面向无人机视频传输的增强传输技术研究与实现

(申请西南交通大学工学硕士学位论文答辩报告)

学　　生：董　　泽　　锋

指导教师：陈 庆 春 教授

信息编码与传输重点实验室

二〇一七年五月

目录

目录

无人机行业发展

近几年国内无人机市场规模飞速发展，兴起了一批如大疆科技、零度智控等科技企业。

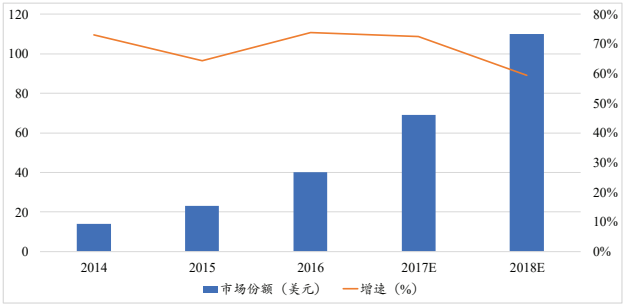
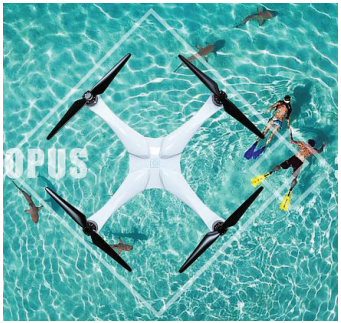


图 1: 中国无人机市场规模

无人机航拍

无人机一项重要应用是航拍。航拍所采用的数据传输方案很大程度上影响用户的体验。



视频传输方案

(1) 专有数据链，如 DJI 的 Lightbridge

(2) 基于 TCP/IP 的网络视频传输

两种传输方案都有各自的优缺点。

两种视频传输方案优缺点

专有数据链

优点

传输时延小、单向广播

缺点

需要专门设备，价格较高，且与目前因特网体系架构不兼容

基于 TCP/IP 的方案

优点

与现有网络体系架构兼容，符合未来万物互联的趋势，设备价格低廉。

缺点

传输时延大，容易受到无线链路中各种干扰，受限于 TCP 协议的缺点。

目录

TCP 简单概述

TCP 协议

TCP 是一种可靠的面向连接的协议。为了保证其可靠性，设计了 ACK 确认机制，流控机制，拥塞控制算法。

ACK 机制 对于发送端发出的报文，如果接收端收到，那么会回复一个 ACK 报文进行累积确认。

流控机制 为了让发送端的发送速度和接收端的接收速度匹配，接收端通过 ACK 报文中的接收窗口字段告知发送端自己的处理能力，以让发送端调节好发送速度。

拥塞控制算法 如慢启动、拥塞避免、快速重传和快速回复等，目的是当网络出现拥塞的时候，减慢发送速度，以避免网络进一步拥塞；当网络通畅的时候，增大发送速率，以最大化利用网络带宽。

使用 TCP 进行无人机视频传输存在的问题

TCP 协议最初是针对有线网络设计的，考虑到有线网络中极低的误码率，其拥塞控制模型以丢包为基础。但无人机视频传输有其自身特点

无人机视频传输环境特点

大尺度衰落信道环境、多径干扰导致的高误码率；障碍物遮挡、基站切换导致的连接短暂中断；视频传输对传输时延很敏感。

标准 TCP 在面对这些特点时，表现出了天生的劣势。

劣势

(1) 无法区分拥塞丢包和非拥塞丢包，频繁启动拥塞控制算法，降低发送速率。

(2) TCP 的重传确保了可靠，但一定程度上增大了传输时延，而视频传输对时延很敏感。

如何克服这些缺点？ I

在处理拥塞丢包和非拥塞丢包问题上，有两种思想：掩盖非拥塞丢包和区分两种丢包。

掩盖非拥塞丢包

ARQ	应用于无线网络的链路层，但会和 TCP 的原有机制冲突，导致乱序。
FEC	将前向纠错码和 TCP 结合，如使用异或运算对前 n 个报文编码。
Indirect-TCP	在有损信道入口处终止原始 TCP 连接，由 TCP-agent 接管报文。
TCP-snoop	保持端到端语义，保存经过的数据的拷贝，处理重复 ACK。

如何克服这些缺点？ II

区分丢包

ELN	接收端的 MAC 层可以检测出错，将这一信息告知上层 TCP，上层 TCP 会发送给对方一个报文告知出错。
TCP-vegas	让拥塞检测和丢包解耦。但其同样无法处理丢包引起的重传导致的时延过大问题。
ECN	显示拥塞控制标志位。利用 IP 首部的 TOS 字段。

Sundararajan 等人提出 TCP/NC[1]，在 OSI 协议栈的 TCP 层和 IP 层添加一个网络编码层。在网络编码层对数据包进行冗余编码，掩盖链路中出现的丢包。

[1] J. K. Sundararajan et al. "Network Coding Meets TCP". . In: *Proc. IEEE INFOCOM 2009*. Apr. 2009, pp. 280–288. DOI: 10.1109/INFCOM.2009.5061931.

