# WebRTC 基于 Kalman Filter 的视频接收缓冲区延迟模型

在WebRTC的视频处理流水线中，接收端缓冲区JitterBuffer是关键的组成部分：它负责 RTP 数据包乱序重排和组帧，RTP 丢包重传，请求重传关键帧，估算缓冲区延迟等功能。其中缓冲区延迟 JitterDelay 对视频流的单向延迟有重要影响，很大程度上决定着应用的实时性。本文不打算全面分析接收端缓冲区的实现细节，只针对缓冲区延迟 JitterDelay 的计算这一议题进行深入分析。

## 1 接收端延迟的组成

WebRTC 视频接收端延迟包括三部分：缓冲区延迟 JitterDelay，解码延迟 DecodeDelay 和渲染延迟 RenderDelay。其中 DecodeDelay 和 RenderDelay 相对比较稳定，而 JitterDelay 受发送端码率和网络状况影响较大。JitterDelay 也是造成接收端延迟的最大因素。

缓冲区延迟由两部分延迟构成：传输大尺寸视频帧造成码率 burst 引起的延迟和网络噪声引起的延迟。WebRTC 采用卡尔曼滤波 Kalman Filter 估算网络传输速率和网络排队延迟，进而确定缓冲区延迟。本文在理论学习卡尔曼滤波的基础上，结合 WebRTC 源代码，详细深入分析缓冲区延迟的计算和更新过程。

## 2 卡尔曼滤波理论学习

本节主要参考引用文献[1][2][3]，以此为入门资料学习**卡尔曼滤波**的基本原理。简单来说，卡尔曼滤波器是一个最优化自回归数据处理算法，文献[2]概括卡尔曼滤波的基本思想：

“本质上来讲，滤波就是一个信号处理与变换(去除或减弱不想要的成分，增强所需成分)的过程。卡尔曼滤波属于一种软件滤波方法，其基本思想是：以最小均方误差为最佳估计准则，采用信号与噪声的状态空间模型，利用前一时刻的估计值和当前时刻的观测值来更新对状态变量的估计，求出当前时刻的估计值。算法根据建立的系统方程和观测方程对需要处理的信号做出满足最小均方误差的估计。”

下面以文献[1]为基础，简要分析卡尔曼滤波的基本过程。

### 2.1 建立系统数学模型

首先，我们先要引入一个离散控制过程的系统。该系统可用一个线性随机微分方程（Linear Stochastic Difference equation）来描述：

$$X(K) = A\,X(K-1) + B\,U(k) + W(k) \qquad (2.1.1)$$

再加上系统的测量值：

$$Z(k) = H\,X(k) + V(k) \qquad (2.1.2)$$

上两式子中，X(k)是 k 时刻的系统状态，U(k)是 k 时刻对系统的控制量。A 和 B 是系统参数，对于多模型系统，他们为矩阵。Z(k)是 k 时刻的测量值，H 是测量系统的参数，对于多测量系统，H 为矩阵。W(k)和 V(k)分别表示过程噪声和测量噪声。他们被假设成高斯白噪声，他们的协方差分别是 Q，R。

### 2.2 卡尔曼滤波过程

首先我们要利用系统的过程模型，来预测下一状态的系统状态。假设现在的系统状态是 k，根据系统的过程模型，可以基于系统的上一状态而预测出现在状态：

$$X(k|k-1)=A\,X(k-1|k-1)+B\,U(k) \qquad (2.2.1)$$

式(1)中，X(k|k-1)是利用上一状态预测的结果，X(k-1|k-1)是上一状态最优的结果，U(k)为现在状态的控制量。到现在为止，我们的系统结果已经更新。可是，对应于 X(k|k-1)的误差协方差还没更新。我们用 P 表示误差协方差：

$$P(k|k-1)=A\,P(k-1|k-1)\,A' + Q \qquad (2.2.2)$$

式(2)中，P(k|k-1)是 X(k|k-1)对应的误差协方差，P(k-1|k-1)是 X(k-1|k-1)对应的误差协方差，A’表示 A 的转置矩阵，Q 是系统过程的过程噪声协方差。式子 1，2 就是卡尔曼滤波器 5 个公式当中的前两个，也就是对系统的预测。

