



中山大學  
SUN YAT-SEN UNIVERSITY

# C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements

IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, JCR Q1



小组成员：方桂安，陈石翰，陈金华



指导老师：古博

# 目录

CONTENTS

01

背景与挑战

02

相关工作

03

模型与算法

04

评价与分析

05

总结与思考



中山大學  
SUN YAT-SEN UNIVERSITY

Part.01

## 背景与挑战



## 背景

- 新兴应用对时延要求增加
  - 虚拟现实
  - 增强现实
  - 自动驾驶
- 蜂窝网络流量增长

## 挑战

- 蜂窝网络本身问题
  - 高度可变的信道
  - 快速波动的容量
  - 自身造成的排队延迟
  - 随机数据包丢失
  - 无线电上行/下行链路调度延迟。
- 现有方案的缺陷
  - 需要部署新设计的底层交换机
  - 缺乏灵活性，无法适用于不同应用的延迟与吞吐量要求



中山大學  
SUN YAT-SEN UNIVERSITY

Part.02

## 相关工作



## 2 相关文献概述

变体	反馈	所需的更改	优点	公平性
(New) Reno	丢包	—	-	延迟
Vegas	延迟	服务端	更少丢包	成比例
High Speed	丢包	服务端	高带宽	—
BIC	丢包	服务端	高带宽	—
CUBIC	丢包	服务端	高带宽	—
C2TCP	丢包/延迟	服务端	超低延迟和高带宽	—
NATCP	多位信号	服务端	接近最佳性能	—
Elastic-TCP	丢包/延迟	服务端	高带宽/长短距离	—
Agile-TCP	丢包	服务端	高带宽/短距离	—
H-TCP	丢包	服务端	高带宽	—
FAST	延迟	服务端	高带宽	成比例
Compound TCP	丢包/延迟	服务端	高带宽	成比例
Westwood	丢包/延迟	服务端	-	—
Jersey	丢包/延迟	服务端	-	—
BBR	延迟	服务端	-	—
CLAMP	多位信号	客户端, 路由器	-	最大-最小
TFRC	丢包	服务端, 客户端	无重传	最小延迟
XCP	多位信号	服务端, 客户端, 路由器	-	最大-最小
VCP	两位信号	服务端, 客户端, 路由器	-	成比例
MaxNet	多位信号	服务端, 客户端, 路由器	-	最大-最小
JetMax	多位信号	服务端, 客户端, 路由器	高带宽	最大-最小
RED	丢包	路由器	减少延迟	—
ECN	单位信号	服务端, 客户端, 路由器	减少丢包	—

TCP拥塞控制是传输控制协议（TCP）避免网络拥塞的算法，是互联网上主要的一个拥塞控制措施。它使用一套基于AIMD（加增倍减算法）的多样化网络拥塞控制方法（包括慢启动和拥塞窗口等模式）来控制拥塞。左图是基于TCP的一些变体。