目录

网络编码概念

R.Ahlswede 等人在论文“Network Information Flow[2]”首次提出网络编码概念。网络编码建立在一个简单而广泛的概念的基础上：

网络编码

在包交换网络中，中间节点不仅仅是简单地路由转发接收到的数据包，而是对它们进行一些函数操作并计算、转发操作结果。

运用网络编码可以提高网络吞吐率、均衡网络负载和提高网络带宽利用率。
我们以简单的蝶形网络为例，说明 Network Coding 的基本原理。

[2] R. Ahlswede et al. “Network information flow”. In: *IEEE Transactions on Information Theory* 46.4 (July 2000), pp. 1204–1216. ISSN: 0018-9448. DOI: 10.1109/18.850663.

网络编码的一个简单例子

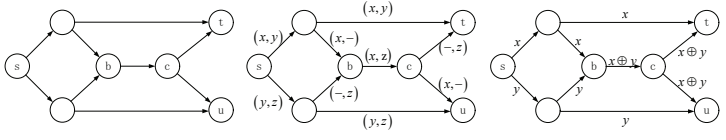


图 2: 蝶形网络

图2(b) 为传统的路由算法，其多播吞吐量为 1.5。图2(c) 为应用了网络编码的路由算法，其多播吞吐量为 2，且是理论上最大的网络多播吞吐量。

定理

一个网络的最大多播吞吐量取决于分割源节点和目的节点的最小割集[2]。

线性网络编码 I

令 $(p_1, p_2, \ldots, p_r)^T$ 表示进入源节点 s 的数据包 $\mathrm{X}_{\mathrm{I}(s)}$。每个数据包 p_i 都是 $\mathbb{F}_q$ 上长度为 m 的矢量。我们可以用一个定义在 $\mathbb{F}_q$ 上的 $r \times m$ 矩阵来表示 $\mathrm{X}_{\mathrm{I}(s)}$，该矩阵的第 i 行是 p_i。

本地编码函数是 $\mathbb{F}_q$ 上的线性函数，即任意中间节点 v 输出的数据包列向量 $\mathrm{X}_{\mathrm{O}(v)}$ 与其接收到的数据包列向量 $\mathrm{X}_{\mathrm{I}(v)}$ 之间的关系可以用以下线性方程组表示：

$$\mathrm{X}_{\mathrm{O}(v)} = \mathrm{L}_v \mathrm{X}_{\mathrm{I}_{(v)}} \tag{1}$$

其中，L_v 是定义在 $\mathbb{F}_q$ 上的系数矩阵。考虑到网络中只允许线性操作，因此任意边上传输的数据包都是原数据包 $p_1, p_2, \ldots, p_r$ 的线性组合。

线性网络编码 II

也就是 $\forall v \in V$，有

$$\mathrm{X}_{\mathrm{I}(v)} = G_v \begin{bmatrix} p_1 \\ \vdots \\ p_r \end{bmatrix} \tag{2}$$

其中，G_v 是定义在 $\mathbb{F}_q$ 上的系数矩阵，也称为 v 的全局转移矩阵。G_v 的每一行都对应一条边 $e \in \mathrm{I}_{(v)}$，称为边 e 的全局编码向量。图3为标示了本地转移矩阵的蝶形网络，我们可以得到在目的节点 t 和 u 的全局转移矩阵为：

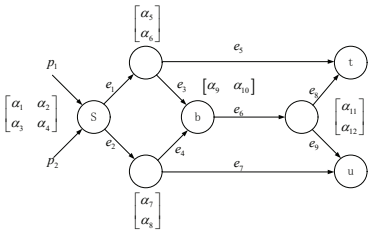


图 3: 标注了转移矩阵的蝶形网络

线性网络编码 III

$$G_t = \begin{bmatrix} \alpha_1\alpha_5 & \alpha_2\alpha_5 \\ \alpha_1\alpha_6\alpha_9\alpha_{11} + \alpha_3\alpha_7\alpha_{10}\alpha_{11} & \alpha_2\alpha_6\alpha_9\alpha_{11} + \alpha_4\alpha_7\alpha_{10}\alpha_{11} \end{bmatrix}$$

$$G_u = \begin{bmatrix} \alpha_1\alpha_6\alpha_9\alpha_{12} + \alpha_3\alpha_7\alpha_{10}\alpha_{12} & \alpha_2\alpha_6\alpha_9\alpha_{12} + \alpha_4\alpha_7\alpha_{10}\alpha_{12} \\ \alpha_3\alpha_8 & \alpha_4\alpha_8 \end{bmatrix} \tag{3}$$

