A NEW ALGORITHM TO ACHIEVE BANDWIDTH FAIRNESS: POWER-RED

Ching-Han Yu[1] Chun-Liang Lin[2,*] Ping-Min Hsu[2]

ABSTRACT

It's becoming one of the major research issues that how to handle data congestion and avoidance of that on the network in an efficient way has become a crucial issue because of bursting data traffic. There are a variety of congestion control strategies have been proposed to avoid congestion at the bottleneck nodes. Among which, random-early-detection (RED) is widely applied in IP network nowadays. It provides high bandwidth (BW) utilisation by dropping packets randomly selected from the queue. These dropped packets will then serve as signals that notify their transmission ends to decrease the sending rate, preventing congestion from happening. Unfortunately, RED algorithm provides little protection against aggressive flows, such as flows with large packets, or congestion insensitive flows. As a result, it is easy for these flows to consume most of the bandwidth, causing unfair bandwidth sharing. In this paper, we propose a new algorithm, named as Power-RED, which aims to achieve BW fairness. Power-RED keeps a throughput statistics, and adjusts packet drop probabilities according to power law when a network tends to become congested. Its goal is to maintain a fair bandwidth share among all incoming connections. Power-RED monitors throughput of active flows, and adjusts drop probabilities individually according to their throughputs for the purpose of BW fairness. The extensive real-world experiments in a PC networks shows that Power-RED can effectively guarantee fairness not only in packet numbers but also in their sizes.

Key words: network control, congestion control, random-early-detection.

1. INTRODUCTION

As data traffic was progressively bursting on the Internet, congestion controls and avoidance has become a crucial issue. Several congestion control algorithms have been proposed to avoid congestion at bottleneck nodes. Random-Early-Detection (RED) [1] is one of the most well-known ones. It continuously monitors the average queue size, and then drops packets randomly before the queue reaches its maximum size. These dropped packets then become signals of congestion that notify their transmission ends to slow down sending rates, preventing congestion from happening. Several RED's variants have been proposed for special concern after RED was announced [2-10]. Some of them tried to deal with fairness issues, for RED can only provide little protection against aggressive flows from consuming most of the bandwidth. These aggressive flows include: TCP flows with short round trip time (RTT), TCP flows with large packet size, and congestion insensitive flows such as UDP flows.

Byun et al. [2] proposed a modified the window

[1] *School of Electrical and Computer Engineering, Cornell University, New York, USA.*
[2] *Department of Electrical Engineering, National Chung Hsing University, Taichung 402, Taiwan, R.O.C.*; 國立中興大學電機工程學系
* Corresponding author, E-mail: chunlin@dragon.nchu.edu.tw
DOI: 10.6287/JENCHU.2014.2501.01

control algorithm that guarantees TCP fairness by collecting network information with the aid of successive explicit congestion notifications. However, it is only applicable in congestion sensitive scenarios. EF-RED [3] modifies the formula of drop probability calculation by taking packet size into consideration. Drop probability of each incoming flow is individually computed, whose value is equal to the square of the ratio of the arriving packet size to the largest packet size among the flows. However, its formula is derived from a steady-state formula in [11], but its calculation is based on real-time statistics, which might have dramatic changes in dynamic networks. CHOKe [4] is a variation of RED aim to deal with congestion insensitive flows. When a new packet arrives, it randomly chooses a packet within the buffer and makes a comparison between their source addresses. If they are the same, both packets are dropped immediately. Otherwise, the chosen packet is left untouched and the incoming packet is processed through the original RED algorithm. If a flow consumes more bandwidth, its packets are more easily to be chosen and then discarded. WARD [5], which is an improved version of CHOKe, not only can it protect TCP flows from UDP flows, but it also can resolve the unfairness problem caused by different TCP versions. StoRED [6] classifies incoming packets into several groups by using a time-varying hash function. It then modifies drop probabilities of each group according to its load. However, this may cause some normal flows to be mispunished due to sharing bandwidth with the aggressive flows. BF-RED [7] activates its control mechanism only when the router is congested, that is, its average queue size is greater than 0.5 (max_{th} + min_{th}) , where max_{th} and min_{th} are, respectively, the upper and lower decision bounds to help determine the mode of drops: no drops, random drops, and forced drops. This threshold, suggested by Zieger et al. in [12] and [13], is a stability criterion of the RED algorithm. BF-RED calculates drop weight of incoming flows based on packet drop history, and uses this as a parameter to adjust the packet drop probabilities. Computer simulations show that BF-RED can minimizes the impact caused by flows with small RTT, and also can alleviates the unfairness resulting from congestion insensitive UDP flows. But packet drop history cannot accurately reflect actual bandwidth shares when drop probabilities are large, and the size of packet is not always a constant in general. Although packet size is limit by maximum transmission unit (MTU), whose value is suggested by IPv4 [14] to be set to 576 bytes, it can still be set to other values. For example, the MTU of Ethernet [15] is set to 1,492 bytes, which means packets from an Ethernet may be twice larger than packets from an IP network when both are interconnected. Hence, to achieve ultimate bandwidth fairness, it is necessary to collect the information of packet size.