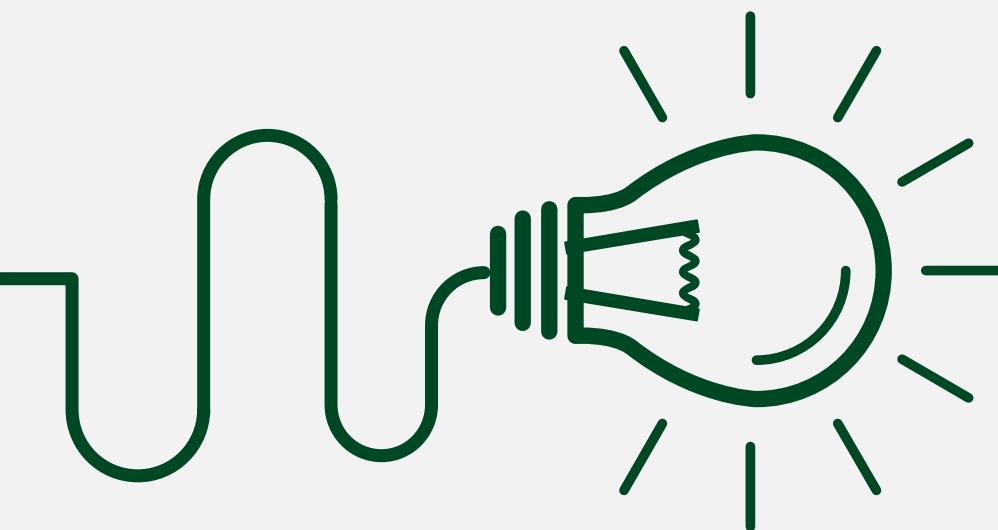
中山大學  
SUN YAT-SEN UNIVERSITY

Part.03

# 模型与算法



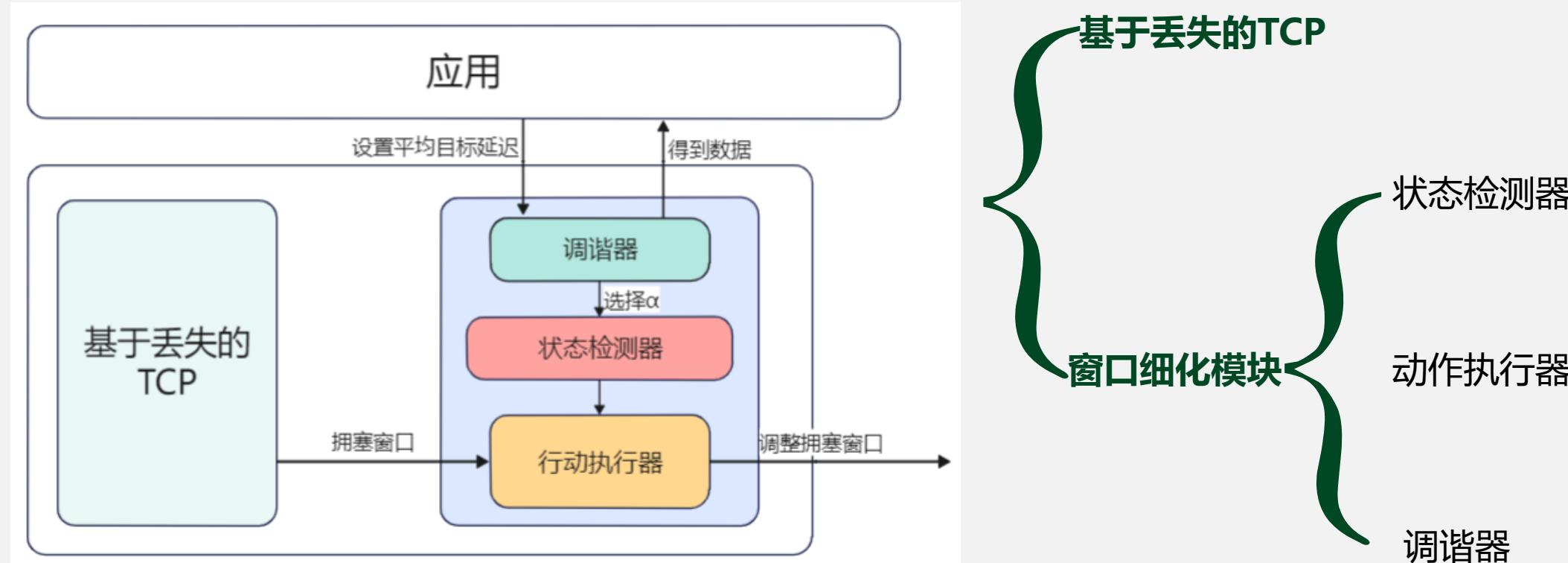
### 3.1 设计原则



- 1 灵活的端到端方案：**运行在不同系统/虚拟机/容器中的各种服务器应用程序(在云/移动边缘)具有不同的目标延迟。
- 2 简单性：**信道的复杂性和不可预测性促使我们避免使用任何信道建模/预测或增加蜂窝网络的复杂性。
- 3 黑盒网络：**延迟源是模糊的。为了降低成本，把蜂窝网络看作一个“黑盒”，它不直接提供任何关于它自身的信息。



### 3.2 整体架构





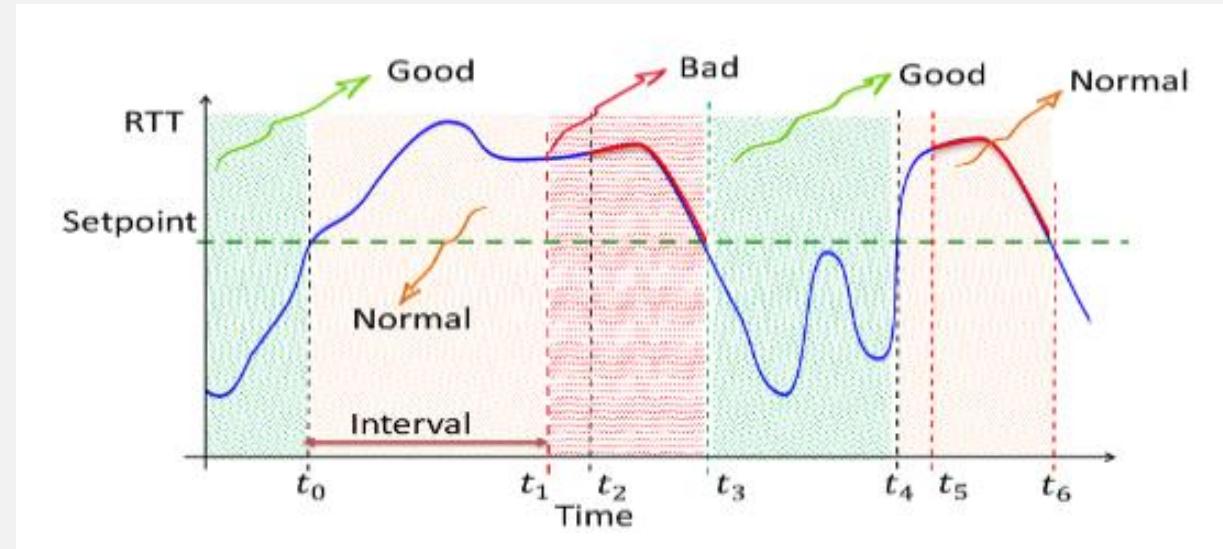
### 3.2.1 状态检测器

定义：

RTT(t)：在时间t接收的ack分组的往返时延

Interval：移动时间窗口的大小

Setpoint：在t时刻所需的最小RTT



- $B - C : \min(RTT(t')) \geq Setpoint \text{ for } (t - Interval) \leq t' \leq t.$
- $N - C : RTT(t) \geq Setpoint \text{ and } \min(RTT(t')) < Setpoint \text{ for } (t - Interval) \leq t' \leq t.$
- $G - C : RTT(t) < Setpoint$