对于目的节点 t 来说，当且仅当 $|\mathrm{I}(t)| \times r$ 阶全局转移矩阵 G_t 的秩为 r 时，t 才能恢复出 $(p_1, p_2, \ldots, p_r)$ 中的所有元素。因为只有 G_t 满秩时，才存在满足 ${G_t}^{-1} G_t = \mathrm{I}_r$ 的左逆矩阵，其中 I_r 是 $r \times r$ 阶单位矩阵。只需 G_t 满秩，而不要求 G_t 是可求逆的方阵。目的节点 t 可以通过计算 ${G_t}^{-1}\mathrm{X}_{\mathrm{I}(t)}$ 得到 $(p_1, p_2, \ldots, p_r)^T$。

Batch Coding 和 Pipeline Coding I

将网络编码应用于提升有损链路的吞吐率，其应用方式主要有两种：Batch Coding 和 Pipeline Coding。

Batch Coding

定义 *generation* 为数据包的集合，作为编解码的整体。源节点和网络中的中间节点在同一个 *generation* 上进行编码，目的节点只有在收到足够多的编码报文后，才能一次性解出这一个 *generation* 中的所有原始报文。

假设 $p_1, p_2, \ldots$ 为原始数据包，k 为 *generation* 的大小，第 i^{th} 个 *generation* 产生的一个编码包可以被表示为：

$$c = \sum_{j=1}^{k} e_j p_{i \times k + j} \tag{4}$$

Batch Coding 和 Pipeline Coding II

Pipeline Coding

无需等待一个 *generation* 满秩，每当一个数据报文来之后，就会产生一个编码包。目的节点会逐步解出原始报文。

Pipeline Coding 的编码函数为：

$$c = \sum_{j=1}^{m} e_j p_{i \times k + j} \tag{5}$$

其中 m 表示目前在编码缓存中数据包的个数。图4为两种编码方式示例，其中冗余度都为 1.25，即每发送 4 个报文发送一个冗余包。

Batch Coding 和 Pipeline Coding III

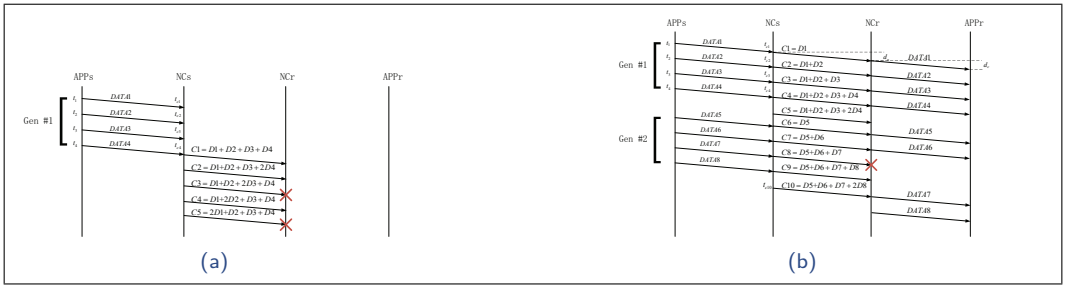


图 4: (a) Batch Coding (b) Pipeline Coding

采用 Pipeline Coding 的优点：解码时延更低。

目录

ACK On degree of freedom I

TCP、ARQ 协议中的 ACK 都是对原始数据包的确认，*ACK On degree of freedom*[3]则对自由度进行确认。

定义 (See a packet)

如果一个节点根据现有的信息可以计算出如$(p+q)$ 形式的线性组合，那么我们就说这个节点“***see packet*** p”。其中q 本身就是只包含序号比 p 大的报文的线性组合。解码出某个报文也算作是“***see a packet***”，此时$q=0$。

接收端如果 *see packet* p，那么就需要对 p 进行 ACK。表1为 *ACK on degree of freedom* 的例子。

ACK On degree of freedom II

表 1: *ACK on degree of freedom* 例子

Time	Sender's queue	Transmitted packet	Channel state	Destination Node A	
				Decoded	Seen but not decoded
1	p_1	p_1	$\nrightarrow A$		
2	p_1, p_2	$p_1 \oplus p_2$	$\rightarrow A$		p_1
3	p_2, p_3	$p_2 \oplus p_3$	$\rightarrow A$		p_1, p_2
4	p_3, p_4	$p_3 \oplus p_4$	$\nrightarrow A$		p_1, p_2
5	p_3, p_4, p_5	$p_3 \oplus p_4 \oplus p_5$	$\rightarrow A$		p_1, p_2, p_3
6	p_4	p_4	$\rightarrow A$	p_4	p_1, p_2, p_3, p_5
7	p_5	p_5	$\rightarrow A$	p_1, p_2, p_3, p_4, p_5	