Motivated by that RED and CHOKe are not capable of protecting bandwidth fairness, we propose a new algorithm named Power-RED which can achieve fairness in a great extent by monitoring packet transmission history and adjusting packet drop probabilities according to a power law. Therefore, even if TCP version differs among incoming flows, Power-RED can still avoid unfairness, since it simply modifies drop probability according to the throughput. With Power-RED, the unequal share in a complex network can be significantly alleviated. Instead, service providers can maintain fair and reasonable benefit-cost-ratio among their clients. Our approach has been successfully verified in a real-world experimental platform which consists of five personal computers with different operational systems under the topology in Figure 1. The labels on the lines between nodes represent the packet size of the flows. The data sending begins randomly between 0 and 1 second, and the experiment lasts for 1,500 seconds. Considered algorithms are applied only at routers R1 and R2, and Drop-Tail, the rest. Flow *i* is denoted as the connection between node S*i* and D*i*, via the congested routers R1 and R2. We set packet size of flow 1 as 1,496 bytes, which is almost three times larger than the rest of the flows. Furthermore, extensive experiments show that the Power-RED can guarantee fairness not only in packet numbers but also in their sizes, yielding improved data transmission performance.

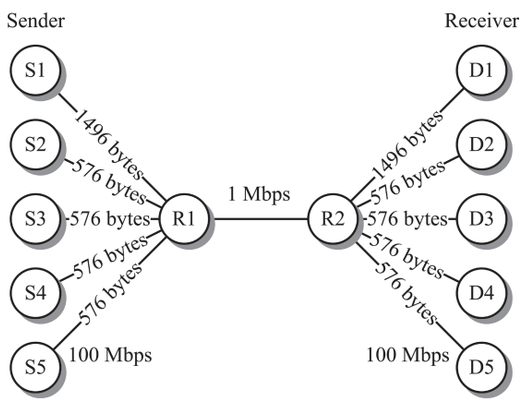

Figure 1. The network configuration under consideration.

2. Power-RED Algorithm

The core idea of Power-RED is to adjust the drop probability of each active incoming flow according to its throughput in a congested router. Assume that there is a congested router, when a new packet arrives, Power-RED checks if it belongs to any known active flows. If so, it then updates the number of active flows. Power-RED continuously monitors, keeps throughput statistics for each active flow, and adopts the information to compute average throughput arithmetically. The drop probability of each active flow is adjusted as follows.

First, the intermediate drop probability is calculated by

$$P_b = min\ (max_p\ (q_{avg} - min_{th}\ )/(max_{th} - min_{th}\ ),1) \quad (1)$$

where max_p represents the maximum drop probabilities, *avg q* represents the average queue size, which is regularly updated, max_{th} and min_{th} are the upper and lower decision bounds to help determine the mode of drops: no drops, random drops, and forced drops, as illustrated in Figure 2.

P_a another intermediate probability according to

$$P_a = \frac{P_b}{1 - cP_b} \quad (2)$$

where c is a count variable defined in RED. The ultimate drop probability for each active flow is derived by passing P_a to a power equation.

$$P_i' = P_a\ (\frac{x_i}{\mu_x})^r \quad (3)$$

where x_i is the throughput of the i-th flow in a time interval, μ_x is the average throughput in the same time interval, and γ is the power factor, which is an integer and is self-adjusted as follows.

