Discrimination of Malicious Ddos Attack Traffic Flow From Normal TCP Flow

D . J . Anusha[1],
Academic Consultant,
Department of CSE,
Sri Padmavati Mahila Visvavidyalayam,
Tirupati.

D . J . Prathyusha[2],
M.Tech Student,
Department of CSE,
Sree Vidyanikethan Engineering College,
Tirupati.

A . Supriya[3],
Academic Consultant,
Department of CSE,
Sri Padmavati Mahila Visvavidyalayam,
Tirupati.

***Abstract*— Now-a-days, we have experienced a wave of DDoS attacks threatening the welfare of the internet. These are attacks presenting an increasing threat to the global inter-networking infrastructure. While TCP's congestion control algorithm is highly robust to diverse network conditions, its implicit assumption of end-system cooperation results in a well-known vulnerability to attack by high-rate non-responsive flows. To defend against distributed denial of service (DDoS) attacks, one critical issue is to effectively isolate the attack traffic from the normal ones. A novel DDoS defense scheme based on TCP is hereby contrived because TCP is the dominant traffic for both the normal and lethal flows in the Internet. Unlike most of the previous DDoS defense schemes that are passive in nature, the proposal uses proactive tests to identify and isolate the malicious traffic. Simulation results validate the effectiveness of our proposed scheme.**

Keywords: DDoS defense, proactive test, TCP. Denial of service, retransmission timeout, TCP.

1. INTRODUCTION

Distributed denial of service (DDoS) attacks are probably the most ferocious threats to the integrity of the Internet. It is well known that it is rather easy to launch, but difficult to defend against, a DDoS attack. The underlying reasons include (1) IP spoofing; (2) the distributed nature of the DDoS attack (a huge number of sources generate attack traffic simultaneously); (3) no simple mechanism for the victim to distinguish the normal packets from the lethal traffic.

Consider the following two scenarios. 1) The high rate traffic is legitimate while the attack traffic is low-rate. Rate-limiting is improper in this case. 2) Most flows carry attack traffic during a flood-based DDoS attack while good traffic is low-rate. Under this scenario, even imposing fair sharing of bandwidth does not help the good traffic much because the majority of bandwidth is consumed by malicious flows[1]. To ensure good performance and accommodate as many normal users as possible, it is critical to differentiate traffic. However, it is by no means trivial to make such a distinction. Discrimination based on packet headers is vulnerable to IP spoofing; discrimination based on packet contents may be thwarted by the increasing use of end-to-end encryption.

We hereby propose to identify malicious traffic from their behaviors. We believe that aggressiveness is the salient feature of DDoS traffic, besides IP spoofing. One example of the aggressive behavior is that an attack source may not care about whether it may receive the response from the victim or not, and it can still conduct an attack by bombarding its target with a monstrous number of useless packets. Note that "aggressiveness" is not equivalent to "high-rate". It is possible that a high-rate flow is a normal TCP stream.[2] The receiver may identify the aggressive behavior by intentionally testing the response of a source upon certain control signals from the receiver. Any source that fails to pass such tests is regarded as a lethal one and can be punished accordingly. However, a source, which passes the test, may not be necessarily benign.

A sophisticated attacker may pass the test by behaving well initially, and perform deleterious operations later. To handle this case, the receiver may increase the frequency of such tests. A better solution is to introduce some dynamics into the test and randomly determine the frequency of the test for each flow, especially the high-rate ones. To accommodate high-rate legitimate traffic better, we set a threshold that defines the maximal number of successful tests for a flow. No more tests are conducted on packets from a flow once the flow successfully passes the specified number of tests. By actively testing a source, the receiver can determine with high confidence the nature of a flow from that source and react accordingly. Filtering based on behavior brings an attack source into a dilemma: sends packet aggressively at the risk of being identified and punished, or reduce the attack rate to meet the requirements of the receiver so that the effect of an attack is diminished. In so doing, the receiver may throttle the scope and impact of potential attacks.

The above design is feasible for TCP solely because TCP has the built-in congestion control and reliable transmission mechanism. Note that TCP is the dominant traffic in the Internet, and as much as 90% of DDoS traffic uses TCP. Currently, TCP occupies 80% in terms of the number of flows, and 90% with respect to the number of packets. It is thus essential for DDoS defense schemes to accommodate TCP traffic effectively and efficiently[2].

II. LOW-RATE TCP-TARGETED DENIAL OF SERVICE ATTACKS

Denial of service (DoS) attacks consume resources in networks, server clusters, or end hosts, with the malicious objective of preventing or severely degrading service to legitimate users. Resources that are typically consumed in such attacks include network bandwidth, server or router CPU cycles, server interrupt processing capacity, and specific protocol data structures. Example DoS attacks include TCP SYN attacks that consume protocol data structures on the server operating system; ICMP directed broadcasts that direct a broadcast address to send a flood of ICMP replies to target host thereby overwhelming it; and DNS flood
attacks that use specific weaknesses in DNS protocols to generate high volumes of traffic directed at a targeted victim.