现在我们有现在状态的预测值，然后我们再收集现在状态的测量值。结合预测值和测量值，我们可以得到现在状态(k)的最优化估算值 X(k|k)：

$$X(k|k) = X(k|k-1) + Kg(k)\,(Z(k) - H\,X(k|k-1)) \qquad (2.2.3)$$

其中 Kg 为卡尔曼增益(Kalman Gain)：

$$Kg(k)= P(k|k-1)\,H' / (H\,P(k|k-1)\,H' + R) \qquad (2.2.4)$$

到现在为止，我们已经得到了 k 状态下最优的估算值 X(k|k)。但是为使卡尔曼滤波器不断的运行下去直到系统过程结束，我们还要更新 k 状态下 X(k|k)的误差协方差：

$$P(k|k)=(I-Kg(k)\,H)\,P(k|k-1) \qquad (2.2.5)$$

其中 I 为单位矩阵。当系统进入 k+1 状态时，P(k|k)就是式子(2)的 P(k-1|k-1)。这样，算法就可以自回归的运算下去。

上述式子 1～5 是卡尔曼滤波的核心算法所在，包括预测值计算，误差协方差计算，最优值估算，卡尔曼增益计算，误差协方差更新等五个重要步骤。

## 3 WebRTC 中 JitterDelay 的计算过程

本节结合 WebRTC 源代码，分析视频接收端缓冲区延迟 JitterDelay 的计算过程。

### 3.1 JitterDelay 的计算公式

由第一节分析可知，JitterDelay 由两部分延迟造成：传输大帧引起的延迟和网络噪声引起的延迟。其计算公式如下：

JitterDelay = theta[0] * (MaxFS – AvgFS) + [noiseStdDevs * sqrt(varNoise) – noiseStdDevOffset]

(3.1.1)

其中 theta[0]是信道传输速率的倒数，MaxFS 是自会话开始以来所收到的最大帧大小，AvgFS 表示平均帧大小。noiseStdDevs 表示噪声系数 2.33，varNoise 表示噪声方差，noiseStdDevOffset 是噪声扣除常数 30。

解码线程从缓冲区获取一帧视频数据进行解码之前，会根据公式 3.1.1 计算当前帧的缓冲区延迟，然后再加上解码延迟和渲染延迟，得到当前帧的预期渲染结束时间。然后根据当前时刻，确定当前帧在解码之前需要等待的时间，以保证视频渲染的平滑性。

### 3.2 JitterDelay 的更新过程

解码线程从缓冲区获取一帧视频数据进行解码之前，会根据当前帧的大小、时间戳和当前本地时刻，更新缓冲区本地状态，包括最大帧大小、平均帧大小、噪声平均值、信道传输速率、网络排队延时等参数。 更新过程如图 1 所示：