优点：发送端可以减小发送队列的长度。

[3] J. K. Sundararajan, D. Shah, and M. Medard. "ARQ for network coding". In: *2008 IEEE International Symposium on Information Theory*. 2008, pp. 1651–1655. DOI: 10.1109/ISIT.2008.4595268.

TCP/NC 概述

为了提高标准 TCP 在有损信道环境下的性能，Sundararajan 等人提出了 TCP/NC。将网络编码和 TCP 结合起来，在发送端冗余编码，掩盖链路中出现的丢包。
其关键点有：

TCP/NC 关键点

1. 在发送端的 TCP 层和 NC 层插入一个网络编码层，在编码层对数据包进行线性网络编码。
2. 利用 *ACK On degree of freedom* 的概念，重新定义 TCP 中 ACK 的概念。
3. 发送端通过冗余编码来补偿链路中出现的丢包

TCP/NC 框图

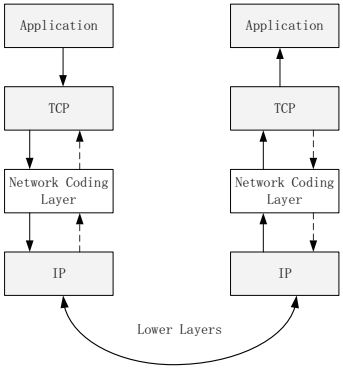


图 5: TCP/NC 结构框图

NC 层发送端

1. 接收 TCP 层的数据包，根据编码窗口和冗余因子对数据包进行线性组合，然后发往 IP 层。
2. 处理接收端回复的 ACK 报文，并据此删除编码队列的报文。

NC 层接收端

1. 接收 IP 层传上来的编码包，并进行解码运算。如果有新的报文被看到，回复发送端 ACK 报文；如果有新的原始数据包被解码，将其传给上层 TCP。
2. 拦截上层 TCP 下来的 ACK 报文。

编解码示例

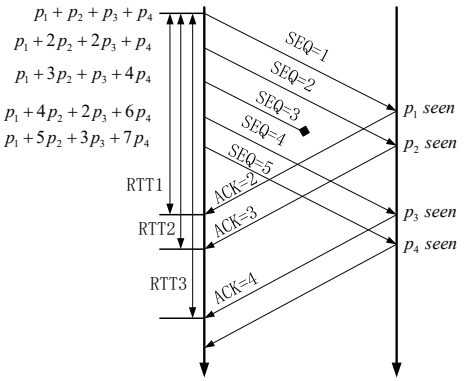


图 6: 编码发送示例

图6所示为冗余度 1.25 时编码发送的一个示例。

尽管 $SEQ = 3$ 丢失，但是接收端在收到 $SEQ = 4$ 的编码包后，看到了报文 p_3，所以回复给发送端的是 $ACK = 4$。在收到冗余报文 $SEQ = 5$ 后，看到了 p_4，同时也将 $p_1 \sim p_4$ 全部解出。

链路的丢包在发送端的 TCP 层看来仅仅是 RTT 的增大。

冗余度

TCP/NC 能够抵抗链路丢包的关键是发送端进行了冗余编码。冗余度决定了 TCP/NC 发送冗余包的多少。
理论上，当链路的丢包率为 ρ 时，设置的冗余度 R 应该为

$$R = \frac{1}{1-\rho} \tag{6}$$

此时，链路中出现的丢包刚好全部被冗余包补偿。

目录

有限域

有限域

有限域提供了一个有限集，在该有限集上明确定义且有效地实现了加法、减法、乘法和除法运算，并允许系统使用矩阵、行列式、高斯消元等线性代数中常见的运算工具来解决该域上的联立线性方程组问题。

+	0	1	A	B
0	0	1	A	B
1	1	0	A	B
A	A	B	0	1
B	B	A	1	0

.	0	1	A	B
0	0	0	0	0
1	0	1	A	B
A	0	A	B	1
B	0	B	1	A

编码示例

我们以两个数据包 p_1 和 p_2 的运算为例，说明如何对数据包进行编码。假定数据包 p_1 和 p_2 都为一个 2 字节的报文，以比特流的形式表示，
$p_1 = \{1010\ 0101\ 0110\ 1111\}$，$p_2 = \{1111\ 0000\ 1101\ 0110\}$，我们需要计算 $p_{encoded} = 7p_1 \oplus 13p_2$ 的值。步骤如下：

