World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:9, 2009

SMCC: Self-Managing Congestion Control Algorithm

Sh. Jamali, and A. Eftekhari

Abstract**—**Transmission control protocol (TCP) Vegas detects network congestion in the early stage and successfully prevents periodic packet loss that usually occurs in TCP Reno. It has been demonstrated that TCP Vegas outperforms TCP Reno in many aspects. However, TCP Vegas suffers several problems that affect its congestion avoidance mechanism. One of the most important weaknesses in TCP Vegas is that alpha and beta depend on a good expected throughput estimate, which as we have seen, depends on a good minimum RTT estimate. In order to make the system more robust alpha and beta must be made responsive to network conditions (they are currently chosen statically). This paper proposes a modified Vegas algorithm, which can be adjusted to present good performance compared to other transmission control protocols (TCPs). In order to do this, we use PSO algorithm to tune alpha and beta. The simulation results validate the advantages of the proposed algorithm in term of performance.

Keywords**—**Self-managing, Congestion control, TCP.

I. Introduction

A considerable amount of research has been carried out on the transmission control protocol (TCP) congestion avoidance mechanism in order to enhance the performance of networks in view of the fact that a slow start, fast retransmission and congestion control mechanism was proposed to effectively regulate the transmission [1]. The Vegas algorithm, introduced by Brakmo et al. as an alternative to TCP Reno [2], has higher performance in compare with other congestion control algorithms [3, 4 and 5]. The Vegas algorithm tries to adjust number of queued packets in an acceptable level i.e. between alpha and beta. Alpha and beta are the lower bound and upper bound of the desired queue length, that have been set statically in the Vegas algorithm (alpha=1 and beta=3). Unfortunately, these static values in the Vegas algorithm cannot lead to desired performance level. We propose a modified Vegas, namely SMCC, in which alpha and beta are tuned dynamically with respect to the network conditions. In this way we use PSO algorithm to find optimum points of alpha and beta for any network conditions. SMCC is a source algorithm that estimates network conditions by using measured RTTs and then adjusts alpha and beta to direct network to a high-performance state.

Sh. Jamali is with the University of Mohaghegh Ardabili, Ardabili, Ardabil (corresponding author to provide phone: +98451553917; e-mail: jamali@ iust.ac.ir).

A.Eftekhari is now MSc. Student at the Department of Computer, Islamic Azad University of Arak, Arak, Iran (e-mail:eftekhari@gmail.com).

The rest of the paper is organized as follows. In Section II, the Vegas algorithm is reviewed. In section III we present an introduction to PSO method. Section IV presents a methodology to apply PSO technique to design a congestion control algorithm and brings simulation results of the algorithm in ns-2 environment. Finally concluding remarks are reported in Section VI.

II. TCP Vegas

TCP-Vegas [6] is a delay-based transport protocol, which adjusts its congestion window according to the phases it performs and the gap $\triangle$ between the real and estimated sending rates. Three thresholds α, β and γ are defined in Vegas. TCP senders compare $\triangle$ with γ in slow start phase and with α and β in congestion avoidance phase to determine window adjustments. The estimation of the gap is done once per *RTT* period. Vegas sets *BaseRTT* to the minimum of all measured round trip times (*RTTs*) and computes the expected rate as *Expected =w/BaseRTT*, where *w* denotes window size. Let *RTTa* denote the average measured RTT, then Vegas calculates the Actual rate as *Actual=w/RTTa*. Then the gap between real and estimated sending rates is $\triangle$*=(Expected - Actual)*BaseRTT*.

In slow start phase, the congestion window is smaller than the slow start threshold W_{th}. When receiving a new ACK and $\triangle$ is less than γ, TCP senders increase *w* by one. If not, Vegas decreases the window size by a specific percentage *p*, sets W_{th} to be the reset value W^{r}_{th}, and switches to the congestion avoidance phase. Slow start is initiated at the very beginning or after a timeout event and ends when the window is larger than *Wth*. Vegas implements timeout mechanism by a coarse grain timer, which is checked once per 500 ms. The window update in slow start phase can be described as in (1).

$$w = \begin{cases} w + 1 & \text{if} \quad \Delta \prec \gamma \\ w \times (1 - p) & \text{if} \quad \Delta \geq \gamma \end{cases} \tag{1}$$