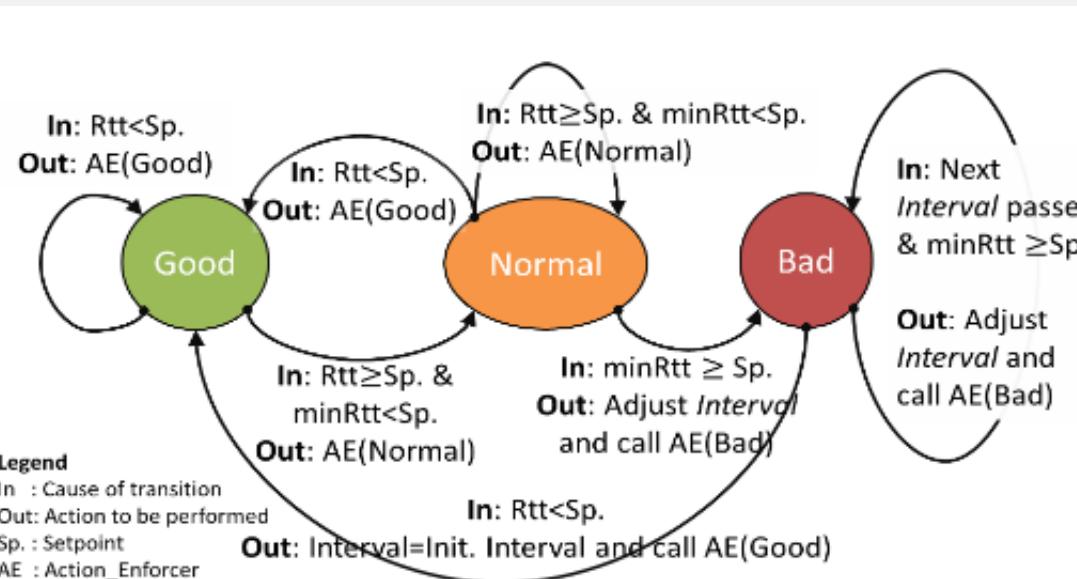


### 3.2.1 状态检测器

调整移动时间窗口的大小，每次乘于 $1/\sqrt{N}$ ，N为监测到的连续坏条件的个数。

在状态良好时，将在收到每个Ack包后调用Action Enforcer(动作执行器)。但是，在每个错误条件开始时就会调用动作执行器。

Initial Interval=Setpoint= $\alpha \times \text{MINRTT}$



Algorithm 1: Condition Detector's Logic

```

1 Function pkts_acked() // process a new received Ack
2   rtt ← current_rtt
3   now ← current_time
4   action_required ← false
5   if rtt < MINRTT then
6     MINRTT = rtt
7     Setpoint =  $\alpha \times \text{MINRTT}$ 
8   if rtt < Setpoint then
9     Interval = Setpoint
10    condition = Good
11    first_time = true
12    N = 1 // N: Num. of consecutive backoffs
13    action_required = true
14  else if first_time then
15    condition = Normal
16    next_time = now + Interval
17    first_time = false
18  else if now > next_time then
19    condition = Bad
20    next_time = now +  $\frac{\text{Interval}}{\sqrt{N}}$ 
21    N ++
22    action_required = true
23  if action_required then
24    Action_Enforcer(condition, rtt, Setpoint)
  
```



### 3.2.2 动作执行器

当在源位置检测到Bad条件时，Action Enforcer会覆盖基于丢失的TCP的决策，并将Cwnd设置为1(类似于基于丢失的TCP中的超时)。

基于丢失的TCP：每收到一个ACK分组就将Cwnd提高  $1/Cwnd$ ，没有考虑到在网络状态良好的情况下可以多发点  
C2TCP：多加一个Setpoint /RTT\_current

正常情况没有变化

#### Algorithm 2 Action Enforcer's Algorithm

```
1 Function Action_Enforcer(condition,rtt,Setpoint) //  
2   switch condition do  
3     case Good do  
4       Cwnd +=  $\frac{Setpoint}{rtt} \times \frac{1}{Cwnd}$   
5     case Normal do  
6       /* Do nothing! */  
7     case Bad do  
8       /* setting ssthresh using default TCP  
          function which normally recalculates  
          it in congestion avoidance phase */  
9       ssthresh ← recalc_ssthresh()  
10      Cwnd ← 1
```



### 3.2.3 调谐器

#### Algorithm 3: Tuner's Algorithm

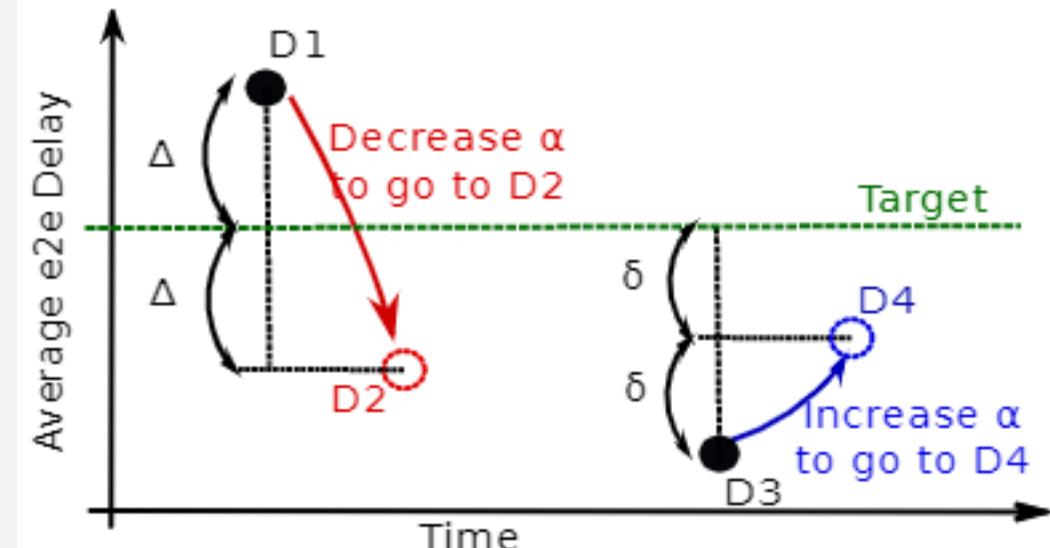
```

1 Function Tune()
  /* Every 0.5 second tune α using following:
   */
2   avg_rtt ← average rtt during previous Tuning Cycle
3   min_α ← 1
4   max_α ← 10
5   if avg_rtt < Target then
6     α +=  $\frac{\text{Target}-\text{avg\_rtt}}{2\text{avg\_rtt}}$ 
7     if max_α ≤ α then
8       α = max_α
9   else if Target < avg_rtt then
10    α -=  $\frac{2(\text{avg\_rtt}-\text{Target})}{\text{Target}}$ 
11    if α ≤ min_α then
12      α = min_α