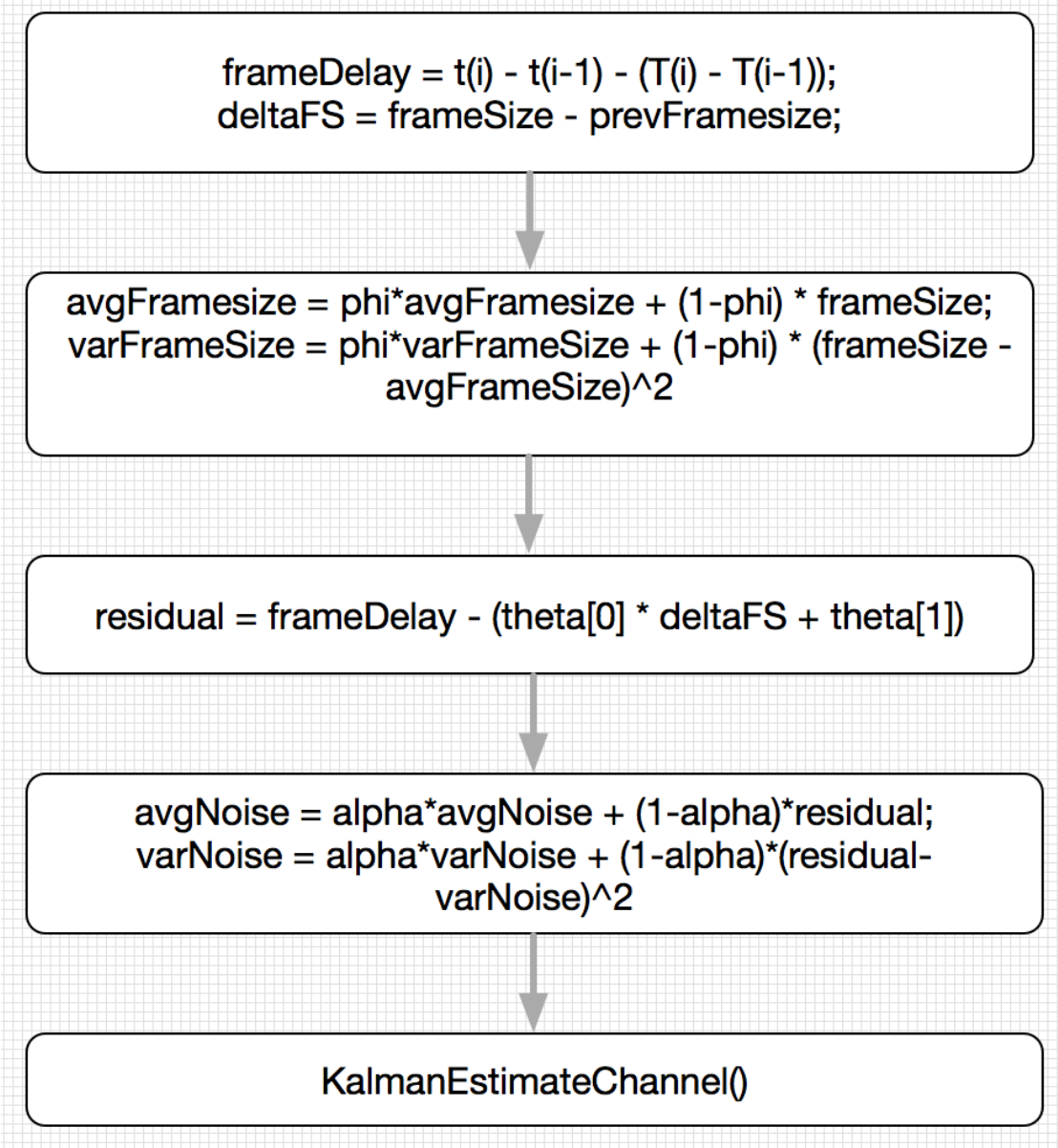


图 1 视频接收端缓冲区状态更新过程

首先，确定帧间相对延迟 frameDelay 和帧间大小差值 deltaFS：

$$frameDelay = t(i) - t(i-1) - (T(i) - T(i-1), \quad (3.2.1)$$

$$deltaFS = frameSize - prevFrameSize, \quad (3.2.2)$$

其中 t(i)表示当前帧本地接收时刻，t(i-1)表示上一帧本地接收时刻；T(i)表示当前帧的时间戳，T(i-1)表示上一帧的时间戳。frameDelay 表示相邻两帧的相对延

迟，frameDelay > 0 表示帧 i 相对于帧 i-1 在路上花费的时间更长。frameSize 和 prevFrameSize 分别表示当前帧和上一帧的大小。

然后计算帧大小的平均值和方差：

avgFrameSize = phi*avgFrameSize + (1-phi)*frameSize (3.2.3)

varFrameSize = phi*varFrameSize + (1-phi)*(frameSize – avgFramesize)^2 (3.2.4)

接下来计算延迟残差(反映网络噪声的大小)，并据此计算噪声均值和方差：

residual = frameDelay – (theta[0] * deltaFSBytes + theta[1]) (3.2.5)

avgNoise = alpha*avgNoise + (1-alpha)*residual (3.2.6)

varNoise = alpha * varNoise + (1 – alpha) * (residual – avgNoise)^2 (3.2.7)

alpha = pow(399/400, 30/fps) (3.2.8)