When TCP sender is in congestion avoidance phase and receives a new ACK, Vegas increases the window by *1/w* if $\triangle$ is less than α and decrements it by *1/w* if $\triangle$ is larger than β, and keeps it unchanged when $\triangle$ falls between α and β. Detail is shown in (2).

$$w = \begin{cases} w + \dfrac{1}{w} & if \quad \Delta \le \alpha \\ w & if \quad \alpha \prec \Delta \prec \beta \\ w - \dfrac{1}{w} & if \quad \Delta \ge \beta \end{cases} \tag{2}$$

Another two effective congestion avoidance mechanisms are fast retransmit and fast recovery, which retransmit lost data packets when receiving three duplicate ACKs without waiting for timeout. Besides coarse grain timeout mechanism, Vegas performs retransmission with another fine grain timer via time stamp included in packets, which allows Vegas to retransmit faster than other TCP variations with only coarse grain timer. More details about Vegas are refer to [6,7].

III. Particle Swarm Optimization Algorithm

Particle swarm optimization (PSO) algorithm was developed in 1995 by James Kennedy and Russell Eberhart, which is a robust stochastic optimization technique based on the movement and intelligence of swarms. It uses a number of particles that constitute a swarm moving around in the search space looking for the best solution [8, 9 and 10].

In PSO algorithm, a swarm consists of *m* particles, in which each particle is treated as a point in a *N*-dimensional space which adjusts its "flying" according to its own flying experience as well as the flying experience of other particles. Each particle uses velocity to determine the direction and value of its "flying", which follows the current optimum in a *N*-dimension space. The position and velocity of particle *i* at iteration *k* can be respectively expressed as $X_i(k)=[\ X_{i1}(k), X_{i2}(k),\dots, X_{iN}(k)]$ *and* $V_i(k)=[\ V_{i1}(k), V_{i2}(k),\dots, V_{iN}(k)]$. Each particle keeps track of its coordinates in the solution space which are associated with the **best solution** that has achieved so far by that particle. This value is called **personal best** $P_{i,best}$, which can be expressed as $P_{i,best}=[P_{i,1}(k), P_{i,2}(k),\dots, P_{i,N}(k)]$. Another best value that is tracked by the PSO is the best value obtained so far by any particle in the neighborhood of that particle. This value is called **global best** $P_{g,best}$, which can be expressed as $P\ g,best =[P_{g,1}(k), P_{g,2}(k),\dots, P_{g,N}(k)]$. The basic concept of PSO lies in accelerating each particle toward its personal best and the global best locations. The velocity and position of particle *i* at iteration *k+1* can be calculated according the following equations:

$$V_i\,(k{+}1) = \mathrm{w}V_i(k) + \mathrm{c}_1\mathrm{r}_1(P_{i,best}\,(k) - X_i(k)) + \mathrm{c}_2\mathrm{r}_2(P_{g,best}\,(k) - X_i(k))$$

$$X_i\,(k{+}1) = X_i\,(k) + V_i\,(k)$$

Where $X_i(k)$ and $V_i(k)$ are the position and velocity of the particle respectively, ω is the inertia weight, c_1 and c_2 are constants which determine the influence of the particle's best previous position $P_{i,best}\ (k)$ and the population's best previous position $P_{g\ ,\ best}(k)$. Parameters r_1 and r_2 are random numbers uniformly distributed within[0,1].

IV. Proposed Model: Theories, Design and Simulation Results

A. *Design Principles*

As we saw in section II, in the Vegas algorithm, α and β play important roles in the system performance. The Vegas algorithm tries to adjust number of queued packets between α and β. In original Vegas alpha and beta have been set statically (α=1 and β=3). It can be found that these static values in the Vegas algorithm cannot leads to desired level of performance. Therefore, we propose a modified Vegas, called SMCC algorithm in which α and β are tuned dynamically with respect to the network conditions such as RTT, bottleneck bandwidth, etc. In this way we use PSO algorithm to find optimum points of α and β for any network conditions. SMCC is a source algorithm that estimates network conditions by using measured RTTs and then determines values of alpha and beta which direct the network to its high-performance state.