```

调优器周期性地利用包的平均延迟的统计数据，并利用应用程序给出的目标来调整条件检测器块的 $\alpha$ 参数。设计的直觉：条件检测器的灵敏度与 $\alpha$ 值的大小成反比

半秒的调优周期较佳， $\alpha$ 的变化量与平均端到端延迟到目标的距离成正比。





### 3.3 具体分析

定义：

e2eDelay：发送数据包的时间(发送方)和接收到相应的Ack数据包的时间之间的延迟

bw：移动时间窗口的大小

P：填满管道而不引起任何排队延迟的包数

MINRTT：全局最小RTT

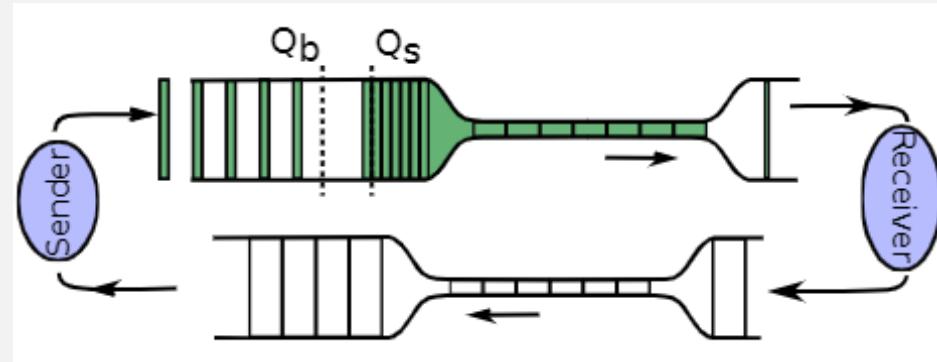
Ws：导致 $e2eDelay=Setpoint$ 的排队延迟 ( $Ws=Setpoint-MINRTT$ )

