文章编号:1001 - 9081(2006)05 - 1137 - 04

基于速度控制的 API 网络拥塞控制策略

陆锦军[1,2],王执铨[1]
(1. 南京理工大学 自动化学院,江苏 南京 210094;
2. 南通职业大学 现代教育技术中心,江苏 南通 226007)
(ljj@mail.ntvc.edu.cn)

摘 要:针对 PI 控制器响应速度方面的不足,提出了一种基于速度控制的自适应网络动态变化新的主动队列管理算法——API-V 控制器。在 PI 控制器的基础上,根据瞬时队列长度,增加速度控制;根据实时测量链路的数据包丢失率,获得当前的负载信息,动态调整 PI 算法中的有关参数。理论分析和仿真表明,API-V 控制器相对于 PI 控制器及其改进算法,不仅具有更快响应速度和收敛速度、更小的队列抖动,而且提高了缓冲区的利用率。

关键词:速度控制;AQM;API;API-V
中图分类号: TP393 **文献标识码**:A

API network congestion control scheme based on velocity control

LU Jin-jun[1,2], WANG Zhi-quan[1]
(1. *School of Automation, Nanjing University of Science and Technology, Nanjing Jiangsu* 210094, *China;*
2. *Center of Education and Technology, Nantong Vocational College, Nantong Jiangsu* 226007, *China*)

Abstract: An adaptive proportional integral-velocity(API-V) Controller for new active queue management scheme was proposed, which is well suited to dynamic network environments based on velocity control, in order to address the slow response of PI controller. Velocity controller was added based on PI controller when the length of queue is larger. The controller obtains present load information according to present measured dropping probability, then sets up the relevant parameters in PI scheme dynamically. Analysis and simulation results show that API-V controller can not only achieve faster convergence speed, faster response and smaller queue oscillation than PI controller and improved PI controller but also increase the buffer utilization.

Key words: control of velocity; AQM; API; API-V

0 引言

随着网络规模的不断扩大,网上业务量的增长,网络拥塞现象越来越明显,于是应用控制理论分析 TCP/AQM 模型并设计相应的主动队列管理(AQM)算法是当前 TCP/IP 网络拥塞控制研究领域中的热点之一。

Floyd 提出的 RED(random early detection)算法和 PI 控制器是两种著名的主动队列管理算法,RED 算法是根据平均队列长度计算报文丢弃概率。理论分析和实验研究表明,RED 及其改进算法的性能和配置参数紧密相关,不正确的参数配置导致很大的队列抖动[1],与其他主动队列管理算法比较,自适应 RED[2,3]算法对 Web 的性能改进最小,而且缺乏严谨的科学论证。自适应虚缓冲[4]和 REM[5]算法是基于优化的方法,更多地关注系统的稳态特性,而不是队列的瞬态性能。文献[6]建立了 TCP/AQM 控制论模型,从而可以使用控制理论研究主动队列管理算法。Hollot 基于对建立的控制论模型的线性化,提出了 PI 算法。理论分析和实验都表明,PI 算法比 RED 算法具有更小的队列抖动,从而在保证高带宽利用率的前提下,为端用户提供更小的延时抖动。但是 PI 算法中,参数是固定设置的,因此导致 PI 算法在较小的目标队列长度情况下收敛速度较慢。针对 PI 算法的缺点,先后出现了一些改进算法,但效果仍不够理想。

本文提出了一种新的自适应主动队列管理算法——API-V控制器,当瞬时队列长度小于启动门限值 H 时,实现 API 算法,即根据实时测量链路的数据包丢失率,动态调整 PI 算法的配置参数,保证 PI 算法有更快的收敛速度和更小的队列抖动;当 IP 流量突发,瞬时队列长度大于启动门限值 H 时,启动速度控制项,实现 API-V 算法,保证对网络状态变化的快速响应,并且提高了缓冲区的利用率。理论分析和仿真表明,API-V 算法在动态网络环境下,性能优于 PI 算法及文献[7~9]中的改进算法。

1 TCP/AQM(PI)控制理论模型

TCP/AQM(PI)系统如图 1 所示。

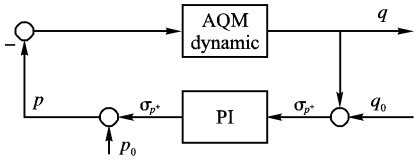

图 1 TCP/AQM 控制系统(PI)