(1) 取 p_1 和 p_2 的第一个字节，分别是 $\{1010\ 0101\}$ 和 $\{1111\ 0000\}$，对应的十进制为 $D_{p_1} = 165$ 和 $D_{p_2} = 240$。在有限域 $GF(2^8)$ 下计算 7×165 和 13×240 的值，分别是 $0xc6$ 和 $0xc4$。则 $p_{encoded}$ 的第一个字节为 $0xc4 \oplus 0xc6 = 0x02$，二进制表示为 $\{0000\ 0010\}$。

(2) 同样的方法处理 p_1 和 p_2 的第二个字节，结果为 $\{1010\ 0111\}$。

(3) 拼接两个结果得到 $p_{encoded} = \{0000\ 0010\ 1010\ 0111\}$。

目录

设备选型

考虑到无人机视频传输的应用场景，及 TCP/NC 需要参与到网络协议栈的流程中去，选用 Raspberry Pi 这款基于 Linux 的嵌入式设备，部署 TCP/NC 协议, 如图7所示。

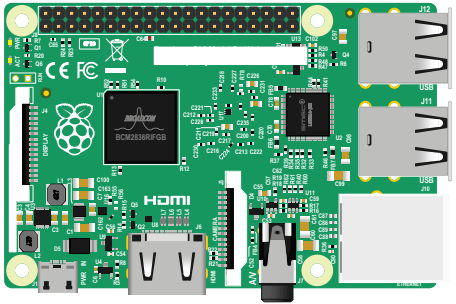


图 7: Raspberry Pi 板子

Netfilter

NC 层实施方案有两种：

1. 直接在内核的 TCP 协议源码上更改，添加 NC 层的各个模块，然后再编译内核源码
2. 利用 Linux 内核提供给用户用于处理网络封包的框架——Netfilter。

本文将 TCP/NC 部署在 Netfilter 的 NF_IP_LOCAL_IN 和 NF_IP_LOCAL_OUT 处。

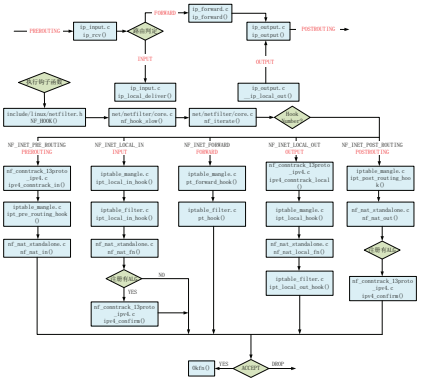


图 8: Netfilter 及其在内核源码中调用关系

关键数据结构

NC 层在 NF_IP_LOCAL_IN 和 NF_IP_LOCAL_OUT 这两个钩子点获得的都是单独的报文，其形式为 *sk_buff* 结构体，如图9所示。

数据包在 Linux 的网络协议栈的不同层之间传递，其体现形式就是不同层的函数对一个数据包对应的 *sk_buff* 进行各种操作。

TCP/NC 通过 *sk_buff* 获得数据包的字节流，然后对数据包进行编码。

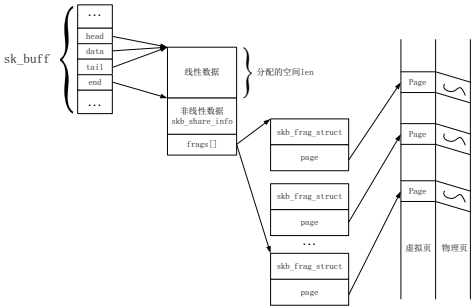


图 9: skbuff 结构

目录

NC 层头部

发送端需要给编码得到的编码包添加 NC 头部，包括编码系数等信息，接收端利用这些信息进行解码。图4为 NC 报文的头部。

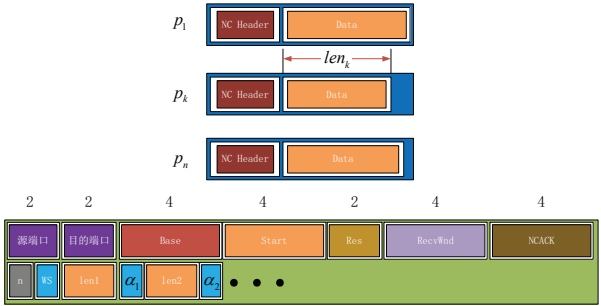


图 10: NC 层头部

编码

发送端的 NC 层维护了一个 *sk_buff* 的链表 *sk_buff_head*，存放从 TCP 层下来的数据包。此链表也作为发送端的编码缓存，其插入操作发生于有数据包文从 TCP 层下来时；删除操作发生于收到接收端的 NC 层的 ACK 报文时。编解码步骤如右图。