Qs：排队延迟为Ws时对应的队列长度

Inflight：网络中包的个数



### 3.3 具体分析



模型：队列，水平方向为时间，垂直方向为带宽，绿色矩形表示数据包，面积不变。

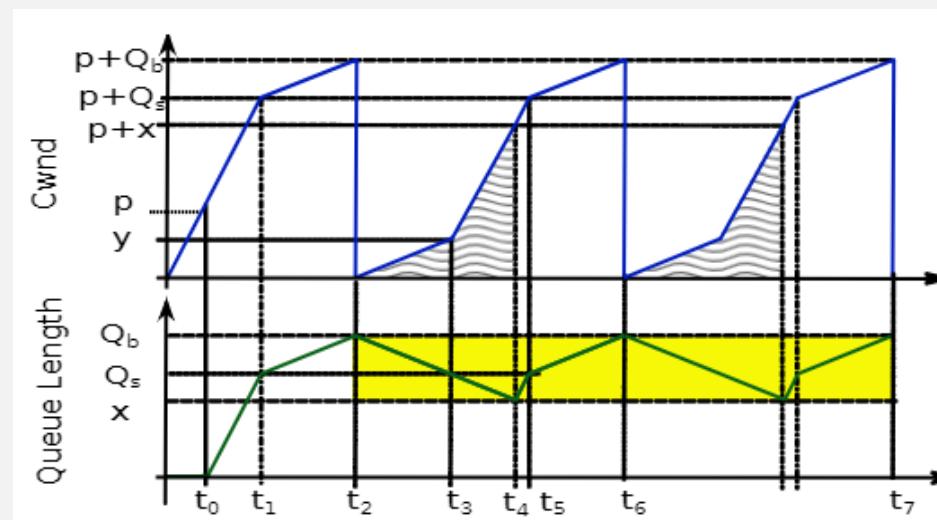
[0,  $t_0$ )  $Cwnd < P$ , 无排队延迟, 好坏情况

[ $t_0, t_1$ ) 管道尺寸与Cwnd不匹配, 队列长度开始增加

[ $t_1, t_2$ ) 排队延迟大于Ws和e2eDelay > 设定点, 正常情况

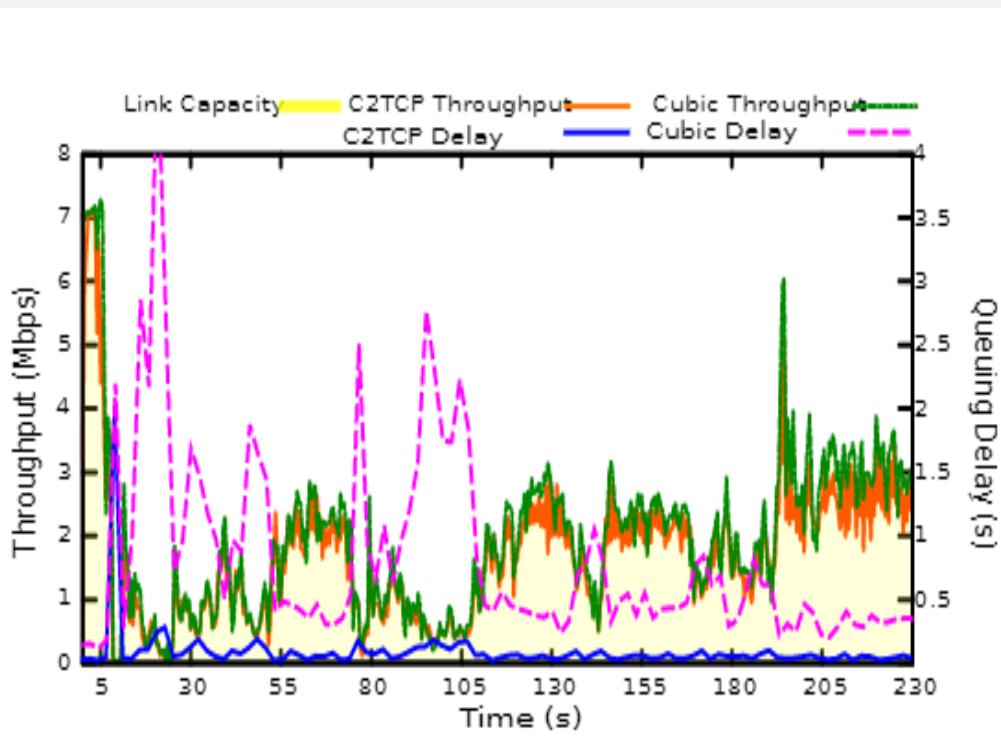
[ $t_2, t_3$ )  $\text{minRTT} > \text{Setpoint}$ , 坏情况。因为发送方只允许向网络发送( $Cwnd - \text{Inflight}$ )数据包, 此时 $CWnd=1 < \text{Inflight}$ , 不发送包, 队列长度减小

[ $t_3, t_4$ ) 队列长度变为 $Q_s$ ,  $e2eDelay = \text{设定点}$ , 在 $t_4$ ,  $Cwnd = \text{Inflight}$ , C2TCP开始发送新包, 队列长度增加。





### 3.4 可行原因



- 1 避免过多的分组发送：保持适量的分组，避免了队列的积累和分组延迟增加，实现蜂窝接入链路的高利用率。
- 2 吸收信道的动态：一般网络中的不同延迟源的动态，而不需要知道这些延迟的确切来源
- 3 蜂窝链路作为瓶颈：新趋势将内容推向终端用户，有助于C2TCP的设计专注于最后一英里的延迟性能
- 4 蜂窝网络中用户队列之间的隔离：不同的UE(用户设备)在BS处获得各自独立的深层队列，使BS的调度器负责使用不同的算法
- 5 C2TCP与其他流共享一个队列，具有良好的公平性
- 6 识别坏条件的思想帮助C2TCP在一个时间窗口中检测到持续的延迟问题，而不是直接对每个大型RTT做出反应。