B. *Packet-level Simulation*

To implement this protocol, we use the packet-level simulator *ns-2* [11], and modify the TCP Vegas module to implement SMCC algorithm. We present the simulation results to demonstrate the validity of our design. We demonstrate through simulations that SMCC outperforms TCP Vegas/RED in a typical network. Our simulations also show that SMCC drops fewer packets in compare with TCP Vegas. It dampens oscillations and smoothly converges to high utilization and small queue size.

Our simulation uses the topology in Fig. 1 and considers a ten-connection network that has a single bottleneck link. We suppose that all flows are long-lived, have the same end-to-end propagation delay and always are active. As follows we consider two simulation scenarios.

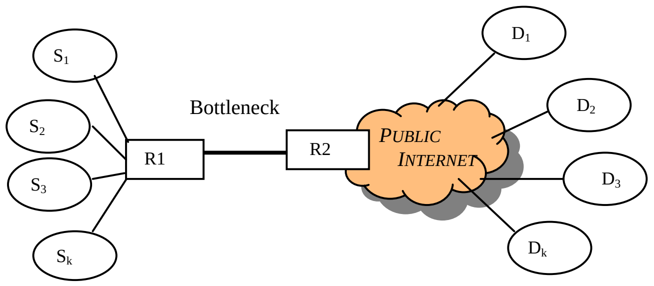


Fig. 1 Network Topology

1) *Scenario 1*

In this scenario bottleneck capacity is 20 Mbps and flows RTT has been considered 40 ms. All flows start at t=0 and continue till t=200 second. The simulation results of this congestion control system are shown in figures (2-4). In order to reference to the results of these figures, we note that:

1. Packet Drop: Fig. 2 and Table I show that SMCC behaves better than Vegas in term of dropped packets count. This comes from fine tuning of α and β that is performed by SMCC.

2. Queue evolution: As can be found in Fig. 3, while Vergas's queue is fluctuating between full and empty states, the SMCC's queue is converging to equilibrium size of 2 packets. This means that queuing delay and jitter are negligible for SMCC.

3. Utilization: According to the Fig. 4, and Table I after the startup transient of the sources, utilization of bottleneck link for SMCC remains always over the 98% that is good enough.

TABLE I
COMPARSION OF SMCC AND VEGAS IN A TYPICAL NETWORK

	Dropped Packets Count	Queue Size	Utilization
TCP Vegas	14329	1.58	91.94
SMCC	817	1.8	98.02

4. Stability and speed of convergence: As we can see in Fig. 2, Fig. 3, and Fig. 4, drop count, queue size and the utilization of SMCC have decreasing oscillation level and track stable behavior in compare with Vegas. Note that convergence is an important feature for any congestion control scheme.

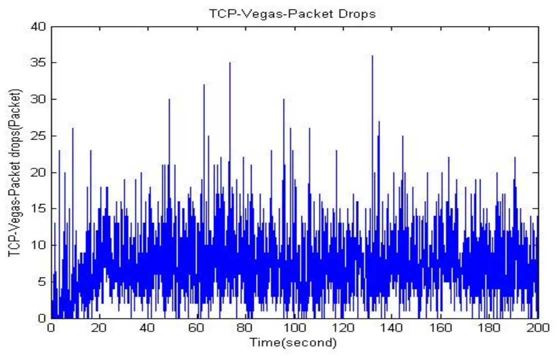
2) Scenario 2

As we know, high-speed networks with high bandwidth-delay (HBD) product present a unique environment where currently TCP may have a major challenge to its performance. In this scenario we consider a high bandwidth-delay product network, in which, bottleneck capacity is 100 Mbps and flows RTT is 40 ms. Figs. (5-7) and Table II show the simulation results for this scenario.

TABLE II
COMPARSION OF SMCC AND VEGAS IN A HBD NETWORK

	Dropped Packets Count	Queue Size	Utilization
TCP Vegas	822	0.14	70
SMCC	93	0.34	75.64

Simulation results demonstrate that as capacity increases, bottleneck utilization decreases significantly for both Vegas and SMCC, but SMCC presents better performance than TCP Vegas in term of bottleneck utilization, queue size and number of dropped packets. Note that this feature makes SMCC suitable for HBD environments.