图中 AQM 模块可表示为:

$$G_1(S) = \frac{K_m e^{-SR}}{(T_1 S + 1)(T_2 S + 1)}$$

PI 模块:

收稿日期:2005 - 11 - 15;**修订日期**:2006 - 02 - 15 **基金项目**:国家自然科学基金资助项目(60374066);江苏省自然科学基金(BK2004 132)
作者简介:陆锦军(1964 -),男,江苏南通人,副教授,主要研究方向:网络系统的拥塞控制、智能控制; 王执铨(1939 -),男,湖北武汉人,教授,博士生导师,主要研究方向:网络系统的拥塞控制、混沌控制、鲁棒控制、容错控制、信息安全.

$$G_2(S) = \frac{1+\tau S}{TS}$$

其中：$K_m = \frac{(RC)^3}{4N^2}$；$T_1 = \frac{R^2C}{2N}$；$T_2 = R$；$\tau$，$T$ 是待确定的参数；q_0 是目标队列长度；P 是数据包丢失率。

系统的开环传递函数为：

$$G(S) = \frac{1+\tau S}{TS} \times \frac{K_m e^{-RS}}{(T_1 S+1)(T_2 S+1)} \tag{1}$$

当系统稳定时，$P(k)$ 在 k 时刻的数据包丢失率为：

$$p(k) = a[q(k) - q_0] - b[q(k-1) - q_0] + p(k-1)$$

其中：$q(k)$ 是在 k 时刻的瞬时队列长度，f 是 PI 算法采样的频率。$a = \frac{\tau}{T} + \frac{1}{Tf}$。$b = \frac{\tau}{T}$。

参数 a，b 是固定不变的，不能随网络动态变化而灵活设置，因而导致 PI 算法收敛速度较慢，而且响应时间受缓冲区大小的影响。因此，本文提出了自适应网络动态变化且响应速度和收敛速度更快的新算法——API-V 控制器。

2 API-V 控制器

2.1 API-V 控制器模型

API-V 控制器不仅在 PI 基础上实现 API 算法，而且在 API 控制器基础上增加了归一化的速度控制项——V/C，其计算分组标记/丢弃概率的差分方程为：[10]

$$p(k) = \begin{cases} p(k-1) + a[q(k) - q_0] - \\ \quad b[q(k-1) - q_0] + \lambda\left[\frac{V(k)}{C}\right], & q > H \\ p(k-1) + a[q(k) - q_0] - \\ \quad b[q(k-1) - q_0], & q \leqslant H \end{cases} \tag{2}$$

其中，λ 为常数；H 为速度控制启动门限；$V(k)$ 为第 k 个抽样时间的瞬时输入速度；C 为链路带宽；$P(k)$ 为第 k 个抽样时间得到的丢弃概率；$q(k)$ 为第 k 个抽样时间得到的瞬时队长。a，b 常数与 PI 控制中不一样不是定值，随着网络动态变化而调整。当瞬时队列长度大于启动门限值 H 时，在 API 基础上，启动速度控制项，实现 API-V 控制；反之，保持 API 控制器，即根据实时测量链路中的数据丢失率，获得当前负载信息，然后动态设置 PI 控制器的相关参数。当前节点的负载变化，引入速度控制使得算法对网络环境变化的响应速度更快，减小了由于计算队列长度引起的时延对响应速度的影响。而且只有当瞬时队列长度超出期望的队列长度一定的范围时才启动速度控制，这样既能进一步提高响应速度，也保留了 API 控制器在保持队长稳定系统响应方面的优点，也消除了不同链路带宽对丢失概率值的影响。式(2)中，λ 为速度控制项的系数，取值过小，速度控制作用不突出，影响响应速度；取值过大，使得丢弃概率过大，造成队长的不稳定，也将使 API 控制部分失去作用。根据仿真经验，一般情况下，$a \leqslant \lambda \leqslant 10a$，$2\,q_0 \geqslant H \geqslant 1.5\,q_0$。[10]