算法 3: 对数据包进行编码

输入:
冗余度因子 R;
编码缓存双向链表 sk_buff_head;
冗余度残留系数 NUM;
编码窗口 $TCPNC\_CODE\_WND$;

输出:
编码报文链表 sk_buff_ret

```
1 NUM ← NUM + R;
2 while NUM ≥ 1 do
3     从 sk_buff_head 表头开始往后遍历 TCPNC_CODE_WND 个报文;
4     在 GF(2^8) 上生成 TCPNC_CODE_WND 个系数;
5     根据系数和原始数据包计算出编码包;
6     给编码报文添加 NC 头部;
7     将编码包加入到 sk_buff_ret 链表;
8     NUM ← NUM − 1;
9 return sk_buff_ret;
```

解码

接收窗口

考虑到 NC 层提前确认了一些数据包，在接收端，我们需要对发出去的 ACK 报文的接收窗口值作出修改。

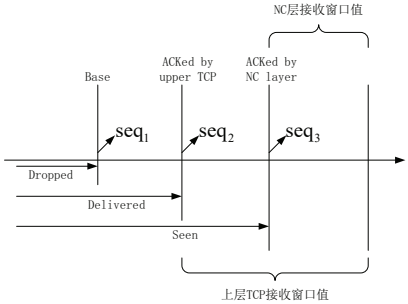


图 11: 接收窗口设定

目录

测试 I

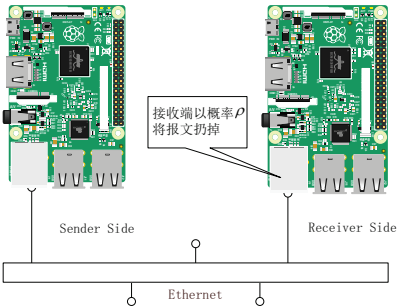


图 12: 测试拓扑

表 2: 测试环境参数

参数名称	参数值
机型	Raspberry Pi 2 Model B
CPU	900MHz Cortex-A7
RAM	1G
以太网卡	10M/100M
操作系统	Debian
内核版本	Linux 3.18
TCP 协议版本	TCP-Reno
测试工具	iperf
测试时间	60 秒

测试 II

固定链路丢包率为 5%, 测试冗余度对吞吐率的影响如图**??**。我们对不同丢包率下 TCP 和 TCP/NC 的性能进行了对比测试，如图**??**所示，其中 TCP/NC 的吞吐率都为最优吞吐率。

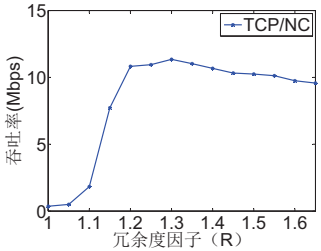


图 13: 吞吐率-冗余度

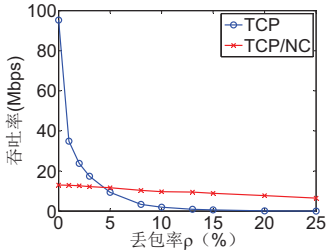


图 14: 吞吐率-丢包率

结论

目录

E-TCP/NC 头部

为了支撑 E-TCP/NC 的新特性，设计了新的头部。

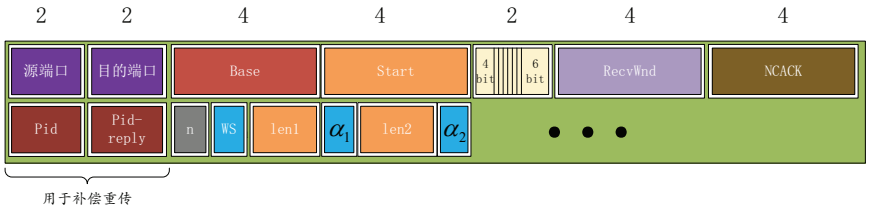
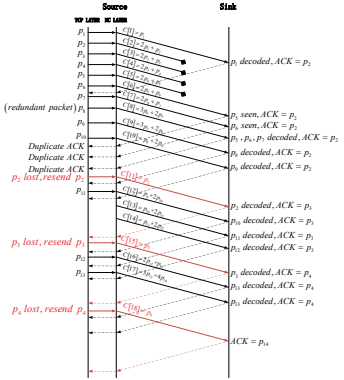


