Gallop-Vegas: An Enhanced Slow-Start Mechanism for TCP Vegas

Cheng-Yuan Ho, Yi-Cheng Chan, and Yaw-Chung Chen

***Abstract:* In this article, we present a new slow-start variant, which improves the throughput of transmission control protocol (TCP) Vegas. We call this new mechanism Gallop-Vegas because it quickly ramps up to the available bandwidth and reduces the burstiness during the slow-start phase. TCP is known to send bursts of packets during its slow-start phase due to the fast window increase and the ACK-clock based transmission. This phenomenon causes TCP Vegas to change from slow-start phase to congestion-avoidance phase too early in the large bandwidth-delay product (BDP) links. Therefore, in Gallop-Vegas, we increase the congestion window size with a rate between exponential growth and linear growth during slow-start phase. Our analysis, simulation results, and measurements on the Internet show that Gallop-Vegas significantly improves the performance of a connection, especially during the slow-start phase. Furthermore, it is implementation feasible because only sending part needs to be modified.**

***Index Terms:* Slow-start, transmission control protocol (TCP), TCP-Vegas.**

I. INTRODUCTION

With the rapid growth of Internet population and the intensive usage of TCP/IP protocol suite, the transmission control protocol (TCP) congestion control algorithm has become a key factor influencing the performance and behavior of the Internet. Several studies have reported that TCP Vegas [1], [2] provides better performance than TCP Reno with respect to overall network utilization, stability, fairness, throughput, packet loss, and burstiness. Since TCP Vegas uses the difference between the expected and actual flow rates to infer the congestion window adjustment policy from throughput measurements, it usually reduces the sending rate before the connection actually experiences a packet loss.

TCP Vegas successfully detects network congestion in the early stage; however, the burstiness during slow-start phase causes Vegas to change from slow-start phase to congestion-avoidance phase too early, especially in a large-bandwidth link with long-delay. Since the sender has no prior knowledge regarding the available bandwidth on the networks, this leads to the abrupt transition of congestion window with exponential growth and transmission of highly bursty traffic from the source, and it in turn would cause buffer overflow at the bottleneck link during the slow-start phase [3]–[8]. To solve this problem, certain new techniques, schemes, or refinements of slow-start phase of TCP Vegas have been proposed. In these earlier works, there are three ways to smooth out the burstiness or postpone the time instant of changing from slow-start phase to congestion-avoidance phase. The first approach is selecting γ dynamically to suit various kinds of bandwidth-delay product (BDP) networks [3]; however, it needs to estimate the available bandwidth of the network at the steady state. In addition, to estimate the available bandwidth based on end-to-end congestion avoidance mechanism on a global internet is difficult. Another way is to set the maximum slow-start threshold to avoid buffer overflow and limit the sending rate [4], [5], but this not only reduces the throughput of a sender but also sets the maximum slow-start threshold to a fixed value, 64 kbytes. In a large BDP network, this value may be so small that causes Vegas switching to congestion-avoidance phase early. On the other hand, this fixed slow-start threshold may be of no use in a small BDP network. The last method uses a smooth slow-start algorithm to reduce burst data transfer [6]. However, it uses 200 msec timer interrupt to control data transfer and only fits some network topology. Furthermore, using timer interrupt increases the overhead of the operating system. Besides, some authors consider several network models and mathematically prove the exponential increase during slow-start causing the burstiness in the communication networks [7], [8]. Overall, these works just describe the problems and try to characterize them into models, but the burstiness problem is still not solved.

In this paper, we propose a modification of TCP Vegas, called Gallop-Vegas. During the slow-start phase, Gallop-Vegas changes the increase manner of congestion window with a growth, whose increase speed is between exponential increase and linear increase; therefore, a smooth transmission for a sender can be achieved and no burstiness problem may happen. In addition, Gallop-Vegas tries to detect incipient congestion by comparing the measured throughput to the notion of expected throughput. This congestion detection mechanism is same as that in TCP Vegas, except that it calculates every round-trip time (RTT) instead of every other RTT. The congestion window is increased only if these two values are close enough, and the increment of congestion window varies according to the current status of the network.

When TCP Vegas changes from slow-start phase to congestion-avoidance phase, it decreases the congestion window by one-eighth. This may be suitable for small-bandwidth links, but in large-bandwidth links it will be slow in reaching the available bandwidth. We make a little modification for slow-start phase while similar congestion detection mechanism is still applied. The implementation of Gallop-Vegas is simple. Only the sending part requires modifications, thus it facilitates incre-

Manuscript received December 28, 2004; approved for publication by Song Chong, Division III Editor, July 26, 2006.

C.-Y. Ho and Y.-C. Chen are with the Department of Computer Science, National Chiao Tung University, Hsinchu City, Taiwan, email: {cyho, ycchen}@csie.nctu.edu.tw.

Y.-C. Chan is with the Department of Computer Science and Information Engineering, National Changhua University of Education, Changhua City, Taiwan, email: ycchan@cc.ncue.edu.tw.

mental deployment in today's Internet. Furthermore, the analysis, extensive simulation results, and measurements on the Internet reveal that Gallop-Vegas is more efficient than TCP Vegas.

The remainder of this paper is organized as follows. Section II describes the details in slow-start phase of TCP Vegas. Section III expresses the algorithm and ideas of Gallop-Vegas as well as the possible probe strategies. The mathematical analysis of Gallop-Vegas is presented in Section IV. Section V demonstrates the simulation results and measurements on the Internet. Finally, Section VI concludes the paper.

II. SLOW-START OF TCP VEGAS

The bandwidth estimation scheme in TCP Vegas [1] is proactive because it tries to avoid rather than react to congestion. Vegas uses the difference in the expected and actual flow rates to estimate the available bandwidth of the network. When the network is not congested, the actual rate is close to the expected flow rate. However, if the actual rate was much smaller than the expected rate, it indicates that the buffer space in the network is filling up and that the network is approaching a congested state. This difference in flow rates can be calculated as $Diff = Expected - Actual$, where $Expected$ and $Actual$ are the expected and actual rates, respectively. If d denotes the minimum-observed round-trip time (also known as $BaseRTT$), D denotes the actual RTT of a packet, and W denotes the congestion window size, then $Expected = W/d$ and $Actual = W/D$. The estimated backlog of packets in the network queues can then be computed as

$$\Delta = (Expected - Actual) \times BaseRTT = W \times \frac{(D-d)}{D}. \quad (1)$$

Similar to Reno, Vegas uses a slow-start mechanism that allows a connection to quickly ramp up to the available bandwidth. However, unlike Reno, to ensure that the sending rate does not increase too fast to congest the network during the slow start, Vegas doubles the congestion window size only *every other RTT*. In addition, every other RTT, Vegas calculates the difference in the flow rates ($Diff$) and Δ as given in (1). When $\Delta > \gamma$ (whose default is 1), Vegas leaves the slow-start phase, decreases its congestion window size by 1/8 and enters the congestion-avoidance phase.