其中 alpha 表示概率系数，受帧率 fps 影响：当 fps 变低时，alpha 会变小，表示当前噪声变大，噪声方差受当前噪声影响更大一些。实时帧率越接近 30 fps 越好。avgNoise 表示自开始以来的平均噪声。theta[0]表示信道传输速率，theta[1]表示网络排队延迟。

最后一步，卡尔曼滤波器根据当前帧间延迟和帧间大小差值，动态调整信道传输速率 theta[0]和网络排队延迟 theta[1]，具体过程在下一节进行详细讨论。

至此，JitterDelay 的一次更新过程结束。当下一帧数据到来时，使用本次更新结

果计算 JitterDelay，并再次执行更新过程。

## 4 卡尔曼滤波更新缓冲区状态

本节结合 WebRTC 源代码，深入分析卡尔曼滤波更新更新信道传输速率 theta[0]和网络排队时延 theta[1]的过程，这也是缓冲区延迟估计的核心算法所在。

### 4.1 数学符号定义

X_bar：向量 X；

X_hat：向量 X 的估计值；

X(i)：向量 X 的第 i 个分量；

[x y z]：包含元素 xyz 的行向量；

X_bar^T：向量 X_bar 的转置向量；

E{X}：随机变量 X 的期望；

### 4.2 理论推导过程

d(i) = t(i) – t(i-1) – (T(i) – T(i-1))　　(4.2.1)

t(i)表示当前帧本地接收时刻，t(i-1)表示上一帧本地接收时刻；T(i)表示当前帧的发送端采样时刻即时间戳，T(i-1)表示上一帧的采样时刻。d(i) > 0 表示帧 i 相对于帧 i-1 在路上花费的时间更长。式子 4.2.1 是系统的测量值。

d(i) = dL(i) / C(i) + m(i) + v(i)　　(4.2.2)

dL(i)表示帧 i 和帧 i-1 的长度之差，C(i)表示信道传输速率，m(i)表示帧 i 的网络排队时延，v(i)表示测量噪声，其协方差为 R。其中[1/C(i) m(i)]是我们要求的目标值，即信道传输速率和网络排队时延。式子 4.2.2 是系统的预测值。

定义：theta_bar(i) = [1/C(i)　m(i)]^T　为帧 i 状态列向量；

h_bar(i) = [dL(i)　1]^T　　为帧间长度差列向量；

theta_bar(i+1) = theta_bar(i) + u_bar(i)；　u_bar(i)表示过程噪声；

Q(i) = E{u_bar(i) * u_bar(i)^T}　　表示过程噪声协方差矩阵；

diag(Q(i)) = [10^-13 10^-3]^T　　Q(i)是对角阵；

估计过程：

theta_hat(i) = [1/C_hat(i)　m_hat(i)]^T；　　目标估计值

z(i) = d(i) – h_bar(i)^T * theta_hat(i-1)

= d(i) – (dL(i)/C_hat(i-1) + m_hat(i-1))　　(4.2.3)

上述式子的意义：用上一时刻估计值估算本时刻的时间消耗：

dL(i)/C_hat(i-1) + m_hat(i-1)

然后用当前观测值 d(i)和估算值求出残差 z(i)。

theta_hat(i) = theta_hat(i-1) + z(i) * k_bar(i)　　(4.2.4)

上述式子表示 i 时刻和 i-1 时刻的状态迭代关系：i-1 时刻的状态加上 i 时刻的残差 z(i)和 i 时刻卡尔曼滤波的乘积，其中 i 时刻的 kalman gain 计算公式如下：

k_bar(i) = ( (E(i-1) + Q(i)) * h_bar(i) ) / (var_v_hat(i) + h_bar(i)^T * (E(i-1) + Q(i)) * h_bar(i))　　(4.2.5)

E(i) = (I – k_bar(i) * h_bar(i)^T) * (E(i-1) + Q(i))　　(4.2.6)

var_noise(i) = max(alpha * var_noise(i-1) + (1 – alpha) * z(i)^2, 1)　(4.2.7)

alpha = pow(399 / 400,　30 / fps)　　(4.2.8)

var_v_hat(i) = (300.0 * exp(-fabs(deltaFS) / (1e0 * maxFrameSize)) +

1) * sqrt(varNoise)　　(4.2.9)