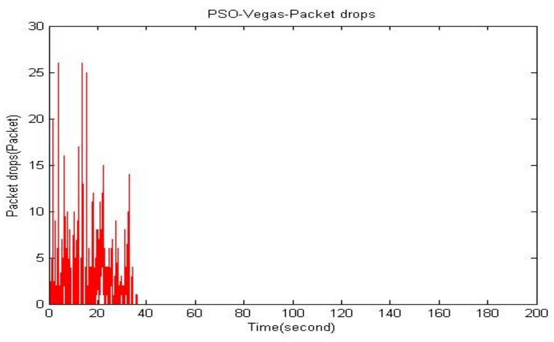


Fig. 2 SMCC drops fewer packets than Vegas

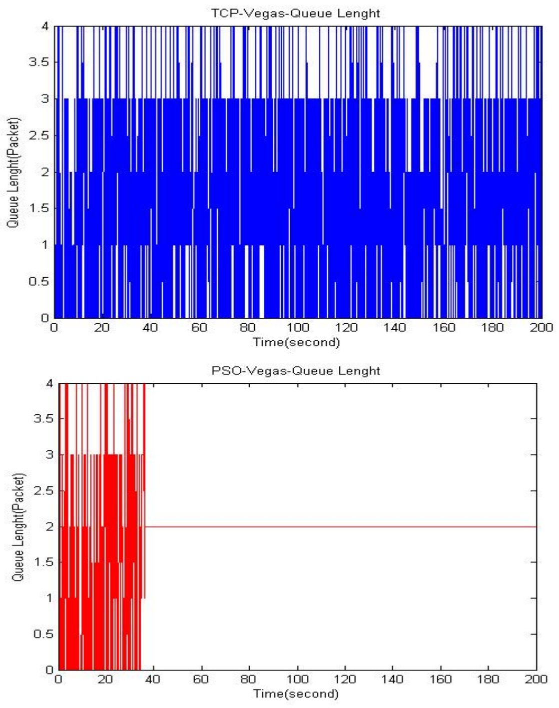


Fig. 3 SMCC's queue is shorter and more stable than Vergas queue

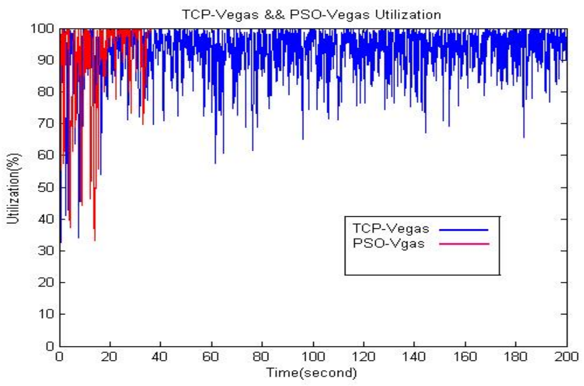


Fig. 4 Utilization of Vegas and SMCC

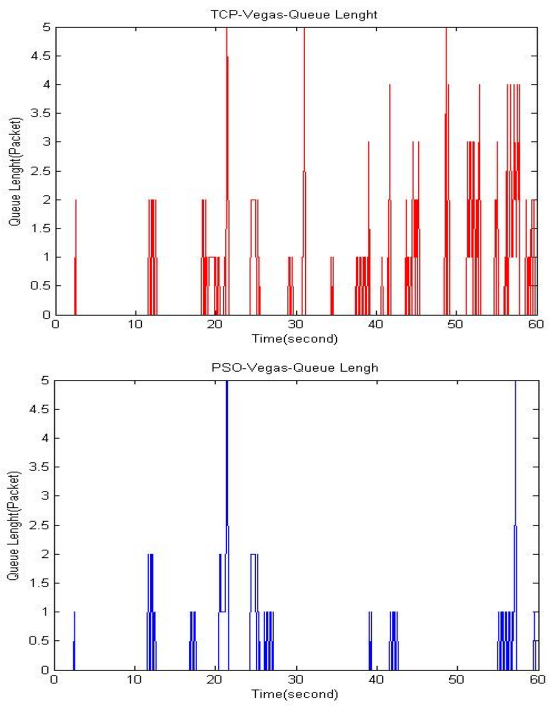


Fig. 5 SMCC drops fewer packets than Vegas in HBD environments

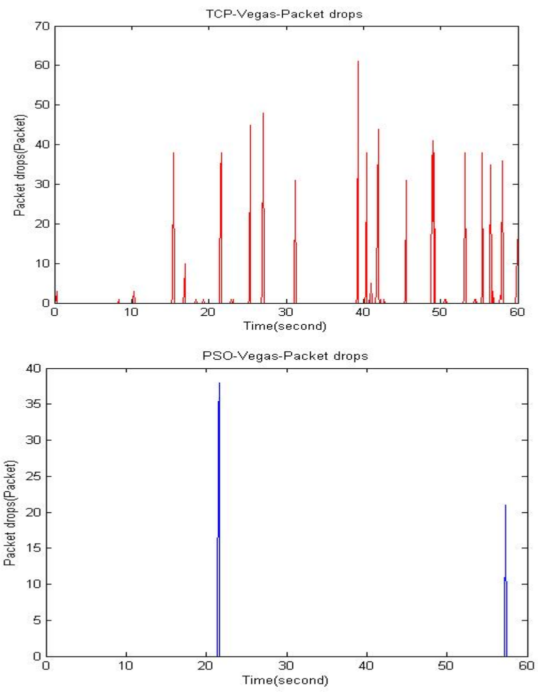


Fig. 6 SMCC's queue is shorter and more stable than Vergas

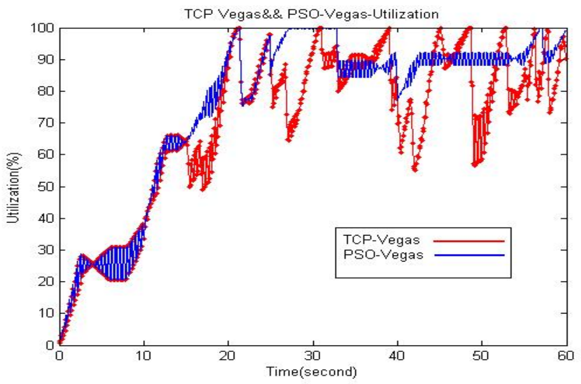


Fig. 7 Utilization of Vegas and SMCC

V. Conclusion

In this paper we have designed a bio-inspired congestion control algorithm. Toward this design, the following steps were considered: (1) formulating the congestion control problem as an optimization problem. (2) Choosing PSO technique as solver of the optimization problem. (3) Implementing of the model in ns-2 environment.

Simulation results show that the proposed algorithm is globally converging to its equilibrium and is high-performance in this equilibrium.

References

[1] V. Jacobson, Congestion avoidance and control, in: ACM SIGCOMM_88, Stanford, CA, 1988, pp. 314–329.

[2] L.S. Brakmo, L.L. Peterson, TCP Vegas: end to end congestion avoidance on a global Internet, IEEE J. Select. Areas Commun. 13 (8) (1995) 1465–1480.