The fundamental problem in the slow-start algorithm of Vegas is that doubling its sending rate in short interval causes Δ bias [3]. This characteristic of slow-start may lead to early transition to congestion-avoidance phase and cause severe performance degradation. Under the TCP/IP architecture, it is difficult to estimate the exact available bandwidth of bottleneck link along the end-to-end path. Although Vegas has the burst avoidance mechanism that limits the number of segments to be sent at one time (that is, back-to-back) to three segments [2], it still causes burstiness in sending packets. According to the work [6], in short-delay networks, the optimum window size is small and there is no significant difference between RTT and actual data transfer time. Thus, burst data transfer occurs un-apparently in short-delay networks. But in long-delay networks, slow-start leads to burst data transfer which causes congestion and leads to much lower window size than optimum one. This in turn results in short slow-start phase and long congestion-avoidance phase.

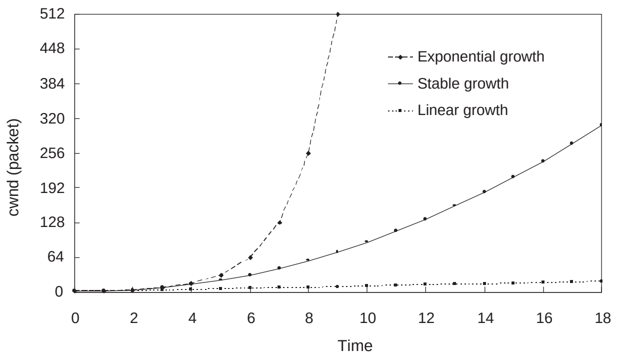

Fig. 1. The growth of congestion window size.

III. GALLOP-VEGAS

A. Motivation

When a source wants to send a big file as a movie data to a destination, the fundamental problem of TCP Vegas in the slow-start phase is that due to exponential growth, the congestion window size increases too quickly. This causes bias of Δ (given in (1)) and performance degradation, as well as leads to long congestion-avoidance phase. On the other hand, if the data whose size is not large is needed to transmit, the TCP Vegas may not enter the congestion-avoidance phase. However, TCP Vegas doubles the congestion window size only every other RTT that may spend more RTTs to finish the communication. Furthermore, in previous works, they either avoid burstiness with some limiting factors or address the problems in the slow-start phase, but they did not provide a valid method for various communication networks. Shortening the duration of raising the transmission rate to the available bandwidth could improve the sender's throughput for different communication networks, especially for a short life-cycle connection. For these reasons, we propose a modification to the slow-start phase, called Gallop-Vegas.[1]

B. The Scheme of Gallop-Vegas

In Gallop-Vegas, we do not increase congestion window size in the first RTT, which is either the beginning of a connection or after a retransmission timeout, because we have no idea about the available bandwidth for this connection. After the second RTT, we start to increase the congestion window with a rate between exponential growth and linear growth. We call it *stable growth*[2] as shown in Fig. 1.

When a sending source increases (or decreases) its congestion window at the n-th RTT, the influence to the network can be detected at the $(n+2)$-th RTT. As a result, Vegas calculates the Δ and doubles the congestion window (if it is possible) every other RTT. However, doubling the congestion window size may cause the traffic burstiness in the network, and therefore

[1]Gallop-Vegas is a variant of TCP Vegas, so the detailed description of parameters such as Δ, γ, α, and β can be found in [1] and [2].

[2]We will prove or disprove if stable growth is the optimal among many possible growth rates under different network conditions.

Table 1. Value, state, and corresponding motivation at the last RTT.

Value	State	CM at the last RTT
0	(1) $\Delta < \gamma$	'*incr*' is increased by one.
	(2) $\gamma \leq \Delta < \beta$	Do not do any action.
1	$\gamma \leq \Delta < \beta$	'*incr*' is decreased by one half
2	$\beta < \Delta$	The congestion window size is decreased by the sum of '*incr*' and surplus of queue ($\Delta - \beta$).

leads to the congestion-avoidance phase too early. To avoid this phenomenon, we calculate the Δ and increase the congestion window with *stable growth* (if it is possible) every RTT. This approach for increasing congestion window is more efficient comparing with Vegas, and the effect can be observed by analysis, simulation results, and measurements on the Internet.

Let '*maxincr*' be a dynamic value representing the maximum value of the congestion window increment, and '*incr*' be the current window increment, with value 0 at the beginning of a connection or after a retransmission timeout, and no bigger than '*maxincr*.' In order to record the comparison result of Δ with γ, or with β (if Δ is not smaller than γ) at the last RTT, we create a parameter '*status*,' whose default value is 0. Table 1 shows three sets of value, state, and corresponding motivation (CM) at the last RTT of '*status*.' We choose β (whose default is 3) to compare with Δ because the router is allowed to queue a number, which is between α (whose default is 1) and β, of packets in Vegas.

In the slow-start phase, we let '*maxincr*' be the current congestion window size first, and then compare Δ with γ. When Δ is smaller than γ, we add the value of '*incr*' to the current congestion window size, then '*incr*' is increased by one until it is no smaller than '*maxincr*.' At last, we set '*status*' to zero to represent this state. The parameter '*incr*' is increased step by step as long as there is enough bandwidth in the network. The idea is that if the increment was successful, it might be the case that there is enough bandwidth and it is worthwhile to move to a more aggressive increasing strategy. However, to ensure that the congestion window will not be increased too fast, Gallop-Vegas can at most double the size of congestion window for every estimation of $\Delta < \gamma$. While Δ is no smaller than γ, we compare Δ with β to adjust those parameters.