图 15: 新 NC 头部设计

添加了 *Pid* 和 *Pid-reply*。*Pid* 表示编码报文的编号；如果此报文为 ACK 报文的话，*Pid-reply* 表示这个 ACK 报文是由编号为 *Pid-reply* 的报文激发的。

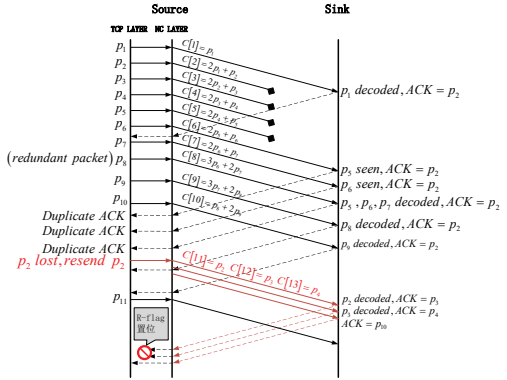
TCP/NC 的突发丢包问题

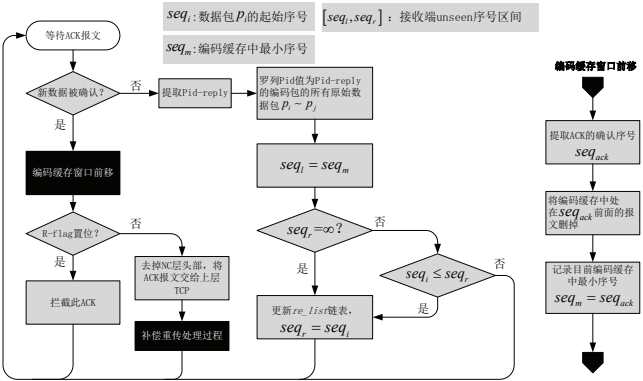
TCP/NC 将重传的任务全部交给上层 TCP，而 TCP-Reno 无法很好地处理一个 RTO 内的连续报文丢失。在未发生超时重传时，TCP 层单个、按序地在多个 RTT 内重传多个丢失的报文。



改进方法

目标：发送端在一个 RTT 内重传所有丢失的报文。
首先需要定位丢包，确定丢了哪些报文。




seq_l：数据包P_i的起始序号
$[seq_l, seq_r]$：接收端unseen序号区间
seq_m：编码缓存中最小序号
等待ACK报文
新数据被确认？
否
是
提取Pid-reply
罗列Pid值为Pid-reply的编码包的所有原始数据包$P_i \sim P_j$
$seq_l = seq_m$
编码缓存窗口前移
R-flag置位？
否
是
去掉NC层头部，将ACK报文交给上层TCP
拦截此ACK
补偿重传处理过程
seq_r=∞？
否
是
$seq_l \le seq_r$
否
是
更新re_list链表，
$seq_r = seq_l$
编码缓存窗口前移
提取ACK的确认序号
seq_{ack}
将编码缓存中处在seq_{ack}前面的报文删掉
记录目前编码缓存中最小序号
$seq_m = seq_{ack}$

目录

自适应冗余度算法

TCP/NC 中冗余度 R 的设置很关键。现实中的网络环境，尤其是无线环境，其丢包率是在变动的。在设置冗余因子时，如果 R 过小，那么冗余信息不足以掩盖链路中的丢包；如果 R 值过大，编码码率就过小，降低网络的有效吞吐率。因此需要找出一种能够根据网络状况自适应调整冗余度的方法，最大化地利用网络带宽。

利用新 NC 头部的 *Pid* 和 *Pid-reply* 字段，发送端可以精确知道网络中丢包情况，即丢包率 ρ。
更新冗余度方法如下：

$$R = \frac{1}{1-\rho} \tag{7}$$

目录

解码矩阵

$$
\begin{array}{c} \\ c_1 \\ c_2 \\ c_4 \\ c_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 \end{array} \\
\left[\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \frac{1}{4} & 0 & 0 & 0 \\
0 & 0 & 0 & a & b & c & 0
\end{array}\right]
\end{array}
$$

图 16: 解码矩阵示例

解码矩阵

图16表示接收端的解码矩阵的一个示例。编码报文 c_3 丢失，c_4 正确收到，c_5 又丢失。可以看到，接收端可以解码出 p_1 和 p_2，看到（see）p_3 和 p_4，无法解出 $p_3 \sim p_6$。

以图16为例，解码矩阵还需要 2 个由 p_3 到 p_6 组成的编码包即可解码出 p_3 和 p_6。如果能让发送端了解到这个信息，发送端在下一回合发送过程中，主动额外地多发相关的编码包，就可以提前解出 $p_3 \sim p_6$，而无需等到下一次根据冗余度发出的冗余包。