Common to the above attacks is a large number of compromised machines or agents involved in the attack and a "sledgehammer" approach of high-rate transmission of packets toward the attacked node. While potentially quite harmful, the high-rate nature of such attacks presents a statistical anomaly to network monitors such that the attack can potentially be detected, the attacker identified, and the effects of the attack mitigated.

TCP congestion control operates on two time-scales. On smaller time-scales of round trip times (RTT), typically tens to hundreds of milliseconds, TCP performs additive-increase multiplicative-decrease (AIMD) control with the objective of having each flow transmit at the fair rate of its bottleneck link. At times of severe congestion in which multiple losses occur, TCP operates on longer time-scales of retransmission time out (RTO).In an attempt to avoid congestion collapse, flows reduce their congestion window to one packet and wait for a period of RTO after which the packet is resent. Upon further loss, RTO doubles with each subsequent timeout. If a packet is
successfully received, TCP re-enters AIMD via slow start[3]. To explore low-rate DoS, we take a frequency-domain perspective and consider periodic on-off "square-wave" shrew attacks that consist of short, maliciously-chosen-duration bursts
that repeat with a fixed, maliciously chosen, slow-time-scale frequency. Considering first a single TCP flow, if the total traffic (DoS and TCP traffic) during an RTT-time-scale burst is sufficient to induce enough packet losses, the TCP flow will enter a timeout and attempt to send a new packet RTO seconds later.

If the period of the DoS flow approximates the RTO of the TCP flow, the TCP flow will continually incur loss as it tries to exit the timeout state, fail to exit timeout, and obtain near zero throughput. Moreover, if the DoS period is near but outside the RTO range, significant, but not complete throughput degradation will occur. Hence the foundation of the shrew attack is a null frequency at the relatively slow time-scale of approximately RTO enabling a low average rate attack that is difficult to detect. In a simplified model with heterogeneous-RTT aggregated flows sharing a bottleneck link, we derive an expression for the throughput of the attacked flows as a function of the time-scale of the DoS flow, and hence of the DoS flow's average rate.

III. DEFENDING AGAINST A DENIAL-OF-SERVICE ATTACK ON TCP

The Internet has undergone a phenomenal growth in the recent past. However, during this period the vulnerabilities found in the TCP/IP protocol suite have been subjected to significant revelation as well. Particularly, the details of a simple denial-of-service attack popularly known as "SYNflooding" were published in two underground magazines and this attack still continues to pose a serious threat against the availability of TCP services. The SYN-flooding attack exploits a common TCP implementation issue and a well-known authentication weakness found in IP[5], which do not seem correctable in the near future since they require the modification of the standards.

Preventive approaches such access control, are not applicable to SYN flooding attacks since the general target is public services. Network monitors are known to be able to detect such low-level network based attacks.

The Internet is a worldwide network that uses the TCP/IP (Transmission Control Protocol/Internet Protocol) protocol suite for communications. IP is the standard internet layer protocol of TCP/IP, which provides for transmitting blocks of data called data grams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. IP is a connectionless protocol. Therefore, IP data grams may get delivered out of order and there is no guarantee that a datagram successfully gets its destination. IP does not either provide address authentication. Actually, any host can send data grams with any source IP address. Therein lies most of the threat against the integrity, secrecy and availability of today's Internet assets.

TCP is the connection oriented transport layer protocol of the TCP/IP suite, designed to
provide a reliable logical circuit between pairs of applications in hosts attached to the Internet. TCP assumes that it can obtain an unreliable datagram service from lower level protocols. In the Internet, this service is provided by IP. The primary purpose of TCP is to provide a reliable connection service on top of a less reliable internet communication system. For this, TCP supports facilities in the following areas: reliability, flow control, multiplexing and connections.

In order to support reliability and flow control, TCP initializes and maintains certain status information for each data stream. The combination of this information including sockets, sequence numbers, and window sizes, is called a connection. A pair of socket (4 tuple consisting of the client IP
address, client port number, server IP address and server port number) specifies the two end points that uniquely identifies each TCP connection in the Internet. A TCP packet is called a "segment".

For a connection to be established, the two TCPs must synchronize on each other's sequence numbers. This is done by exchanging connection establishing segments carrying a SYN control bit and initial sequence numbers

(ISNs). The synchronization requires each side to send it's own ISN and to receive an acknowledgment of it from the other side[3]. Each side must also receive the other side's ISN and send an acknowledgment (ACK).This is illustrated in Figure 1.