While Δ is smaller than β, the addition of congestion window size is still '*incr*.' Nevertheless, we modify '*incr*' according to the state, which is represented by the parameter '*status*,' at the last RTT before increasing the congestion window. Since it exceeds the lower bound of packets queued in the router(s) for a connection, it should slow down the increment. For another reason, it may exceed the available bandwidth without slowing down the increment. If the '*status*' is zero, we have to do three steps. First, '*incr*' is decreased by one half in order to slow down the growth. Second, if '*incr*' is no bigger than one, set it to one, and slow start threshold ($ssthresh$) to two in order to transit to the congestion-avoidance phase at the next RTT. Finally, '*status*' is marked as one to represent entering this state at either the first time or the odd number of times. On the other hand, if '*status*' is one, we just change it to zero when continuously getting into this state at the even number of times. Since we decreased the '*incr*' by one half at the last RTT, we do not know the influence of network yet. If we still decreased the '*incr*' by one half at this RTT again, we may get just the opposite. In other words, it may be too quickly to decrease '*incr*' by one half while Δ is between γ and β.

If Δ is no smaller than β, we cut the congestion window size down by the sum of the increment at the last RTT and surplus packets of queue, which is $\Delta - \beta$. Then we set '*status*' to two for avoiding repeatedly decreasing the congestion window size when changing to the congestion-avoidance phase at the first time. We call the increasing of congestion window (the increment of '*incr*') *stable growth* here after.

We only change one action of Vegas when it gets into congestion-avoidance phase at the first time. If '*status*' is two, we just set '*status*' to zero without changing the congestion window because we don't know the influence of network at the last RTT yet. Otherwise, we perform the same action as in Vegas. In retransmission-timeout phase, we reset these three parameters to their default values.

In summary, Vegas sends two packets back-to-back when it receives one acknowledgement (ACK). This may cause bursty traffic if a lot of ACKs return to the sender consecutively. This may cause Vegas turning to the congestion-avoidance phase early and then congestion window grows up slowly. On the other hand, Gallop-Vegas transmits one packets when receiving one ACK, and it sends an extra packet to increase the congestion window size after getting two or more ACKs. Through this method, we smooth out the burstiness without using timer. Since comparing with congestion window, the increment is always small, therefore Gallop-Vegas reduces the burstiness in transmission and achieves a long slow-start phase. Thus, the throughput of Gallop-Vegas grow up faster and is much larger than Vegas.

C. Pseudo Code of Gallop-Vegas

The following pseudo code represents the aforementioned statements regarding the *stable growth*.

```
In slow-start phase,
   maxincr = cwnd;
   if ($\Delta > \gamma$)
       if ($\Delta >= \beta$)
             cwnd $-=$ (last increment of cwnd $+\Delta - \beta$);
             ssthresh = 2;
             if (cwnd < 2) cwnd = 2;
             status = 2;
       else
             if (status == 0)
                 incr = int(incr/2);
                 if (incr <= 1)
                   ssthresh = 2;
                   incr = 1;
                 status = 1;
             else status = 0;
             cwnd += incr;
   else
       cwnd += incr;
       if (incr < maxincr) incr += 1;
```

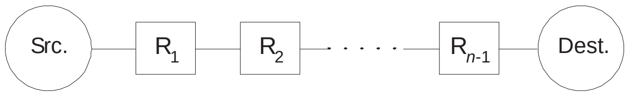

Fig. 2. Network topology for analyses.

```
        status = 0;
```

where cwnd is the congestion window size, and ssthresh is the slow-start threshold.

```
In congestion-avoidance phase,
    if (status ! = 2)
        Do the motion of original Vegas
    else
        status = 0;
In retransmission-timeout phase,
    maxincr = 2; (initial congestion window size)
    incr = 0;
    status = 0;
```

IV. ANALYSIS

In this section, we present behavior analyses of both Vegas and Gallop-Vegas. A simple case is considered when a single connection tries to fill up an empty network with N links connecting the source and the destination. Fig. 2 shows the network topology for analyses. We denote the transmission rate of N links (in packets/s) as X_i, $i = 1, \cdots, N$, and the total round-trip propagation delay of the route path (in seconds) as τ. Similar to the work in [3], we assume that there is one bottleneck in the route path and $X_1 \leq X_2 \leq \cdots \leq X_N$. Since X_1 is the smallest transmission rate (i.e., link 1 behaves as the bottleneck link), we let μ be equal to X_1. The un-congested BDP of this network is then given by μd where

$$d = \tau + (1+a)\sum_{i=1}^{N} \frac{1}{X_i} \tag{2}$$

with *a* being the ACK size relative to the data packet size. Without loss of generality, we assume that *a* is much smaller than the data packet size, so we use 1 to approximate 1+*a* (i.e., $d = \tau + \sum_{i=1}^{N} \frac{1}{X_i}$).

Throughout our analyses, we assume a fluid model and the source always has a packet to transmit, and the buffer sizes in routers are large enough so that packet loss can be ignored. Moreover, the i-th RTT starts with the transmission of W_i packets where W_i is the size of congestion window in this RTT. The i-th RTT ends when the source receives the ACK of the first packet in this RTT, then the source starts transmitting a new packet of the next RTT. Suppose that there is no congestion in ACK path. The congestion window size is named 'conspicuous window size' when Δ is no smaller than γ at the first time. (i.e., Vegas leaves its slow-start phase. Gallop-Vegas changes its way of increasing the congestion window size.) In the following two subsections, we derive mathematically the conspicuous window size for Vegas and Gallop-Vegas. Then, we ascertain the better transient performance for Gallop-Vegas by examples and simulations.

Table 2. The congestion window size and the increase amount for Vegas at the i-th RTT.

i-th	0	1	2	3	4	5	6	7	8	9	10	$\cdots$
W_i	2	2	4	4	8	8	16	16	32	32	64	$\cdots$
z_i	0	2	0	4	0	8	0	16	0	32	0	$\cdots$

Table 3. The congestion window size and the increase amount for Gallop-Vegas at the i-th RTT.

i-th	0	1	2	3	4	5	6	7	8	9	10	$\cdots$
W_i	2	2	3	5	8	12	17	23	30	38	47	$\cdots$
z_i	0	1	2	3	4	5	6	7	8	9	10	$\cdots$

A. TCP Vegas

In this subsection, we derive the conspicuous window size for Vegas. Table 2 shows the value of W_i and z_i at the i-th RTT if Δ is smaller than γ, where z_i is the increase amount at the i-th RTT and

$$\begin{cases} W_i = 2^{\frac{i}{2}+1}, z_i = 0, & \text{if } i \text{ is even or 0} \\ W_i = z_i = 2^{\frac{i+1}{2}}, & \text{if } i \text{ is odd.} \end{cases}$$