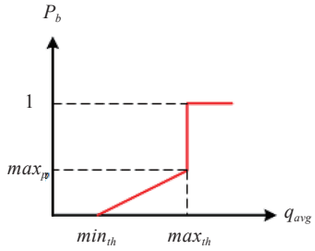

Figure 2. Relation between the average queue size and drop probability P_b is then used to calculate.

The power factor α is initially set to its lower bound, e.g., 0, which means the drop probabilities are left untouched. If the following inequality holds

$$\max_i\ \{x_i\} > \alpha\mu_x$$

where the factor α reflects the extension of control, and γ is smaller than it upper bound, then the value of γ is increased by a step. Thus, if α is set to be 1, the controller will become rather sensitive. Otherwise, γ is decreased by the same step until reaching its lower bound. This makes the flow with the largest throughput suffer more packet drops as time goes by, forcing it to reduce its sending rate if it is congestion sensitive. However, when the protocol of the flow is congestion insensitive, Power-RED will continuously increase its drop probability, until its throughput is reduced to a value no larger than the product of α and μ_x. In this case, Power-RED relies only on raising drop probabilities to maintain bandwidth fairness. The algorithm can be realized in pseudo codes as elucidated below.

Initialization:
$t \leftarrow 0$, $q_{avg} \leftarrow 0$, $c \leftarrow -1$, $x_i \leftarrow 0$, $\mu_x \leftarrow 0$, $\gamma \leftarrow 0$, and $\gamma_{max} \leftarrow 100$
for each packet arrival
  calculate new average queue size q_{avg}
    if queue is nonempty, update q_{avg} by using weight average with weight factor w_q and queue size q
      $q_{avg} \leftarrow (1 - w_q)\ q_{avg} + w_q q$
    else
      $m \leftarrow \phi(t - t_q)$
      $q_{avg} \leftarrow (1 - w_q)^m\ q_{avg}$
if $min_{th} < q_{avg} < max_{th}$
  $c \leftarrow c + 1$
  calculate pr*obability Pb, Pa, and Pi*
    $P_b = \max_p(q_{avg} - min_{th}) / (max_{th} - min_{th})$
    $P_a = P_b / (1 - c \cdot P_b)$
    $P_i = P_a(x_i / \mu_x)^\gamma$
with probability P_i,
    mark or drop the arriving packet of flow i
    $c \leftarrow 0$
else $c \leftarrow -1$
if the queue becomes empty
    $t_q \leftarrow t$
for each packet departure

calculate per flow throughput statistics
calculate average throughput among the flows

$$\mu x \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$$

if $max\{x_i\} > \alpha \cdot \mu_x$ and $\gamma < \gamma_{max}$

$\gamma \leftarrow \gamma + 1$
else if $\gamma > 0$
$\gamma \leftarrow \gamma - 1$

Saved Variables:
q_{avg}: average queue size
c: number of packets since last mark or drop
t_q: the start of queue idle time
x_i: throughput of flow
μ_x: throughput among the active flows
γ: the power factor
γ_{max}: the upper bound of

3. Real-World Implementation

We have realized Power-RED as a Linux kernel module and investigated its performance against CHOKe and conventional RED, which are also two build-in modules. Due to the complexity of the algorithm and the limitations of Linux kernel, BF-RED is presented.

3.1 Experimental Platform and Setup

We conducted a practical implementation by setting up a network topology which contains a cluster of clients running popular operating systems such as Microsoft® Windows XP, and a gateway server running Linux 3.2.8 [16] as the subject system. Figure 3(a) and 3(b) show photos of the experimental system with the configuration of each computer listed in Table 1.

There are some limitations in programing Linux kernel modules. For example, floating point calculation is not allowed in kernel, hence programmers can either handle the status of registers one their own, or map their calculation to integer domain. The later solution is thus suggested. Since the proposed algorithm requires fast power calculation, realizing such calculation by loop sequence will not only consume large resource but also be impractical. We, instead, transform power calculation into simple arithmetic operations with the aid of logarithm, reducing complex computation into simple operations that save much memory consumption. Furthermore, bandwidth is limited by using traffic control utility-tc [17] -- To form the transmission capacity in Figure 1.