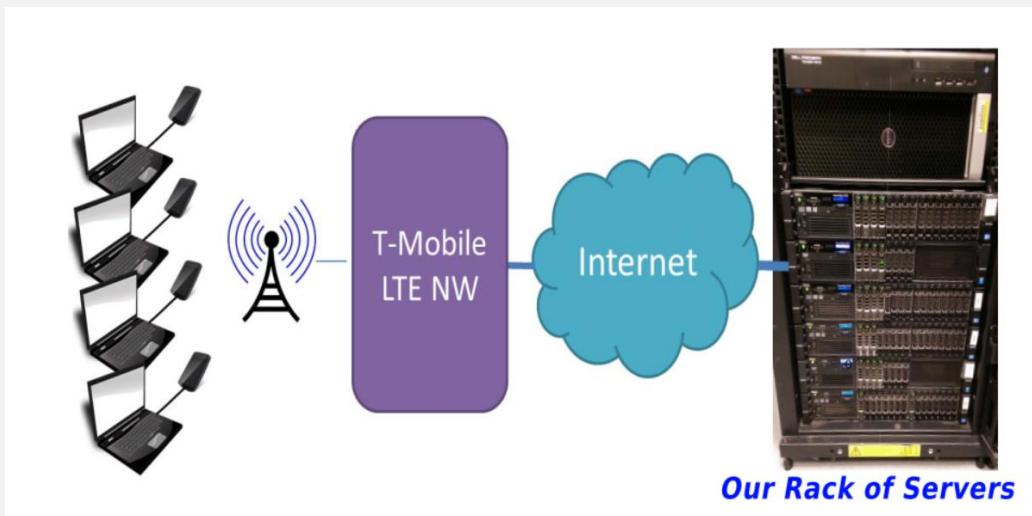
中山大學  
SUN YAT-SEN UNIVERSITY

Part.04

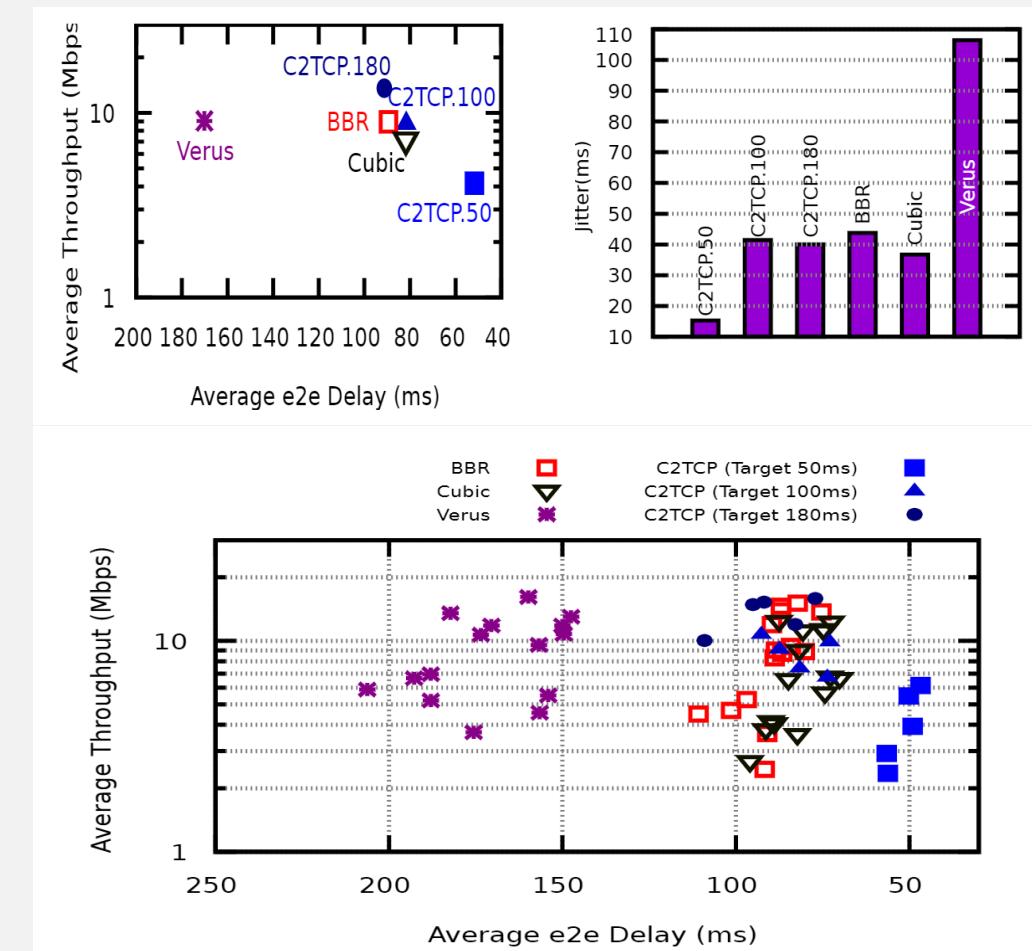
## 评价与分析



## 4.1 真实环境评估

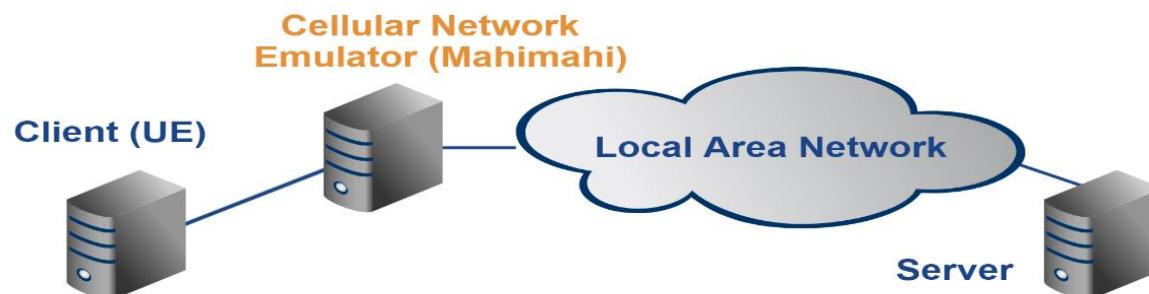


按照上图的拓扑结构进行多次实验；  
结果显示无论是延迟还是抖动，C2TCP都可以很好地控制基于目标数据包的平均性能。  
小的Target下能实现超低延迟与抖动，  
增加Target实现了更高吞吐量，性能与Cubic相当。