Vegas doubles its congestion window every other RTT. Assuming that W_l is the conspicuous window size, where l is an integer in Table 2. According to the work [3], the $BaseRTT$, newly measured RTT, and the conspicuous window size W_l from which Vegas gets out of the slow-start phase are given by d,

$$D = d + \frac{W_l}{2\mu} \tag{3}$$

and

$$W_l \geq \frac{1+\sqrt{1+8\mu d}}{2} \tag{4}$$

respectively. In an actual Vegas implementation, D is the smoothed RTT [2] rather than the RTT of a specific packet. Thus, D for the last packet is the average of the actual RTTs of all packets in the same round, i.e., $D = d + W_l/4\mu$, rather than $D = d + W_l/2\mu$, as given above. By using the smoothed RTT, we have

$$W_l \geq \frac{1+\sqrt{1+16\mu d}}{2} \tag{5}$$

and then Vegas changes to the congestion-avoidance phase in the l-th RTT, where $l \geq 2\lg(1+\sqrt{1+16\mu d}) - 3$ because l is odd. Then, it will take $\lceil \mu d - 7W_l/8 \rceil$ RTTs to the available bandwidth. In other words, at the

$$(l + \lceil \mu d - 7W_l/8 \rceil)\text{-th} \tag{6}$$

RTT, TCP Vegas attains the available bandwidth.

B. Gallop-Vegas

In this subsection, we derive the conspicuous window size for Gallop-Vegas. Table 3 shows the value of W_i and z_i at the i-th RTT when Δ is smaller than γ, where

$$W_i = \frac{i^2 - i + 4}{2}, \quad z_i = i, \quad i \in N \text{ or } 0. \tag{7}$$

Gallop-Vegas increases its congestion window with *stable growth* every RTT. Moreover, with *stable growth*, the extra packet[3] will be transmitted to the network when two or more ACKs of the previous round are received. For example, at the fifth RTT in Table 3, Gallop-Vegas sends an extra packet when it receives the third ACK, then it transmits another extra packet after two ACKs (i.e., the fifth ACK is received). Since in this RTT, Gallop-Vegas increases $\frac{5}{12}$ packet to the congestion window size whenever an ACK of the previous round is received, and sends an extra packet to the network when the additional value is no smaller than one (as $\frac{5}{12} \times 3 = \frac{15}{12} > 1$). In the light of the work [3], the spacing between each ACK of the previous round is $1/\mu$ seconds because μ is the smallest transmission rate (and we assume that there is no congestion along the ACK path). Assuming that W_k is the conspicuous window size, then in the k-th RTT, the last packet will see $z_k - 1$ $(= k-1)$ packets waiting ahead of it in the sender queue. However, for the queues at other nodes along the connection, the last packet will see no packet from the same connection in the queues because $1/X_1 \geq 1/X_2 \geq \cdots \geq 1/X_N$. Therefore, this last packet experiences the highest RTT. The $BaseRTT$ and newly measured RTT are given by d and

$$D = d + \frac{z_k - 1}{\mu} \tag{8}$$

respectively. By combining (8) and (1), Gallop-Vegas will change its way of increase if

$$W_k \frac{(z_k - 1)/\mu}{(z_k - 1)/\mu + d} > \gamma \tag{9}$$

where $\gamma = 1$. Then, we could get the following formula

$$W_k > \frac{\mu}{z_k - 1} d + \gamma = \frac{\mu}{z_k - 1} d + 1. \tag{10}$$

By combining (10) and (7), we could get the following equation:

$$(k-1)(k^2 - k + 2) > 2\mu d. \tag{11}$$

We could get k by Cardan's formula [9], [10] because (11) is a cubic equation, which is the closed-form solution for the roots of a cubic polynomial.

$$\begin{aligned} k \quad \geq \quad & \Bigg\lceil \sqrt[3]{\mu d + \sqrt{\frac{125}{729} + \mu^2 d^2}} \\ + \quad & \sqrt[3]{\mu d - \sqrt{\frac{125}{729} + \mu^2 d^2}} + \frac{2}{3} \Bigg\rceil. \end{aligned} \tag{12}$$

Then, the conspicuous window size that Gallop-Vegas changes its way of increase is given by using the value of k to (7).

In an actual Gallop-Vegas implementation, D is the smoothed RTT [2] rather than the RTT of a packet. This D for the last packet is the average of the actual RTTs of all packets in the same round, i.e., $D = (d + z_k - 1)/2\mu$, rather than $D = (d + z_{k-1})/\mu$, as given above. By using the smoothed RTT, we have

$$\begin{aligned} k \quad \geq \quad & \Bigg\lceil \sqrt[3]{2\mu d + \sqrt{\frac{125}{729} + 4\mu^2 d^2}} \\ + \quad & \sqrt[3]{2\mu d - \sqrt{\frac{125}{729} + 4\mu^2 d^2}} + \frac{2}{3} \Bigg\rceil. \end{aligned} \tag{13}$$

Then, as mentioned before, the addition of z is decreased by a half as long as Δ is bigger than γ. Finally, Gallop-Vegas will leave the slow-start phase when Δ is no smaller than β. In other words, in this time, the congestion window W is close to the maximum window size and three packets will be queued in the router. Thus, we will get

$$D = d + \frac{\beta}{\mu}. \tag{14}$$

By combining (1) and (14), Gallop-Vegas will stop its slow-start phase if

$$W \frac{\beta/\mu}{\beta/\mu + d} \geq \beta = 3. \tag{15}$$

By solving (15) for W, the congestion window size W at which Gallop-Vegas stops its slow-start phase is given by

$$W \geq \mu d + 3. \tag{16}$$

Furthermore, the following is the behavior of Gallop-Vegas from the k-th RTT to the increment being equal to 1 at the first time. At the k-th RTT, the increment of congestion window is $k/2$. Then, Gallop-Vegas adds the increment by 1 every other RTT until the value of the increment is k. In addition, it will take k RTTs. Therefore, the congestion window size is $W_k + (3/4)k^2$ at the $(2k)$-th RTT since $k/2 + k/2 + (k/2+1) + (k/2+1) + (k/2+2) + \cdots + k + k \approx (3/4)k^2$ [11]. Afterward, Gallop-Vegas repeats above steps to adjust the increment value with $k/4, k/8, \cdots, 1$ because when Δ is bigger than γ, the increment will be halved. Therefore, the total amount of this time period and the total increment of the congestion window size are