Congestion control algorithms are deployed at gateway server R1, and the network topology mimics the configuration illustrated in Figure 1. Once the platform was established, we generate data traffic by using iperf [18], a remote-controllable cross-platform traffic generator written in C. It has been customized to generate TCP or UDP traffics with specified packet sizes and start sending packets at the specified time to the specified destinations.

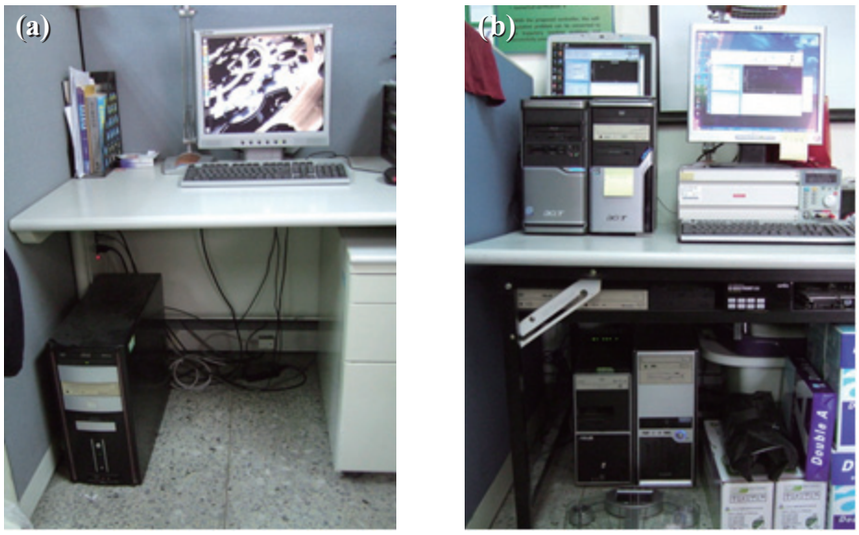

Figure 3. (a) A PC as the platform on which our proposed algorithm is realized; (b) Five clients that generate data traffics.

Table 1. Individual configurations of computers participated in experiment

Computer No.	CPU	RAM Size	Network Interface	OS
R1	Intel® Pentium® 4 3.00 GHz	DDR 1024 MB	Intel® 82562EZ 10/100 Ethernet, Intel® 82559 Ethernet Pro 100	Linux 3.2.8
R2	Intel® Pentium® 4 3.00 GHz	DDR 1024 MB	Intel® 82562EZ 10/100 Ethernet, Intel® 82559 Ethernet Pro 100	Linux 3.8.0-35
S1	AMD Athlon™ 64 3500+	DDR2 1024 MB	NVIDIA® nForce 10/100/1000 Mbps Ethernet	Microsoft® Windows XP
S2	AMD Athlon™ 64 3000+	DDR 512 MB	SiS 190 100/10 Ethernet	Microsoft® Windows XP
S3	Intel Core 2 Duo E8400	DDR2 2048 MB	Realtek PCIe Gigabit Ethernet	Microsoft® Windows XP
S4	Intel Core Duo T2500	DDR2 1024 MB	Realtek PCIe Gigabit Ethernet	Microsoft® Windows XP
S5	AMD Athlon™ 64 X2 5000+	DDR2 1024 MB	Marvell Yukon 88E8056 PCI-E Gigabit Ethernet	Microsoft® Windows XP

On our simulation platform, we choose R2 also serves as our destination ends, D1 to D5, for the following reasons. First, in a dumbbell-like configuration as that proposed in Figure 1, the traffic congestion always occurs before traffic reached the bottleneck, which was the link between R1 and R2, and packets suffered no congestion after leaving R2. Second, since packets generated by traffic generator contained only zero paddings instead of information, the destination needed only to check sequence numbers as well as checksums of these incoming packets. Therefore, R2 can serve as a common sink to all incoming packets. Moreover, it was much easier for us to monitor the status and collect data from a single machine.

3.2 Network Contains Only Congestion Sensitive Flows

To test the performance of Power-RED algorithm in real networks, we initiate a series of evaluations. Similar to that stated in Section 3.1, the first scenario is a network contains only congestion sensitive flows. Packet size of these traffics are set to different values to test if Power-RED possesses better performance on compensating the unfairness caused by different packet sizes than other algorithms. The result is shown in Figure 4.