获码矩阵信息

发送端

对于发出去的每个编码包，若其编号为 *Pid*，记录编号为 *Pid* 的编码包由哪几个原始数据包组成，以图6为例，*Pid=1* 的编码包由 p_1、p_2、p_3 和 p_4 组成。

接收端

接收端在发出去的每个 ACK 报文中都会填写 NC 头部的 *Pid-reply* 字段，表示此 ACK 报文是由发送端的编号为 *Pid-reply* 的编码报文激发的。

获取接收端解码矩阵信息

定义 (矩阵可解状态)

假定接收端收到的序号最大的报文为 p_j，已经确认的序号最大的报文为 p_i。如果 $i = j$，那么就说接收端解码矩阵为可解状态。

发送端的处理过程如下：

步骤

1. 提取 *Pid-reply* 域，罗列值为 *Pid-reply* 的编码包的所有原始数据包 $p_i \sim p_j (i \leq j)$；

获取接收端解码矩阵信息

定义 (矩阵可解状态)

假定接收端收到的序号最大的报文为 p_j，已经确认的序号最大的报文为 p_i。如果 $i = j$，那么就说接收端解码矩阵为可解状态。

发送端的处理过程如下：

步骤

1. 提取 *Pid-reply* 域，罗列值为 *Pid-reply* 的编码包的所有原始数据包 $p_i \sim p_j (i \leq j)$；
2. 如果 ACK 报文的确认序号 $ACK = p_k$，那么可知接收端解码矩阵在收到编号为 *Pid=Pid-reply* 的编码报文时，缺失 $j - k + 1$ 个编码包才可以变为可解状态；

获取接收端解码矩阵信息

步骤（接上页）

3 更新 *loss* 和 *last_loss* 变量，其中 *loss* 表示接收端解码矩阵变为可解状态所需的组合包个数，*last_loss* 表示 *loss* 上一次的值。

获取接收端解码矩阵信息

步骤（接上页）

3 更新 *loss* 和 *last_loss* 变量，其中 *loss* 表示接收端解码矩阵变为可解状态所需的组合包个数，*last_loss* 表示 *loss* 上一次的值。

4 计算当前时间 T_{now} 和上一次补偿重传的时间 T_{last} 的差值是否超过 RTO。如果超过，那么重传当前编码缓存窗口的前 *loss* 个报文；如果未超过，比较 *loss* 的 *last_loss* 的值，如果 $loss \leq last\_loss$，那么不进行补偿重传，如果 $loss > last\_loss$，说明在 T_{last} 到 T_{now} 这段时间又有报文丢失，那么就重传编码缓存窗口的前 $loss - last\_loss$ 个报文。

流程图

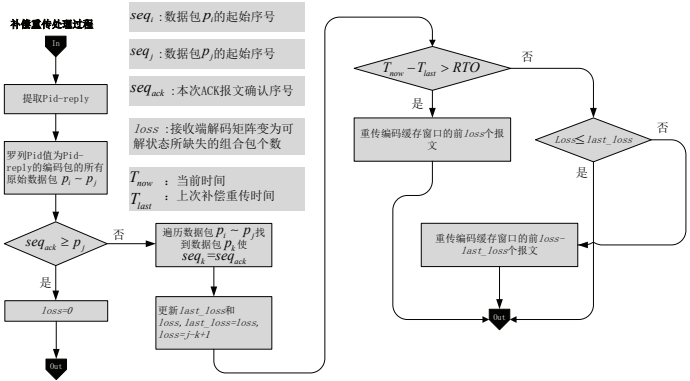


图 17: 补偿重传流程图

目录

主要参考文献 I

[1] J. K. Sundararajan et al. “Network Coding Meets TCP”. In: *Proc. IEEE INFOCOM 2009*. Apr. 2009, pp. 280–288. DOI: 10.1109/INFCOM.2009.5061931.

[2] R. Ahlswede et al. “Network information flow”. In: *IEEE Transactions on Information Theory* 46.4 (July 2000), pp. 1204–1216. ISSN: 0018-9448. DOI: 10.1109/18.850663.

[3] J. K. Sundararajan, D. Shah, and M. Medard. “ARQ for network coding”. In: *2008 IEEE International Symposium on Information Theory*. 2008, pp. 1651–1655. DOI: 10.1109/ISIT.2008.4595268.

致谢

(1) 导师陈庆春老师的精心指导；
(2) 教研室各位同学三年来对我的帮助；
(3) 教研室各个项目的历练；
(4) XX 老师、XX 老师的评审及意见，答辩委员会老师聆听和指导。

Q & A

Questions?

Thank you!