$$2\left(\frac{k}{2} + \frac{k}{4} + \frac{k}{8} + \cdots + 1\right) = k \sum_{n=1}^{\lfloor \lg k \rfloor} \frac{1}{2^{n-1}} \tag{17}$$

and

$$\frac{3}{4}k^2 + \frac{3}{8}k^2 + \cdots + \frac{3}{2^{\lfloor \lg k \rfloor + 1}} k^2 = \frac{3}{4}k^2 \sum_{n=1}^{\lfloor \lg k \rfloor} \frac{1}{2^{n-1}} \tag{18}$$

respectively. Then, Gallop-Vegas increases the congestion window by one every RTT before it reaches W given in (16). In conclusion, Gallop-Vegas attains the available bandwidth at

$$(\lceil k + k \sum_{n=1}^{\lfloor \lg k \rfloor} \frac{1}{2^{n-1}} + \lceil W - (W_k + \frac{3}{4}k^2 \sum_{n=1}^{\lfloor \lg k \rfloor} \frac{1}{2^{n-1}})\rceil\rceil)\text{-th} \tag{19}$$

RTT according to (13) and (16)–(18).

[3]When a source host receives an ACK of the previous round, it transmits two packets to the network. The difference between two packets and one packet is called the extra packet. For example, in Table 3, the source transmits two packets when it receives the second ACK at the fourth RTT.

C. Analysis with Two Examples

We use two examples to quantify our analyses, show that the throughput of Gallop-Vegas is more efficient than TCP Vegas, and Gallop-Vegas uses less time than TCP Vegas to reach the available bandwidth.

Example 1: $\mu = 6250$, $d = 0.1$, and available window size is $626(625 + \alpha) \sim 628(625 + \beta)$.

In Gallop-Vegas, we will get $k \geq \lceil 16.242 \rceil = 17$ by (13) and conspicuous window size $W_k = 138$ by (7). According to (19), at 108th RTT ($= 10.8$ second), Gallop-Vegas will achieve the available bandwidth. On the other hand, in Vegas, we will get conspicuous window size $W_l \geq 50.502$ by (5). $W_l \geq 50.502$ holds when $W_l = 64$ at the 10th RTT, because W_l should be to the powers of 2. Then Vegas changes to the congestion-avoidance phase and increases the congestion window size linearly. Based on (6), TCP Vegas will attain the available bandwidth at 579th RTT ($= 57.9$ second). We could see that Gallop-Vegas only need to spend 10.8 seconds, which is about one sixth of 57.9 seconds that TCP Vegas spends, to arrive the available window size. In addition, the time 10.8 seconds for Gallop-Vegas and 57.9 seconds for TCP Vegas are near to the first simulation result with same parameters in the Figs. 4 and 5.

Example 2: $\mu = 3125$, $d = 0.05$, and available window size is $157(156 + \alpha) \sim 159(156 + \beta)$.

Similar to Example 1, we will get conspicuous window size $W_k = 38$ at the 9th RTT in Gallop-Vegas, and conspicuous window size $W_l = 32$ at the 8th RTT in Vegas. Then Vegas changes to the congestion-avoidance phase and adds the congestion window linearly. The time for Gallop-Vegas and TCP Vegas to achieve the available bandwidth are 26 RTTs ($= 1.3$ seconds) and 136 RTTs ($= 6.8$ seconds), by (6) and (19), respectively. We could see that Gallop-Vegas only need to spend 1.3 seconds, which is about one fifth of 6.8 seconds that TCP Vegas spends, to arrive the available window size. Moreover, the convergence time[4] of Gallop-Vegas (26) and TCP Vegas (136) are near to the third simulation result with same parameters in the Fig. 8.

From the above two examples, we could see that the congestion window size of Gallop-Vegas is no smaller than Vegas when $\Delta > \gamma$ holds at the first time. Then Gallop-Vegas keeps in the slow-start phase and increases its congestion window size with *stable growth*. However, Vegas stops its slow-start phase and changes into the congestion-avoidance phase. So, Gallop-Vegas will spend less time than TCP Vegas to reach the available bandwidth and the throughput of Gallop-Vegas will be no smaller than Vegas. Therefore, the utilization of bandwidth in Gallop-Vegas is more efficient than that in TCP Vegas.

V. PERFORMANCE EVALUATION

A. The Simulation Setup

The simulation experiments are conducted using *ns2* [12], version 2.26, developed at Lawrence Berkeley National Laboratories (LBNL). Suppose that there is no packet loss in the simulation. The simulation network topology of one single link is shown in Fig. 3, where S1 represents a sender host, whose algorithm is either Vegas or Gallop-Vegas. The type of service used in our simulation is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets, which are 1000 bytes. R1 and R2 represent two finite-buffer gateways. The buffer size at each gateway is set to 100 packets. For the constant-load experiment, drop-tail gateways with FIFO service are assumed. The bandwidth of access links are 1Gbps, and propagation delays are 1ms.

[4]The convergence time can be interpreted as the effective number of window transmissions in the transient period since it indicates how many BaseRTTs are required to reach equilibrium. The detail description of the convergence time can be found in [3].

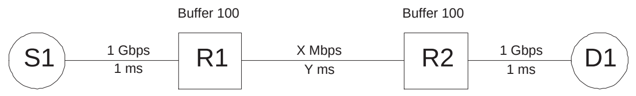

Fig. 3. Simulation topology used for Vegas or Gallop-Vegas experiments.

The bandwidth of connection link is X Mbps, where X is 1.5, 5, 10, 25, or 50, and propagation delay is Y ms, where Y is 3, 8, 23, 28, or 48. The combinations of X and Y generate 25 networks. Although we have used different bandwidths and propagation delays of connection links, only the simulation results of X = 50 and Y = 48 are presented here. Other values of X and Y will be shown in the figure of convergence time. We choose these values of communication network to represent small-bandwidth, large-bandwidth, short-delay, and long-delay, respectively. The sender uses the slow-start at the start of a connection, and/or after a retransmission timeout, and hence it features similar behavior during slow-start phase.

In following sections, we will show the simulation result, convergence time with different BDPs of communication networks, and ten senders with the same algorithm sharing a common bottleneck of 100 Mbps bandwidth and 48 ms propagation delay.