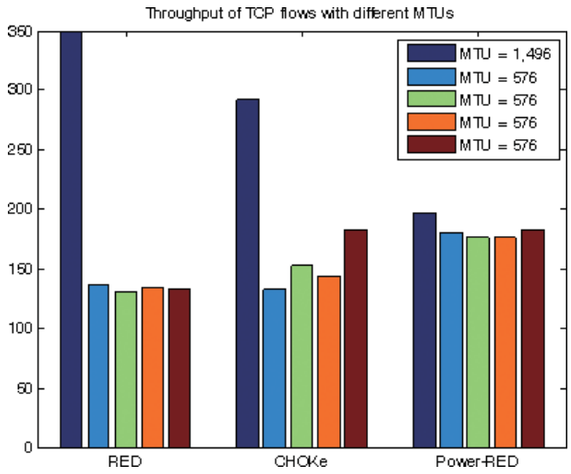

Figure 4. Throughput of TCP flows of different algorithms.

3.3 Network Contains Only Congestion Insensitive Flows

In addition to TCP flows, we proceed to evaluate the performance of Power-RED on UDP flows. The network topology remains the same as that in TCP scenario. The throughput comparison is shown in Figure 5. Practical performance evaluation shows that conventional RED and CHOKe lack the ability to ease unfairness caused by packet size differences whereas fairness can be well maintained by Power-RED.

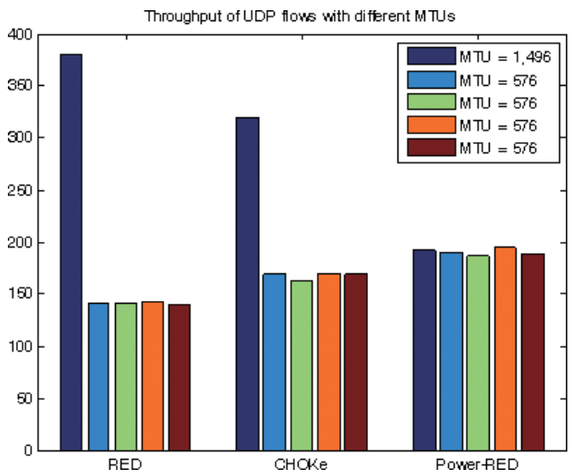

Figure 5. Throughput of UDP flows of different algorithms.

3.4 Network Contains both Congestion Sensitive and Insensitive Flows

To evaluate the proposed algorithm in a more common scenario, we also set up a topology that contains 2 UDP flows and 3 TCP flows and test throughput of different algorithms. The result for throughput measure of such a complex scenario is illustrated in Figure 6. It shows that even though all algorithms fail to reach ideal fairness, the proposed Power-RED still performs better than the other two algorithms.

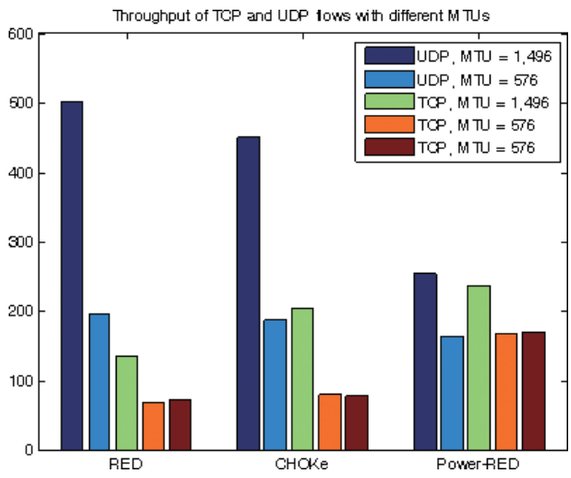

Figure 6. Throughput of UDP and TCP flows of different algorithms.

4. CONCLUSIONS

We have proposed a novel network congestion control algorithm named Power-RED. It is capable of resolving bandwidth fairness issues that has not yet been completely overcome by other algorithms, but also a TCP-friendly router based mechanism. The goal of Power-RED is to maintain a fair bandwidth share among all incoming connections. Power-RED monitors throughput of active flows, and adjusts drop probabilities individually according to their throughputs. Real-world experiments demonstrate that even if flow packet sizes are different, Power-RED with simpler control strategies can guarantee bandwidth fairness than those networked environments governed by RED, CHOKe, or BF-RED algorithm.