2.2 API-V 算法本质及其分析

在 IP 网络中，如果负载突然增加（$q > H$），启动速度控制，设队列长度为 H，$p(0) = 0$，由此得到 $p(k)$ 的表达式为：

$$p(k) = k[(a-b)(H-q) + \lambda V/C] \tag{3}$$

令 API-V 控制器的响应时间为 T_r，队长由 H 降到 q_0 所需的丢弃概率为 p，则有：

$$T_r = \frac{(p-0)T}{p(k) - p(k-1)} = \frac{pT}{(H-q_0)(a-b) + \lambda V/C} \tag{4}$$

由(4)可以看出，由于在 API-V 中引入了速度控制，相对于 API 的响应时间公式，就是在分母中增加了 $\lambda V/C$ 一项，所以相对于 PI 控制器，API-V 控制器响应时间要变小。同时也可以看出，如果要求到达相同的响应速度，API-V 只需要很小的缓冲区间，即可提高缓冲区的利用率。这样，当网络环境变化时，API-V 能使队列长度以最短的时间稳定到 q_0 附近，从而保证时延，链路利用率等 QoS 的要求。

设在 t 时刻，节点处汇聚的瞬时输入速度为 $r(t)$，瞬时队列长度为 $q(t)$，则瞬时队列长度和瞬时输入速度的关系为[11]：

$$q'(t) = v(t) - C \tag{5}$$

其差分方程为：

$$\frac{q(k) - q(k-1)}{T} = v(k) - C \tag{6}$$

将上式代入式(2)可得：

$$p(k) = p(k-1) + a[q(k) - q_0] - b[q(k-1) - q_0] + \frac{\lambda}{C}\left[\frac{q(k) - q(k-1)}{T} + C\right] \tag{7}$$

式(7)即为 PID 控制器的差分方程，在 S 域中表示为：

$$C(S) = K_p + K_i/S + K_d S$$

$C(S)$ 为一个参数可调 PID 控制器的传递函数，TCP/AQM 的线性控制系统如图 2 所示。

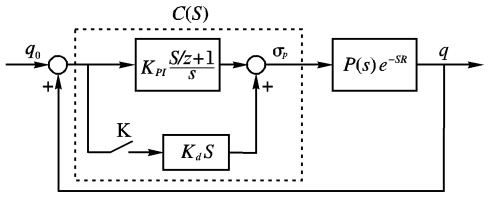

图 2 API-V-TCP/AQM 线性控制系统

其中，$P(S)e^{-SR}$ 为需要控制的 TCP/AQM 系统；其模型为：

$$\frac{\frac{(RC)^3}{4N^2}e^{-RS}}{\left(\frac{R^2C}{2N}S+1\right) \times (RS+1)}$$

R 为平均往返时间，σ_p 为当前丢失概率与工作点丢失概率 p_0 的差值。控制器根据当前队长和期望队长的差值计算得出当前丢失概率，由于源端 TCP 流对分组丢弃做出响应，控制分组发送速度进而影响当前队长。当 $q > H$ 时，开关闭合，$C(s)$ 为一个自适应的 PID 控制器；当 $q < H$ 时，开关打开，$C(S)$ 为一个自适应的 API 控制器。

3 API 控制器

3.1 根据数据包丢失率推算链路负载信息的理论依据

根据文献[12]，TCP 的吞吐量的计算表达式为：

$$T = \frac{MSS}{R\sqrt{\frac{2P}{3}} + \tau_{RTO}\left(\frac{3}{2}\sqrt{\frac{3P}{2}}\right)P(1+32P^2)} \tag{8}$$

其中 R，P 意义同前，τ_{RTO} 是连接超时值，MSS 是报文大小，一般取值 $\tau_{RTO} = 5R$，式(8)变为：

$$T = \frac{MSS}{R\left[\sqrt{\frac{2P}{3}} + \left(\frac{15}{2}\sqrt{\frac{3P}{2}}\right)P(1+32P^2)\right]} \tag{9}$$

令　$h(p)=\dfrac{\sqrt{\frac{2p}{3}}+\left(\frac{15}{2}\sqrt{\frac{3p}{2}}\right)p(1+32p^2)}{MSS}$

若 TCP 链路变为 C,连接的数据为 N,每条连接的 PTT 为 R,则(9)式变为:

$$T=\frac{1}{h(p)R} \tag{10}$$

$$T=\frac{C}{N}=\frac{MSS}{R\left[\sqrt{\frac{2P}{3}}+\left(\frac{15}{2}\sqrt{\frac{3P}{2}}\right)P(1+32P^2)\right]}=\frac{1}{h(p)R} \tag{11}$$