B. Simulation Results

We compare Gallop-Vegas with Vegas which uses two different parameter values, one with γ one, and the other with γ three (as β). It is because Gallop-Vegas changes from slow-start phase to congestion-avoidance phase when Δ is no smaller than β. In Fig. 3, X is 50 and Y is 48, it means that the bandwidth of bottleneck link is 50 Mbps (6250 packets/s), and the end-to-end propagation delay is 50 ms ($1+48+1 = 50$). Figs. 4 and 5 show the congestion window size and throughput between Vegas and Gallop-Vegas, respectively.

We can observe that the performance of Gallop-Vegas is better than Vegas. Both varieties of Vegas turn to the congestion-avoidance phase early, one is at 1.2 seconds and the other is at 1.5 seconds, and they increase congestion window through linear growth. They spend more than 50 seconds (which almost equals 500 RTTs) to reach the available bandwidth. On the other hand, Gallop-Vegas switches to congestion-avoidance when it reaches the available bandwidth at 11.4 seconds, and only spends 0.4 seconds (which approximately equals to 4 RTTs) to reach the real available bandwidth. The maximum number of queuing packets in these algorithms are almost the same.

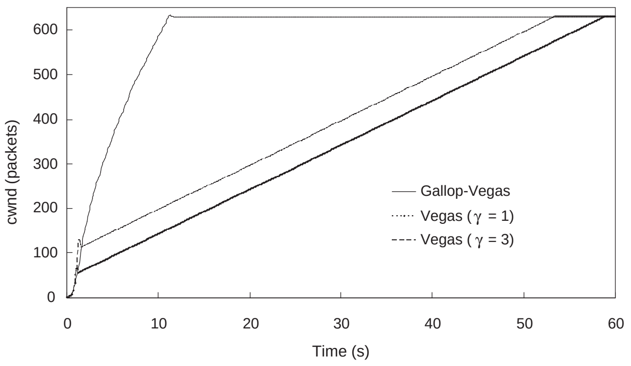

Fig. 4. Congestion window size comparison between Vegas and Gallop-Vegas with 50 Mbps bottleneck bandwidth, and 48 ms link propagation delay.

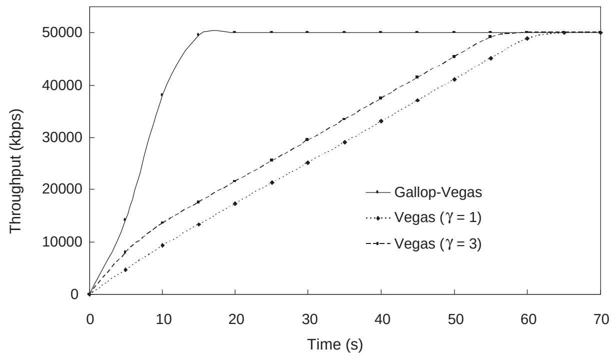

Fig. 5. Throughput comparison between Vegas and Gallop-Vegas with 50 Mbps bottleneck bandwidth, and 48 ms link propagation delay.

An interesting phenomenon in the simulation was observed. Both of Vegas lose packets but Gallop-Vegas does not when the buffer size of the router is decreased. There is an example with the router buffer size 30. The congestion window size and throughput between Vegas and Gallop-Vegas are described in Figs. 6 and 7, respectively. In this environment, the router R1 drops three packets of Vegas at 1.1 seconds because both of Vegas double the congestion window size from 32 to 64. This causes a bursty traffic to a router, which could not handle these packets in time. The same situation happens at 1.6 seconds, where router R1 drops sixteen packets. Since Vegas starts the fast retransmits procedure to redeem the lost packets at 1.1 seconds, after redeeming the lost packets, Vegas doubles the current window size continuously, this action causes the packet loss because of burstiness again. Since there are too many packets losses at this time, Vegas has to wait a retransmission timeout for a long time. On the other hand, Gallop-Vegas increases the congestion window size with *stable growth*, so it does not cause a large burstiness. It could increase congestion window size steadily during the slow-start phase.

Now, we use the convergence time [3] with different BDPs of communication networks to compare Gallop-Vegas with two varieties of Vegas. The result is shown in Fig. 8. We can see that the convergence time of Gallop-Vegas grows slowly (or linearly) while BDP increases quickly. However, the convergence time of both Vegas varieties climb very fast. The convergence time

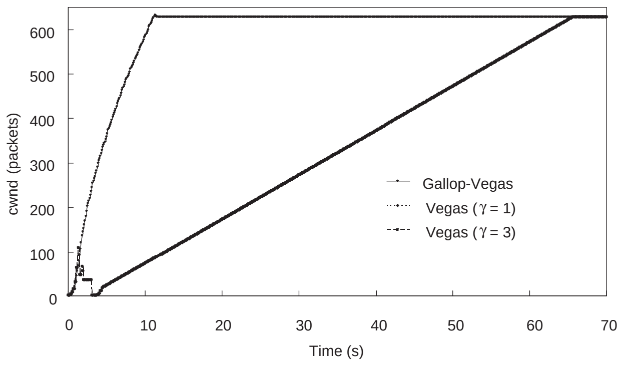

Fig. 6. Congestion window size of Vegas and Gallop-Vegas. There are packet lost in both of Vegas (buffer size $= 30$).

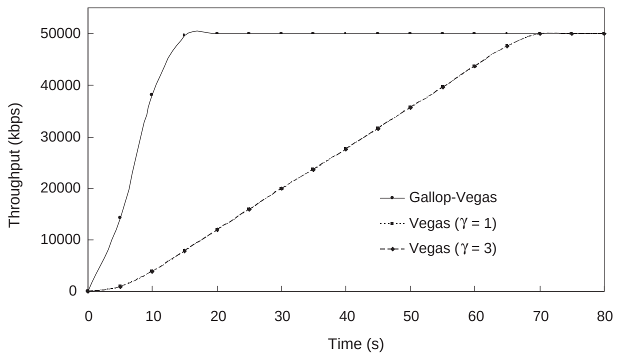

Fig. 7. Throughput of Vegas and Gallop-Vegas. There are packet lost in both of Vegas (buffer size $= 30$).