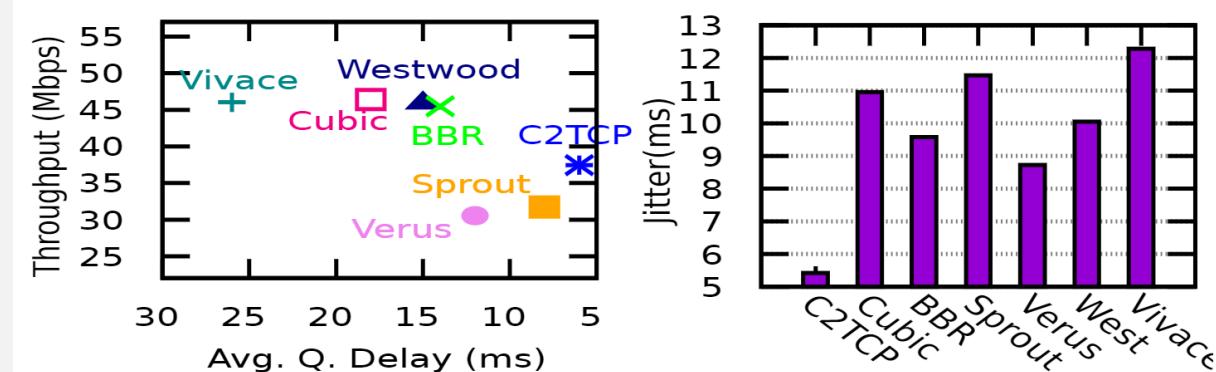


## 4.2 网络仿真评估



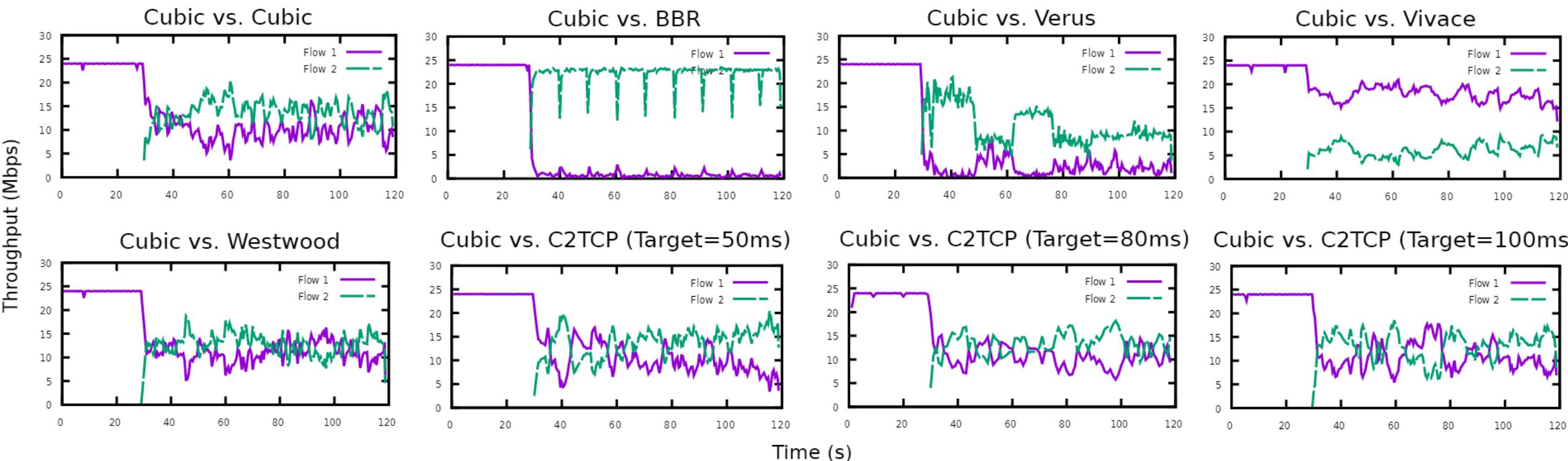
按照上图的拓扑结构进行多次实验；  
我们将不同方案的结果归一化为C2TCP的性能，并在所有评估中取其平均值。  
在所有方案中，C2TCP实现了最低的平均延迟、最低的抖动和最低的95%延迟，同时略微降低了吞吐量。与达到最高吞吐量的Cubic相比，C2TCP减少了大约9倍的平均延迟，而它只降低了0.28倍的吞吐量。

	Throughput	Avg. Delay	Jitter	95th%tile Delay
C2TCP	1	1	1	1
BBR	1.22	2.44	2.15	2.31
Verus	1.12	3.82	9.25	3.22
Cubic	1.28	8.95	7.19	8.54
Sprout	1.13	1.94	1.32	1.64
Westwood	1.26	6.89	6.04	6.78
PCC-Vivace	1.04	10.05	9.25	10.52





### 4.3.1 TCP友好性



TCP友好性是指在有其他TCP变体存在的情况下，竞争流之间共享带宽的公平程度。实验显示BBR过于激进，PCC-Vivace过于温和，这些都是不可取的；而C2TCP对其他TCP流，包括其本身都实现了良好的公平性。

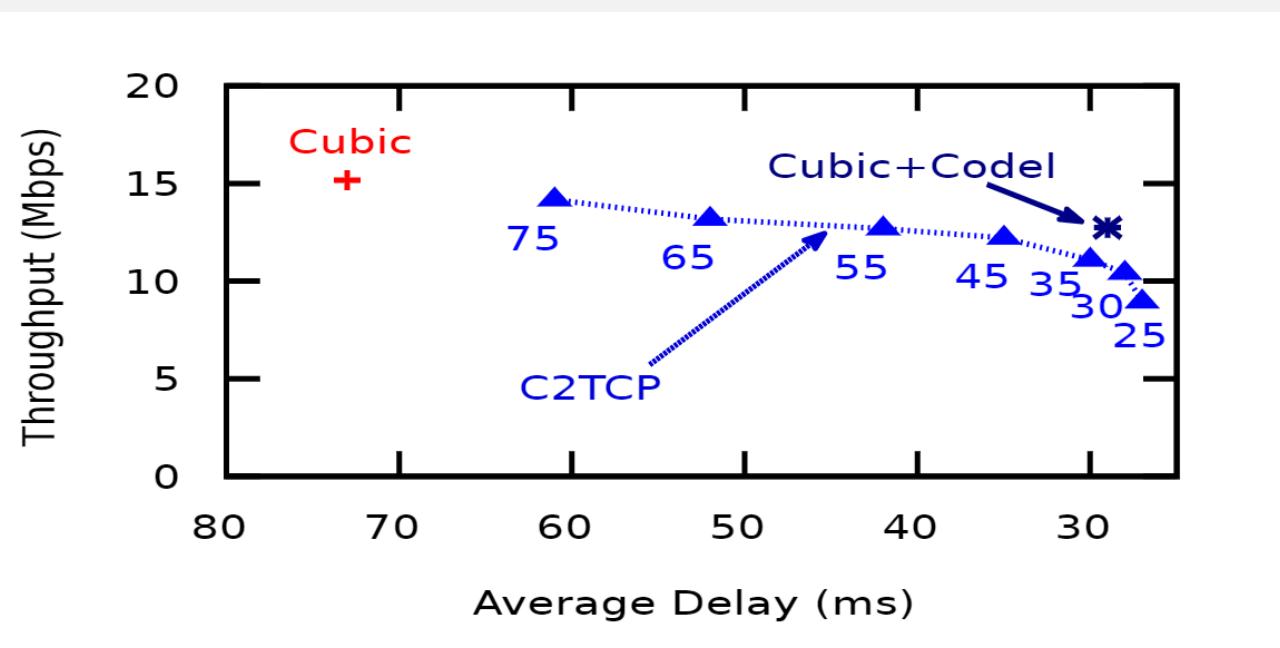


### 4.3.2 与CoDel对比

使用CoDel可以提高Cubic的延迟性能，同时降低其吞吐量。

但是，要拥有像CoDel这样的网络内解决方案，移动运营商必须将它们安装在他们的基站内，以及蜂窝电话上的基带调制解调器或无线电接口驱动程序中，而像C2TCP这样的终端主机解决方案只需要更新服务端的软件。