令　$a=\left[\sqrt{\frac{2P}{3}}+\left(\frac{15}{2}\sqrt{\frac{3P}{2}}\right)P(1+32P^2)\right]\cdot\frac{C}{MSS}=h(p)C$ (12)

那么　$N=\alpha R$ (13)

定义集合 $\Omega_p=\{<N,R>\mid N-h(p)CR=0\}$,若链路丢失率 P 已知,可推测负载 N 和 R。

3.2　API 算法参数调整的理论依据

定理 1[13]　对于 N,R 变化的系统而言,当数据丢失率为 P,取:

$$\omega_g=\min\left\{\frac{\beta}{\sqrt{T_1T_2}}\mid (N,R)\in\Omega_p\right\}$$

$$\beta\leqslant\frac{\pi/4}{\sqrt{2h(p)}}$$

$$\tau\geqslant\frac{1}{\omega_g}$$

$$T\geqslant\max\left\{\frac{k_m\tau}{\sqrt{(1-T_1T_2\omega^2)^2+(T_1+T_2)^2\omega_g^2}}\right|<N,R>\in\Omega_p\Bigg\}$$

则 $\forall\ <N,R>\in\Omega_p$,系统稳定。

证明　对于 $\forall\ <N,R>\in\Omega_p$,令 $\omega'_p=\dfrac{\beta}{\sqrt{T_1T_2}}$,则 $\omega_g\leqslant\omega'_g$,系统的频率响应为:

$$|G(j\omega)|=\frac{\sqrt{1+\tau^2\omega^2}}{T\omega}\times\frac{K_m}{\sqrt{(1-T_1T_2\omega^2)^2+(T_1+T_2)^2\omega^2}}$$

$$|G(j\omega_g)|\approx\frac{\tau}{T}\times\frac{K_m}{\sqrt{(1-T_1T_2\omega_g^2)^2+(T_1+T_2)^2\omega_g^2}}\leqslant\frac{\sqrt{(1-T_1T_2\omega_g^2)^2+(T_1+T_2)^2\omega_g^2}}{K_m}\times\frac{K_m}{\sqrt{(1-T_1T_2\omega_g^2)^2+(T_1+T_2)^2\omega_g^2}}=1$$

相角频率特性:

$$\varphi(\omega_g)=-90^\circ+\operatorname{arctg}(\tau\omega_g)-\operatorname{arctg}\frac{(T_1+T_2)\omega_g}{1-T_1T_2\omega_g^2}-\omega_gR\geqslant-90^\circ+\operatorname{arctg}(\tau\omega_g)-\operatorname{arctg}\frac{(T_1+T_2)\omega_g}{1-T_1T_2\omega_g^2}-\omega'_gR\geqslant-180^\circ$$

由 Nyquist 稳定性判据可知 $\forall\ <N,R>\in\Omega_p$ 系统稳定。

定理 2[13]　对于 N,R 变化的系统而言,当报文丢失率为 p,最大的 RTT 为 $R_{\max}$ 时,取 $\omega_g=\dfrac{\pi/4}{R_{\max}}$,$\tau\geqslant\dfrac{1}{\omega_g}$,

$$T\geqslant\max\left\{\frac{k_m\tau}{\sqrt{(1-T_1T_2\omega^2)^2+(T_1+T_2)^2\omega^2}}\right|<N,R>\in\Omega_p\Bigg\}$$

则 $\forall\ <N,R>\in\Omega_p$,系统稳定。

证明　由定理 1 可知,只要证明:

$$\omega_g=\min\left\{\frac{\beta}{\sqrt{T_1T_2}}\mid <N,R>\in\Omega_p\right\}=\frac{\pi/4}{R_{\max}}$$

取 $\beta=\dfrac{\pi/4}{\sqrt{2h(p)}}$,由前证可知:

$$\omega_g=\min\left\{\frac{\beta}{\sqrt{T_1T_2}}\mid <N,R>\in\Omega_p\right\}=\frac{\pi/4}{\sqrt{2h(p)}}\times\min\left\{\frac{1}{\sqrt{T_1T_2}}\mid <N,R>\in\Omega_p\right\}=\frac{\pi/4}{\sqrt{2h(p)}}\times\min\left\{\frac{\sqrt{2h(p)}}{R}\mid <N,R>\in\Omega_p\right\}$$