[3] T. Bonald,Comparison of TCP Reno and TCP Vegas via fluid approximation, Tech. Rep. RR, 3563, 1998.

[4] J. Mo, R.J. La, V. Anantharam, J.C. Walrand, Analysis and comparison of TCP Reno and Vegas, in: INFOCOM, vol. 3, 1999, pp. 1556–1563.

[5] S.H. Low, L.L. Peterson, L. Wang, Understanding Vegas: a duality model, J. ACM 49 (2002) 207–235.

[6] E. H. Miller, "A note on reflector arrays (Periodical style—Accepted for publication)," *IEEE Trans. Antennas Propagat.*, to be published.

[7] J. Wang, "Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication)," *IEEE J. Quantum Electron.*, submitted for publication.

[8] R. Eberhart, and J. Kennedy, "A New Optimizer Using Particles Swarm Theory, Proc. Sixth International Symposium on Micro Machine and Human Science , IEEE Service Center, Piscataway, NJ, 39-43, 1995.

[9] J. Kennedy and R. Eberhart, Particle Swarm Optimization, IEEE International Conference on Neural Networks NJ, IV: 1942-1948, 1995.

[10] Y. Shi, R. Eberhart, Parameter Selection in Particle Swarm Optimization, The 7th Annual Conference on Evolutionary Programming, San Diego, USA, 1998.

[11] Ns-2.Network Simulator. http://www.isi.edu/nsnam/ns.