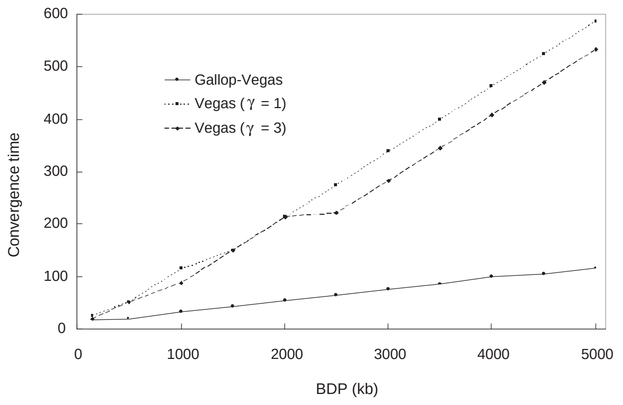

Fig. 8. The convergence time with different BDPs of communication networks.

of Vegas is more than five times of Gallop-Vegas at 5000 kb. We could conclude that Gallop-Vegas is as good as Vegas in the small BDP and much better than Vegas in the large BDP with the demonstration in Fig. 8. In addition, the difference in convergence time between Gallop-Vegas and TCP Vegas may be very large in high-speed and long-delay networks.

C. Multiple Senders in One Network

After comparing one sender in the same network topology, we compare the cases of multiple senders with the same algo-

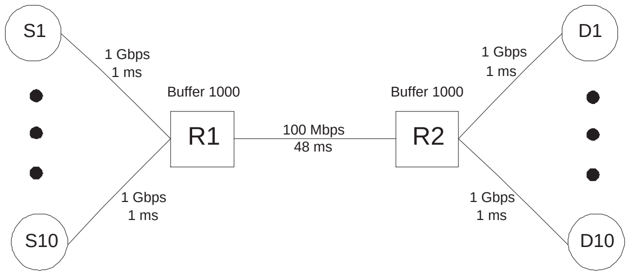

Fig. 9. Simulation topology with multiple sender used for Vegas or Gallop-Vegas experiments.

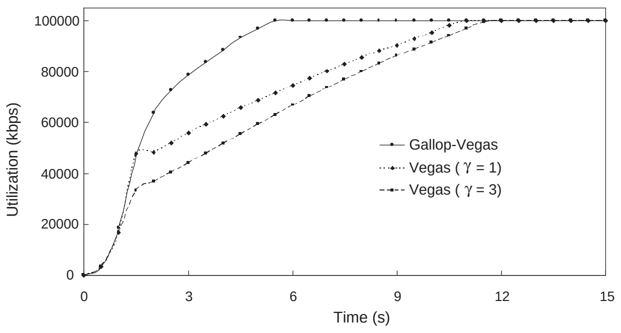

Fig. 10. The utilization of bottleneck link between Vegas and Gallop-Vegas.

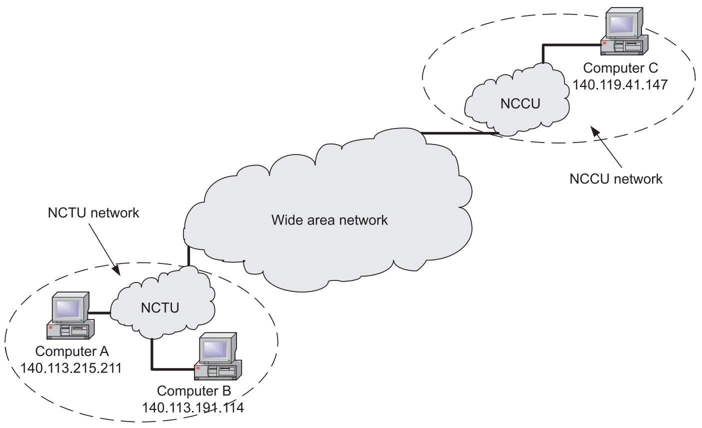

Fig. 11. The real test-bed network.

Table 4. The information of connections.

	Computer A – Computer B		Computer A – Computer C	
Hop	IP address	Delay (ms)	IP address	Delay (ms)
1	140.113.215.211	0	140.113.215.211	0
2	140.113.215.254	0.345	140.113.0.165	0.43
3	140.113.0.166	0.31	140.113.0.97	0.6
4	140.113.0.169	0.33	211.79.59.146	2.33
5	140.113.191.114	0.35	211.79.59.153	2.48
6	X	X	211.79.59.101	2.39
7	X	X	140.119.243.5	2.42
8	X	X	140.119.41.147	2.5

rithm of Gallop-Vegas and Vegas. The used simulation network topology is shown in Fig. 9, and the whole skeleton is same as that in Fig. 3. The difference between Figs. 3 and 9 is that there are more senders, bigger buffer size, and larger bottleneck bandwidth in Fig. 9.

The utilization of the bottleneck link is shown in Fig. 10. As seen in this figure, Gallop-Vegas utilizes the bandwidth of bottleneck link more efficiently than Vegas. One interesting observation is that doubling congestion window causes bursty traffic and makes all senders turn into congestion-avoidance phase at the same time when Vegas' γ is three. This phenomenon is fair to all senders when using TCP Vegas algorithm; however, the utilization of the available bandwidth is not efficient.

D. Internet Results

Now, we present measurements of TCP Vegas and Gallop-Vegas over the Internet. Specifically, we measured TCP Vegas and Gallop-Vegas transfers between the National Chiao Tung University (NCTU) and NCTU, and between NCTU and the National Cheng Chi University (NCCU). Fig. 11 shows the test-bed network. There are three computers in this test: Computer A, B, and C. Computer A, whose IP address is 140.113.215.211, is a sender with TCP Vegas or Gallop-Vegas algorithm in NCTU. Computer B with IP 140.113.191.114 in NCTU and Computer C with IP 140.119.41.147 in NCCU are receivers. In addition, the operating systems of three computers are all Linux with kernel 2.6.11 and all computers are equipped with a 450 MHz Pentium III processor, 256 MB RAM, and 100/10 M Ethernet card. Moreover, we only installed Gallop-Vegas scheme in Computer A and did not modify any mechanism running in Computer B and C since in Gallop-Vegas, only sending part needs to be modified.