由于:$T_1(N,R)=\dfrac{R^2C}{2N}=\dfrac{R^2C}{2\alpha R}=\dfrac{R}{2h(p)}$

$$=\frac{\pi}{4}\times\min\left\{\frac{1}{R}\mid <N,R>\in\Omega_p\right\}=\frac{\pi/4}{R_{\max}}$$

说明 API 算法比 PI 算法具有更大的开环带宽,开环带宽越大,系统的收敛速度越快;当 $N<N_{\min}$ 时,由 PI 算法给出的稳定条件可能使系统不稳定,但按定理进行参数调整,仍可使系统处于稳定状态。当丢失率为 P 时,API 的稳定条件比 PI 给出的稳定条件更加严格。

3.3　关于丢失率 P 的改进计算

文献[14]中数据包的标注概率计算为瞬时队列长度的线性函数,文献[15]论述了几何随机数计算 P 方法的不足之处,本文采用均匀分布的随机数法。

每个包的标注概率均为 p,每两个标注包之间的间隔次数 X 为所到达的包的数目,为了使 X 为 $\{1,2,\cdots,1/p\}$ 中的均匀分布随机数(简单起见,假设 $1/p$ 是一个整数)。对每个达到的数据包的标注概率取为 $p/(1-count\cdot p)$(其中 $count$ 是自上一个标注的包以来所达到的未标注包的数目),实际也就是在取 X 为 $\{count,\cdots,1/p\}$ 中的均匀分布随机数($count$ 可为 $1,2,3,\cdots,1/p$)。在这种情况下,

$$\text{prob}[x=n]=\frac{p}{1-(n-1)p}\prod_{i=0}^{n-2}\left(1-\frac{p}{1-ip}\right)=\frac{p}{1-(n-1)p}\cdot(1-p)\cdot\frac{1-2p}{1-p}\cdot\frac{1-3p}{1-2p}\cdot\cdots\cdot\frac{1-(n-1)p}{1-(n-2)p}=p,\qquad 1\leqslant n\leqslant 1/p \tag{14}$$

$$\text{Prob}[x=n]=0,\qquad n>1/p \tag{15}$$

$$E[X]=\sum_{k=1}^{1/p}kp(X=k)=p+2p+\cdots+\frac{1}{p}=p\left(1+2+\cdots+\frac{1}{p}\right)=1/(2p)+1/2 \tag{16}$$

这种方法比其他方法的均值小很多(因为正常的话丢失的概率是值较小的小数),可以看出这种方法两个标注包之间的间隔包的数目是相对比较规则,分布比较均匀。

3.4 API 算法设计

根据前述理论依据和定理,设计 API 算法,即根据丢失率 P 推算参数 a,b 数值。

此处仅给出主要程序段,所用变量均要先定义:

```
const float C = C1;
const float Pi = 3.14;
const float R1 = 常量;
int i = 0
do {
   i++;
   P = i/100;
   if (p ==0)  p = 0.001;
   Alpha = ( sqrt(2 * p/3 + (15/2) * sqrt(3 * p/2)
         * p * (1 +32 * p * p)) * C/MSS;
   R = R1;                                          //令 R = R1
   N =  Alpha * R1;                                 //则 N = AlphaR1
   T1 = (R1 * R1 * C)/(2 * N);
   T2 = R1;
   KM = ((R1 * C) * CR1 * C) * (R * C))/(4 * N * N)
   ωg = (Pi/4)/R ;                                  //R 相当于 Rmax
   t = 1/ωg;
   T = KM/sqrt((1 - T1 * T2 * ωg * ωg)(1 - T1 * T2 * ωg * ωg)
       + (T1 + T2) * (T1 + T2) * ωg * ωg) ;
   //据 t 和 T 求 a, b 值
   a = t/T + 1/(T * f);                             //f 为常量
   b = t/T;
} while( i <40);
```