Further works involve studying the performance and characteristics of this algorithm under a wide range of packet size differences, network topologies, and real

traffics, providing us deep insight and information for hardware implementations. Besides, the performance analysis of Power-RED with short-lived traffics and dynamic links is currently under investigation.

ACKNOWLEDGMENT

This research was sponsored by National Science Council, Taiwan, R.O.C. under the Grant NSC 101-2221-E-005-015-MY3.

REFERENCES

1. Floyd, S. and Jacobson, V., "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397-413 (1993).
2. Byun, H.J. and Lim, J.T., "Fair TCP Congestion Control in Heterogeneous Networks with Explicit Congestion Notification," *IEE Proceedings-Communications*, Vol. 152, No. 1, pp. 13-21 (2005).
3. Yang, X., Chen, H. and Xiao, P., "An Algorithm of Enhancing RED Fairness," *Proceedings of the 7th World Congress on Intelligent Control and Automation 2008*, *WCICA'2008*, Chongqing, China, pp. 2149-2152 (2008).
4. Pan, R., Prabhakar, B. and Psounis, K., "CHOKe: A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation," *Proceedings of IEEE INFOCOM' 2000*, Tel Aviv, Israel, pp. 942-951 (2000).
5. Ho, C.-Y., Chan, Y.-C. and Chen, Y.-C., "WARD: A Transmission Control Protocol-Friendly Stateless Active Queue Management Scheme," *IET Communications*, Vol. 1, No. 6, pp. 1179-1186 (2007).
6. Chen, S., Zhou, Z. and Bensaou, B., "Stochastic RED and Its Application," *Proceeding of the IEEE International Conference on Communications, 2007, ICC'2007*, Glasgow, UK, pp. 6362-6367 (2007).
7. Yang, J., Liu, J. and Lei, Z., "BF-RED: A Novel Algorithm for Improving Bandwidth Fairness of RED," *Proceedings of IEEE ICNSC'2006*, Fort Lauderdale, FL, pp. 1001-1005 (2006).
8. Abdel-Jaber, H., Woodward, M., Thabtah, F. and Abu-Ali, A., "Performance Evaluation for DRED Discrete-Time Queueing Network Analytical Model," *Journal of Network and Computer Applications*, Vol. 31, No. 4, pp. 750-770 (2008).
9. Abbasov, B. and Korukoglu, S., "Effective RED: An Algorithm to Improve RED's Performance by Reducing Packet Loss Rate," *Journal of Network and Computer Applications*, Vol. 32, No. 3, pp. 703-709 (2009).
10. Tahiliani, M.P., Shet, K.C. and Basavaraju, T.G., "CARED: Cautious Adaptive RED Gateways for TCP/IP Networks," *Journal of Network and Computer Applications*, Vol. 35, No. 2, pp. 857-864 (2012).
11. Floyd, S. and Fall, K., "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 4, pp. 458-472 (1999).
12. Ziegler, T., Fdida, S., Brandauer, C. and Hechenleitner, B., "Stability of RED with Two-Way TCP Traffic," *Proceedings of the Ninth International Conference on Computer Communications and Networks, 2000*, Las Vegas, NV, pp. 214-219 (2000).
13. Zieger, T., Fdida, S. and Brandaur, C., "Stability Criteria of RED with TCP Traffic," *Technical Report*, Paris, France, pp. 1-66 (2000).
14. Postel, J., "RFC 879: The TCP Maximum Segment Size and Related Topics," IETF, (1983). https://tools.ietf.org/html/rfc879
15. Mamakos, L., Lid, K., Evarts, J., Carrel, D., Simone, D. and Wheeler, R., "RFC 2516: A Method for Transmitting PPP Over Ethernet (PPPoE) (1999). http://tools.ietf.org/html/rfc2516.
16. The Linux Foundation, Linux Kernel version 3.2.8. (n.d.). http://www.kernel.org/
17. The Linux Foundation, Iproute2 Package. (2009). http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2
18. NLANR/DAST, Iperf, An IP Network Performance Analyser. (2013). http://sourceforge.net/projects/iperf/

Manuscript Received: Dec. 09, 2013
Revision Received: Feb. 07, 2014
and Accepted: Feb. 07, 2014