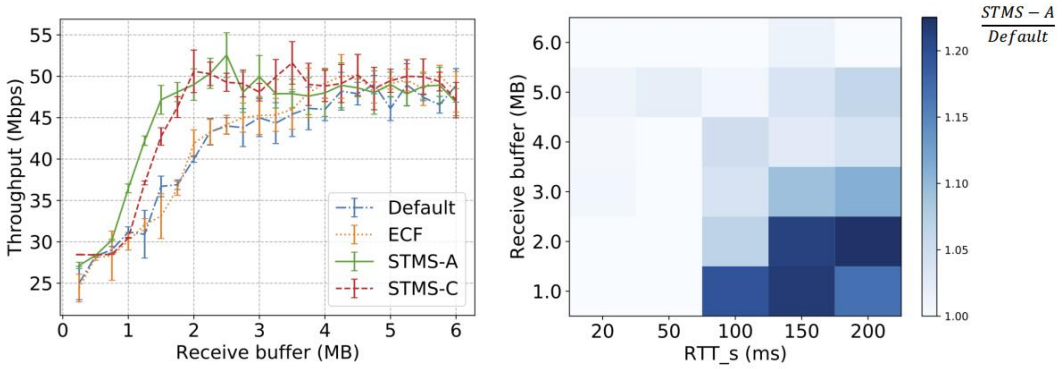

图 11 STMS 在 host buffer 变化时的表现

5 领域的相关工作

关于 MPTCP 改进的研究很多。 为了解决 host buffer 问题，Raiciu 等人提出 PR 机制[8]。费林等人提出了 Blocking Estimation-based Scheduler (BLEST)[9]，其目的是通过减少 slow path 的使用来防止阻塞，即使它具有可用的 CWND 空间。两个调度算法都试图限制 slow path 的使用，以减少对 host buffer 的大容量需求，从而导致 host buffer 的利用率不足。Kuhn 等人建议 DAPS 解决两条路径的 RTT 差异[10]，但它只考虑稳定的 CWND，因此无法对动态网络变化作出快速反应。

6 总结

本文分析了异构的网络路径下 MPTCP 吞吐量降低的根本原因，提出了 STMS，有效的缓解 host buffer 和 in-network buffer 大小受

限而导致的问题，在 STMS 算法的评估实验中也证明了 STMS 在各种网络情况下均有不同程度的改善。

本项目是在国家重点研发项目、国家 863 项目以及华为技术协议研究实验室的支持下完成。

参考文献

[1] FORD, A., RAICIU, C., HANDLEY, M., AND BONAVENTURE, O. Tcp extensions for multipath operation with multiple addresses.Tech. rep., 2013.

[2] JIANG, H., WANG, Y., LEE, K., AND RHEE, I. Tackling buffer bloat in 3g/4g networks. In Proceedings of the 2012 ACM conference on Internet measurement conference (2012), ACM,
pp. 329 – 342.

[3] RAICIU, C., PAASCH, C., BARRE, S., FORD, A., HONDA, M.,DUCHENE, F., BONAVENTURE, O., AND HANDLEY, M. How hard can it be? designing and implementing a deployable multipath tcp. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (2012), USENIX Association, pp. 29 – 29.

[4] JACOBSON, V. Congestion avoidance and control. In ACM SIGCOMM computer communication review (1988), vol. 18, ACM,pp. 314 – 329.

[5] APPENZELLER, G., KESLASSY, I., AND MCKEOWN, N. Sizing router buffers, vol. 34. ACM, 2004.

[6] C. PAASCH, S. BARRE, E. A. Multipath TCP in the Linux Kernel. http://www.multipath-tcp.org.

[7] Alibaba Cloud. https://www.alibabacloud.com/.

[8] RAICIU, C., PAASCH, C., BARRE, S., FORD, A., HONDA, M.,DUCHENE, F., BONAVENTURE, O., AND HANDLEY, M. How hard can it be? designing and implementing a deployable multipath tcp. In Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (2012), USENIX Association, pp. 29 – 29.

[9] FERLIN, S., ALAY, O., M ¨ EHANI, O., AND BORELI, R. Blest:Blocking estimation-based mptcp scheduler for heterogeneous networks. In IFIP Networking Conference (IFIP Networking) and Workshops, 2016 (2016), IEEE, pp. 431 – 439.

[10] KUHN, N., LOCHIN, E., MIFDAOUI, A., SARWAR, G., MEHANI,O., AND BORELI, R. Daps:Intelligent delay-aware packetscheduling for multipath transport. In Communications (ICC),
2014 IEEE International Conference on (2014), IEEE, pp. 1222 – 1227.