4 仿真及其分析

下面利用 NS2[16] 进行仿真,验证 API-V 控制器的性能,同时与 PI 控制器进行了比较,模拟拓扑如图 3 所示。

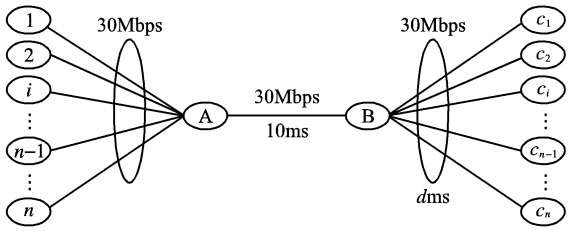

图 3 模拟拓扑

其中 1 ~ n 为发送节点,C_1 ~ C_n 为接收节点,A、B 为路由器,$i \rightarrow A$ 和 $B \rightarrow C_i$ 的传输延迟为[10ms,50ms]之间平均分布的随机延迟,$A \rightarrow B$ 的链路构成瓶颈。A 上分别运行 API-V 和 PI 算法。目标队列长度设置为 100 个数据包。TCP 使用 Reno,数据包大小为 1000 字节;CBR 报文大小为 500 字节,报文发送间隔为 0.5s;HTTP 短连接的大小为 10k 字节,PI 和 API-V 算法的参数设置如表 1。

表 1 对比算法参数设置

	A	b	T/s	λ	H
PI	1.812×10^{-5}	1.826×10^{-5}	1/180		
API-V	1.812×10^{-5}	1.826×10^{-5}	1/180	10×10^{-5}	150

实验 1:比较了 API-V 和 PI 算法的响应速度。在 0s ~ 10s 之间启动 200 条 FTP 连接,然后在 50s ~ 60s 之间再启动 200 条 FTP 连接,总的模拟过程持续 100s,瞬时队列长度随时间变化的过程如图 4 所示。从图 4 可以看出,API-V 算法不仅具有较快的收敛速度,而且具有比 PI 算法更小的队列抖动。

实验 2:验证了 API-V 在重负载情况下的性能。在 0s ~ 10s 之间启动 500 条 TCP 连接。模拟结果如图 5 所示。从图 5 可以看出,在重负载情况下,PI 算法的收敛速度很慢,在重负载情况下,API-V 算法虽然经过了 API 算法到 API-V 算法的转变,但响应曲线比较平滑,其算法响应时间近 10s。由此可知,API-V 算法性能明显优于 PI 算法。

实验 3:测试了 API-V 算法在混合连接类型情况下的性能。在 0s ~ 10s 之间,启动 150 条 FTP 连接,在 150s 时停止这些连接,在 20s ~ 30s 之间启动 100 条 CBR 连接,在 120s 时停止这些连接,在 40s ~ 50s 时启动 200 条 HTTP 连接,在 60s ~ 70s 之间再启动 70 条 FTP 连接,这些连接在 140s 时停止,在 80s ~ 90s 之间启动 100 条 CBR 连接,在 160s 时停止这些连接。模拟结果如图 6 所示,从实验 3 可以看出,在有短连接(HTTP)和 UDP(CBR)连接存在的情况下,API-V 算法仍然具有较快的收敛速度和较小的队列长度抖动,性能优于 PI 算法。