此外，终端主机解决方案为应用程序提供一定程度的自由来控制它们的操作点，而不是让网络内解决方案缺乏适应各种应用程序需求的灵活性，需要修改网关和网络设备。



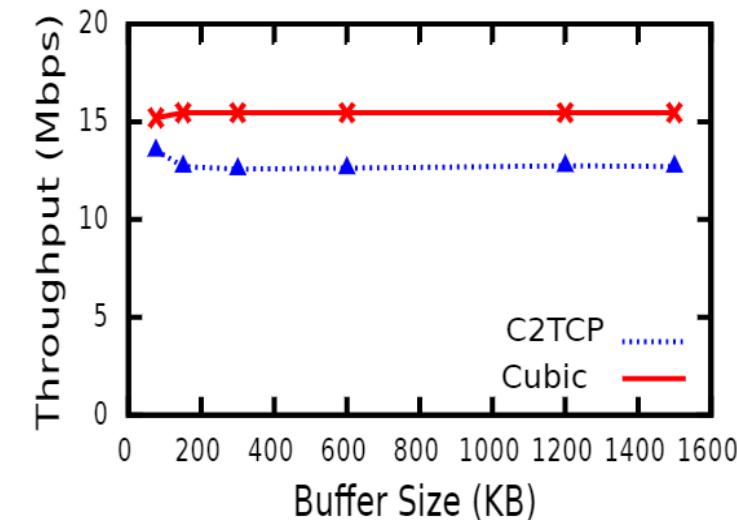
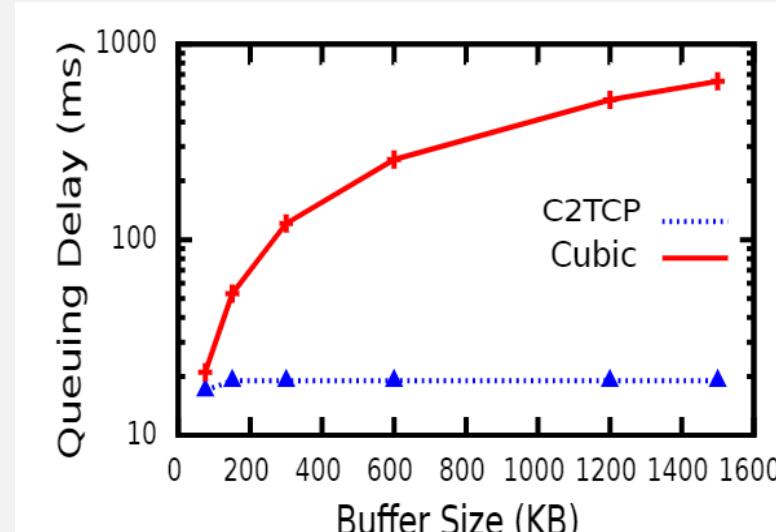


### 4.3.3 缓冲区的影响

缓冲区的这种大小可能导致TCP拥塞控制算法失效（缓冲膨胀）。

C2TCP的主要设计目标是控制数据包的延迟。它的一个主要特征和重要特性是它对队列大小的敏感度很低。

C2TCP总是尝试向网络发送适当数量的包(这几乎与缓冲区大小无关)，以控制它们的延迟。因此，瓶颈缓冲区大小的变化不会影响其吞吐量性能。





中山大學  
SUN YAT-SEN UNIVERSITY

Part.05

## 总结与思考



## 5.1 亮点与不足

亮点之处

- 动态适应不同应用对时延的需求
- 只在服务端修改，部署成本低
- 低时延同时保证满足正常吞吐量需求

不足之处

- 仅适用于蜂窝网络
- 存在用户滥用参数风险





## 5.2 总结

**论文结论：**C2TCP，是一个为蜂窝网络设计的拥堵控制协议，用于控制数据包的延迟并实现高吞吐量。

- 01 不使用任何网络状态分析、通道预测或复杂的速率调整机制
- 02 不改变网络设备，只修改服务端以实现超低延迟和高带宽的通信
- 03 实际实验和跟踪评估表明，C2TCP优于著名的TCP变体和现有的使用信道预测或网络延迟分析的最先进方案。



## 5.3 参考文献

- ◆ [1] Abbasloo S , Xu Y , Chao H J . C2TCP: A Flexible Cellular TCP to Meet Stringent Delay Requirements[J]. IEEE Journal on Selected Areas in Communications, 2019, PP(4):1-1.
- ◆ [2] Abbasloo S , Li T , Xu Y , et al. Cellular Controlled Delay TCP (C2TCP)[J]. IEEE, 2019.



中山大學  
SUN YAT-SEN UNIVERSITY



谢谢各位的聆听

中山大学智能工程学院

A photograph of the traditional stone archway at Sun Yat-Sen University. The archway is ornate with carvings and inscriptions. In the background, modern buildings are visible against a clear sky.

小组成员：方桂安，陈石翰，陈金华



指导老师：古博