其中 I 是 2*2 单位阵，E(i)是误差协方差(它是卡尔曼滤波迭代的关键参数)。

var_noise 是噪声方差，var_v_hat 是噪声标准差指数过滤平均后取值，其实就是帧 i 的测量噪声协方差 R，它对最终计算出的卡尔曼增益有较大影响：var_v_hat(i)较大时，卡尔曼增益较小，表示本次测量中噪声较大，最终估计值更靠近上次估计值而较少受本次残差的影响。

### 4.3 WebRTC 代码实现

下面伪代码是对 WebRTC 中 VCMJitterEstimator::KalmanEstimateChannel 函数的精简概括，描述了卡尔曼滤波的代码实现。

```
void VCMJitterEstimator::KalmanEstimateChannel(frameDelayMS, deltaFSBytes) {
  // 计算误差协方差和过程噪声协方差的和：E = E + Q；
  _thetaCov[0][0] += _Qcov[0][0];
  _thetaCov[0][1] += _Qcov[0][1];
  _thetaCov[1][0] += _Qcov[1][0];
  _thetaCov[1][1] += _Qcov[1][1];

  // 计算卡尔曼增益:
  // K = E*h'/(sigma + h*E*h')
  // h = [deltaFS 1], Eh = E*h'
  // hEh_sigma = h*E*h' + sigma
  Eh[0] = _thetaCov[0][0] * deltaFSBytes + _thetaCov[0][1];
  Eh[1] = _thetaCov[1][0] * deltaFSBytes + _thetaCov[1][1];
  // sigma为测量噪声标准差的指数平均滤波结果，即测量噪声协方差R。
  double sigma = (300.0 * exp(-fabs(static_cast<double>(deltaFSBytes)) /
                                        (1e0 * _maxFrameSize)) +1) * sqrt(varNoise);
  hEh_sigma = deltaFSBytes * Eh[0] + Eh[1] + sigma;
  kalmanGain[0] = Eh[0] / hEh_sigma;
  kalmanGain[1] = Eh[1] / hEh_sigma;

  // 计算残差，获得最优估计值
  measureRes = frameDelayMS - (deltaFSBytes * _theta[0] + _theta[1]);
```

```
    _theta[0] += kalmanGain[0] * measureRes;
    _theta[1] += kalmanGain[1] * measureRes;

    // 更新误差协方差：E = (I - K*h)*E；
    t00 = _thetaCov[0][0]; t01 = _thetaCov[0][1];
    _thetaCov[0][0] = (1 - kalmanGain[0] * deltaFSBytes) * t00 -
                      kalmanGain[0] * _thetaCov[1][0];
    _thetaCov[0][1] = (1 - kalmanGain[0] * deltaFSBytes) * t01 -
                      kalmanGain[0] * _thetaCov[1][1];
    _thetaCov[1][0] = _thetaCov[1][0] * (1 - kalmanGain[1]) -
                      kalmanGain[1] * deltaFSBytes * t00;
    _thetaCov[1][1] = _thetaCov[1][1] * (1 - kalmanGain[1]) -
                      kalmanGain[1] * deltaFSBytes * t01;
}
```

至此，卡尔曼滤波估计信道发送速率和网络排队时延的一次迭代完成。

## 5 总结

本文在理论学习卡尔曼滤波基本原理的基础上，结合 WebRTC 源代码，深入分析了 WebRTC 视频接收端缓冲区延迟的计算方法和更新过程。通过本文，更进一步加深对 WebRTC 的学习和了解。

## 参考文献


[1] 卡尔曼滤波的原理说明

http://blog.sciencenet.cn/blog-1009877-784428.html

[2] 卡尔曼滤波的基本原理及应用[J]. 软件导刊, Vol.8 No.11, Nov. 2009.

[3] An Introduction to the Kalman Filter [J]. University of North Carolina at Chapel Hill Department of Computer Science Chapel Hill, NC 27599-3175

[4] A Google Congestion Control Algorithm for Real-Time Communication https://tools.ietf.org/html/draft-alvestrand-rmcat-congestion-03

[5] Analysis and Design of the Google Congestion Control for Web Real-time Communication[J]. MMSys '16 Article No.13