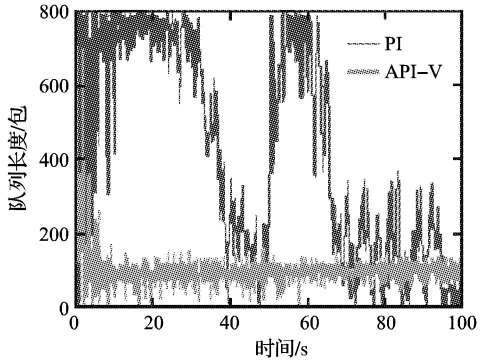

图 4 实验 1 模拟结果

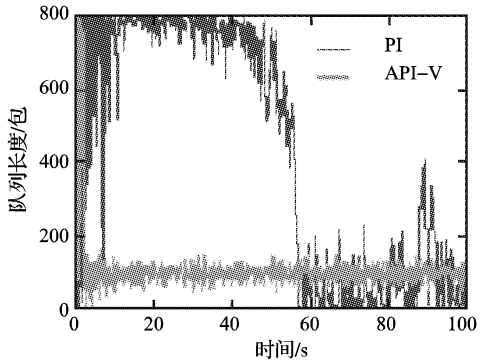

图 5 实验 2 模拟和结果

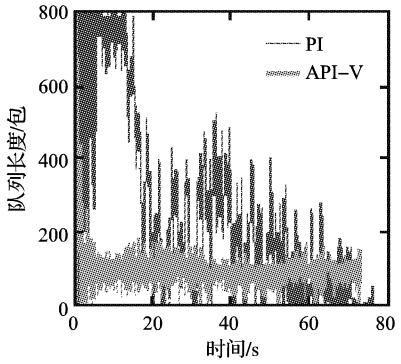

图 6 实验 3 模拟和结果

5 结语

本文针对 PI 算法的缺点提出了一种新的自适应 AQM 算法——API -V 控制器,当瞬时队列长度小于启动门限值 H 时,实现 API 算法 ,即根据实时测量链路的数据包丢失率,动态调整 PI 算法的配置参数,保证算法有更快的收敛速度和更小的队列抖动,当IP流量突发,瞬时队列长度大于启动门限

(下转第 1143 页)

它以执行二进制代码的方式控制每个功能模块,可以在数据流的适当位置打开/关闭各个功能引擎,以对数据进行不同的处理。通过编程可以使 DPU 处理不同的安全协议,它是 HIPP 芯片可以支持多安全协议的关键模块。

在 Encode 时,输入为 RTP 包,输出为 SRTP 包,Decode 时相反。会话所采用的加密、认证算法由配置模块设置,而对每个包的处理由 DPU 通过执行事先编译好的可执行代码来控制,作者已经用 DPU 的指令集实现了 SRTP(包索引维护算法采用改进的算法 2),并编译成二进制可执行代码,DPU 执行该代码就可以完成对 SRTP 处理的控制。

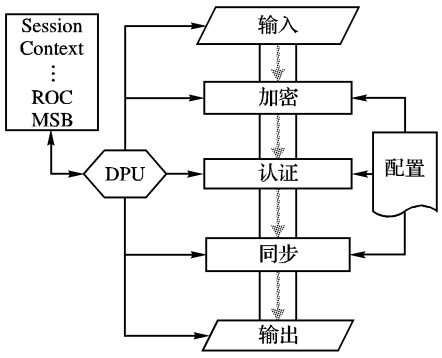

图 3　HIPP 的流水体系结构

HIPP 体系结构下,芯片在做 Decode 处理时认证引擎计算并比较 HMAC 的结果,该结果由认证引擎以一个标志位通知发送该包到芯片来处理的客户程序,而 DPU 无法知道该结果(这样设计的目的是优化整个流水线的效率,DPU 无需等待认证结束而可以尽快启动下一个操作)。SRTP 实现中 ROC 等的更新维护由 DPU 控制,更新前 DPU 必须知道认证的结果,DPU 完成这个功能需要控制 3 遍流水操作,认证引擎只负责计算 HMAC,其比较由 DPU 自己来完成:

(1) 正常处理,如果发现 ROC 与 MSB 需要更新,触发第二遍流水;

(2) 控制认证引擎计算 HMAC 并将结果添加到数据流的末尾,触发第三遍流水;

(3) DPU 比较第二遍计算的 HMAC 值与 SRTP 包中的 HMAC 值是否匹配,如果匹配则进行相应更新,否则返回错误值。最后结束该包的操作。

而如果不需要更新 ROC 等,DPU 也就不需要知道与等待认证的结果(HMAC 的比较可以由认证引擎完成),所有操作可以一遍流水完成。在算法 1 的情况下,需要更新 ROC 与s_l 的概率很高,所以需要多遍流水处理的概率也很高。而算法 2 把多遍处理的概率降低到了 $1/2^{15}$,基本上忽略不计,可以成倍的提高效率。

4　结语

为满足实时的要求,RTP 包一般比较小。SRTP 在为 RTP 提供安全特性的同时所增加的额外数据的大小将会严重影响 RTP 通信效率及实时性。32 位的 ROC 记录了 RTP 包序列号溢出 16 位的次数,但它不在 SRTP 包中传输,而由通信各方共同维护与同步。

ROC 值的同步本身并不复杂,RFC3711 提供了一个算法,很好地完成了同步的功能,但由于该算法要求更新 ROC 与 s_l 的概率太高,并且更新必须在认证通过后进行,导致在硬件流水或者多处理器体系结构下算法实现的代价很高。本文提出一个功能一致但更新概率很低的算法,很好地解决了效率问题,并以美国 Hifn 公司的 HIPP 芯片为例讨论了改进前后算法在实现效率上的区别。

参考文献:

[1] The Secure Real-time Transport Protocol[S]. RFC3711.