The information of connections such as the number of hops between Computer A and Computer B, and between Computer A and Computer C is as shown in Table 4. The results are derived from a set of runs over a seven day period from November 21–28, 2005. Each run consists of a set of fourteen transfers from Computer A to Computer B (from NCTU to NCTU) and fourteen transfers from Computer A to Computer C (from NCTU to NCCU)—TCP Vegas sends 480 kB, 1 MB, 2 MB, 5 MB, 50 MB, 100 MB, and 400 MB, and Gallop-Vegas also sends 480 kB, 1 MB, 2 MB, 5 MB, 50 MB, 100 MB, and 400 MB. We inserted a 1 minute delay between each transfer in a run to give the network a chance to settle down, a run started approximately once every hour, and we shuffled the order of the transfers within each run.

Table 5 shows the results for all transfers, where TT and AT mean *transmission time* (second) and *average throughput* (Mb/s), respectively. We could see that Gallop-Vegas spends less time to complete the data transmission than TCP Vegas does whether the data size is large or small, and the RTT is long or short. For example, the average throughput of Gallop-Vegas (35.65 Mbps) to transmit 400 MB data from NCTU to NCCU is about 1.99 times higher than that of Vegas (17.92 Mbps). Our proposed mechanism spends 0.12 second, which is four fifths of the time TCP Vegas used, to finish 1 MB data transmission from NCTU to NCCU. Similarly, compared with TCP Vegas, the throughput improvement of Gallop-Vegas is kept between 3% and 44% when sending data from Computer A to Computer B.

Table 5. Internet results for all transfers.

	TCP Vegas				Gallop-Vegas			
	NCTU – NCTU		NCTU – NCCU		NCTU – NCTU		NCTU – NCCU	
data size	TT	AT	TT	AT	TT	AT	TT	AT
480 kB	0.05	73.85	0.17	22.15	0.048	78.33	0.16	24.00
1 MB	0.11	76.67	0.15	54.91	0.10	78.90	0.12	67.73
2 MB	0.23	78.25	0.22	81.46	0.18	100.79	0.19	93.40
5 MB	0.51	75.14	0.58	69.65	0.35	108.85	0.37	103.24
50 MB	7.02	56.69	6.99	56.94	5.60	71.07	5.67	70.19
100 MB	15.70	53.48	15.00	55.98	12.40	67.71	12.70	66.11
400 MB	59.10	55.49	183.00	17.92	45.10	72.71	92.00	35.65

VI. CONCLUSION

We propose and evaluate a new variant of the slow-start algorithm in TCP Vegas, called Gallop-Vegas, to reduce the burstiness, to raise the rate to the available bandwidth in shorter time, and to improve the start-up performance. In this work, we achieve more efficient throughput in the slow-start phase comparing with original TCP Vegas from analysis, simulation results, and measurements on the Internet. Although Gallop-Vegas is more suitable for large bandwidth or long-delay networks, it still increases transmit performance in small bandwidth or short-delay networks. Furthermore, the design of Gallop-Vegas is simple and implementation feasible on existing operating systems. Further work involves studying the performance and spatial characteristics analysis of this algorithm under a wider range of parameters, network topologies and real traffic traces, obtaining more accurate theoretical models and insights, and considering hardware implementation issues. Also, we will combine the improvement of congestion-avoidance phase and show the fairness in the future.

REFERENCES

[1] L. Brakmo and L. Peterson, “TCP Vegas: End-to-end congestion avoidance on a global Internet,” *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[2] U. Hengartner, J. Bolliger, and Th. Gross, “TCP Vegas revisited,” in *Proc. IEEE INFOCOM 2000*, Mar. 2000, pp. 1546–1555.

[3] S. Vanichpun and W. Feng, “On the trransient behavior of TCP Vegas,” in *Proc. IEEE ICCCN 2002*, Miami, Florida, Oct. 2002, pp. 504–508.

[4] H. Wang, H. Xin, D. S. Reeves, and K. G. Shin, “A simple refinement of slow-start of TCP congestion control,” in *Proc. ISCC 2000*, 3–6 July 2000, pp. 98–105.

[5] H. Wang and C. Williamson, “A New Scheme for TCP congestion control: Smooth-start and dynamic recovery,” in *Proc. MASCOTS’98*, Montreal, Canada, July 1998, pp. 69–75.

[6] Y. Nishida, “Smooth slow-start: Refining TCP slow-start for large-bandwidth with long-delay networks,” in *Proc. LCN’98*, Boston, Massachusetts, 11–14 Oct. 1998, pp. 52–60.

[7] D. Starobinski and M. Sidi, “Stochastically bounded burstiness for communication networks,” *IEEE Trans. Inform. Theory*, vol. 46, pp. 206–212, Jan. 2000.

[8] T. Konstantopoulos and V. Anantharam, “Optimal flow control schemes that regulate the burstiness of traffic,” *IEEE/ACM Trans. Networking*, vol. 3, pp. 423–432, Aug. 1995.

[9] R. Calinger, *Classics of Mathematics*, Moore Publishing, Oak Park, Illinois, pp. 235–237, 1982.

[10] J. Stillwell, *Mathematics and Its History*, Springer-Verlag, New York, p. 55, 1989.

[11] G. Chatranon, M. A. Labrador, and S. Banerjee, “A survey of TCP-friendly router-based AQM schemes,” *ELSEVIER Computer Commun.*, vol. 27, no. 15, pp. 1424–1440, Sept. 2004.

[12] The network simulator, available at http://www.isi.edu/nsnam/ns/.

Cheng-Yuan Ho is currently a Ph.D. candidate in computer science at National Chiao Tung University, Hsinchu City, Taiwan. He also works with the Wireless and Networking Group of Microsoft Research Asia, Beijing, China since December 2005. His research interests include the design, analysis, and modelling of the congestion control algorithms, high speed networking, QoS, and mobile and wireless networks.

Yi-Cheng Chan received his Ph.D. degree in computer science and information engineering from National Chiao Tung University, Taiwan in 2004. During 1994–2002, he worked at Accton Technology Corporation as a software engineer. He is currently an assistant professor in the department of computer science and information engineering of National Changhua University of Education, Changhua City, Taiwan. His research interests include design and analysis of network protocols, switch architecture, wireless networks, and active queue management.

Yaw-Chung Chen received his Ph.D. degree in computer science from Northwestern University, Evanston, Illionis, USA in 1987. During 1987–1990, he worked at AT&T Bell Laboratories. He joined department of computer science, National Chiao Tung University, Hsinchu, Taiwan in 1990 as an associate professor. Currently he is a professor and director of computer and network center. His research interests include multimedia communications, high speed networking, P2P network security, and wireless networks. He is an IEEE senior member.