[2] RTP: A Transport Protocol for Real-Time Applications [S]. RFC3550.

[3] PATTERSON DA, HENNESSY JL. Computer Architecture: A Quantitative Approach[M]. 3rd edition Chapter 6. 北京: 机械工业出版社, 2002.

[4] http://www.hifn.com/products/Security.html[EB/OL].

(上接第 1140 页)

值 H 时,启动速度控制项,实现 API-V 算法,快速响应网络的动态变化。理论分析和模拟仿真实验表明,API-V 算法能够自适应动态变化的网络环境,性能优于 PI 算法及其已有的改进算法。

参考文献:

[1] HOLLOT CV, MISRA V, TOWSLEY D, *et al.* A control theoretic analysis of RED[A]. AMMAR M, ed. Proceedings of the IEEE INFOCOM[C]. Anchorage: IEEE Communications Society, 2001. 1510 - 1519.

[2] LE L, AIKAT J, JEFFAY K, *et al.* The effects of active queue management on Web performance[A]. Proceeding of the ACM SIGCOMM 2003[C]. Karlsruhe, 2003. 265 - 276.

[3] FLOYD S, GUMMADI R, SHENKER S. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management[EB/OL]. http://www.icir.org/~floyd, 2001.

[4] KUNNIYUR S, SRIKANT R. A time scale decomposition approach to adaptive ECN marking[A]. AMMAR M, ed. Proceedings of the IEEE INFOCOM[C]. Anchorage: IEEE Communications Society, 2001. 1330 - 1339.

[5] ATHURALIYA S, LOW S, LI VH, *et al.* REM: Active queue management[J]. IEEE Network, 2001, 15(3): 48 - 53.

[6] MISRA V, GONG WB, TOWSLEY D. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED[A]. Proceedings of the ACM SIGCOMM 2000[C]. Stockholm, 2000. 151 - 160.

[7] CHEN Q, YANG OWW. A ST-PI-PP controller for AQM router[J]. IEEE International Conference on Communications, 2004, 27(1): 2277 - 2281.

[8] WANG CG, LI B, SOHRABY K. API: adaptive proportional-integral algorithm for active queue management under dynamic environments[A]. Proceeding IEEE HPSR 2004[C]. 2004. 51 - 55.

[9] CHANG XL, MUPPALA JK, JEN-TE YU. A robust nonlinear PI controller for improving AQM performance[A]. IEEE International Conference on Communications[C], 2004. 20 - 24.

[10] ZENG ZM, ZHANG TK, FEBNG CY, *et al.* An AQM School for Fast Response[J]. Journal of Beijing University of posts and Telecommunications, 2005, 28(4): 5 - 9.

[11] PARK EC, LIM H, PARK KJ, *et al.* Analysis of the virtual rate control algorithm in TCP networks[J]. Globecom '02. IEEE, 2002, 3: 2619 - 2623.

[12] PADHYE J, FIROIU V, TOWSLEY D, *et al.* Kurose J. Modeling TCP throughput: A simple model and its empirical validation[A]. ORAN D, ed. Proceedings of the ACM SIGCOMM[C]. Vancouver: ACM Press, 1998. 303 - 314.

[13] LU XC, ZHANG MJ, ZHU PD. API: A Adaptive PI Active Queue Management Algorithm[J]. Journal of software, 2005, 16(5): 903 - 910.

[14] HOLLOT CV, MISRA V, TOWSLEY D, *et al.* Analysis and design of controllers for AQM routers supporting Tcp flows[J]. IEEE Transactions on Automatic Control, 2002, 47(6): 945 - 959.

[15] CAI XL, WANG XF, WANG ZQ, *et al.* Analysis and Improvement of PI control for Active queue Management[J]. Journal of Nanjing University of Science and Technology, 2005. 29(3): 368 - 371.

[16] VINT project U C Berkeley/LBNL, ns2[J/OL]. http://www.isi.edu/nsman/ns/ns - allione, 2004 - 04 - 05.