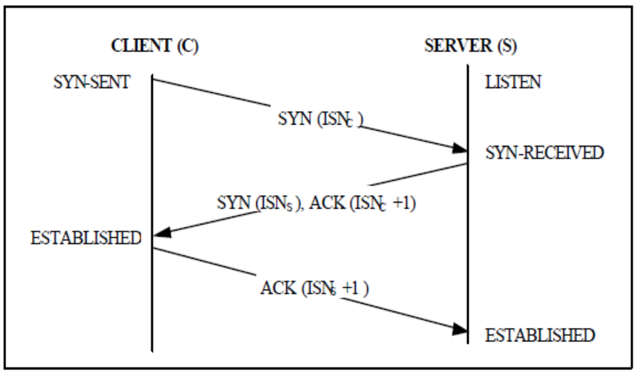


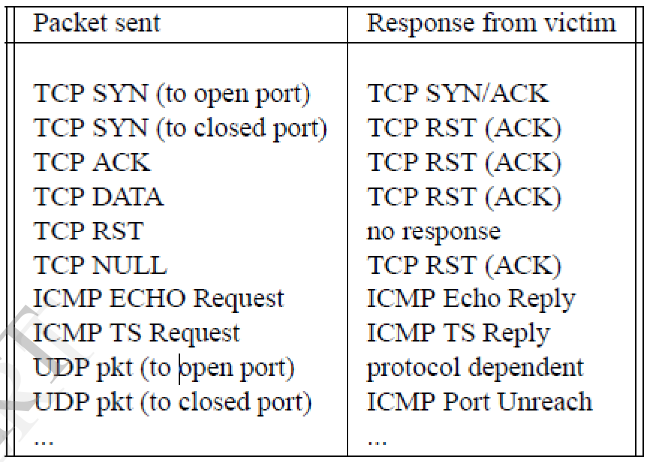
Figure 1:TCP 3-way HandShake

A three-way handshake is necessary because the client and server sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking ISNs.

The server which receives the first SYN has no way of knowing whether the segment is an old delayed or not, and thus it must ask the sender to verify this SYN. TCP servers are concurrent. A TCP server starts a new process to handle each client therefore the listening server is always ready to handle the next coming connection request. However, there is still a chance that multiple connection requests arrive while the server starts a new process. In order to handle these incoming connection requests while the listening application is busy TCP employs a fixed length queue for the connections that have not been accepted by the server application. This is referred to as the "backlog queue". However, there is an upper limit to the number of connection requests waiting in this queue. This limit is specified by the listening application by calling the listen() system call.

When a new connection request arrives (i.e., a SYN segment), TCP acknowledges this if there is room for this new connection on the requested service's queue. However it should be noted that the server application will not see these new connection until the third segment of the 3-way handshake is received. If there is no room on the requested service's queue, TCP ignores the received SYN packet. By ignoring the SYN packet the server forces the client to retransmit the SYN later, hoping that the queue will then have room. The SYN-Flooding attack exploits both this design and the authentication weakness in IP. The attacker generates several SYN segments with invalid source IP addresses to a target TCP server. Since the source hosts are non-existing or closed, the 3-way handshake for these connections will never complete or be broken (a valid source host would certainly reset such an unreferenced connection by sending a RST segment), resulting in several half-open connections filling up the server's backlog queue. Then, the target service becomes unavailable (interrupted) until a connection establishment timer expires.

Denial-of-service attacks consume the resources of a remote host or network that would otherwise be used for serving legitimate users. There are two principal classes of attacks: *logic*
attacks and *flooding* attacks. Attacks in the first class, such as the "Ping-of-Death", exploit existing software flaws to cause remote servers to crash or substantially degrade in performance. Many of these attacks can be prevented by either upgrading faulty software or filtering particular packet sequences, but they remain a serious and ongoing threat. The second class, flooding attacks, overwhelm the victim's CPU, memory, or network resources by sending large numbers of spurious requests. Because there is typically no simple way to distinguish the "good" requests from the "bad", it can be extremely difficult to defend against flooding attacks.



Packet sent	Response from victim
TCP SYN (to open port)	TCP SYN/ACK
TCP SYN (to closed port)	TCP RST (ACK)
TCP ACK	TCP RST (ACK)
TCP DATA	TCP RST (ACK)
TCP RST	no response
TCP NULL	TCP RST (ACK)
ICMP ECHO Request	ICMP Echo Reply
ICMP TS Request	ICMP TS Reply
UDP pkt (to open port)	protocol dependent
UDP pkt (to closed port)	ICMP Port Unreach
...	...

Table 1: A sample of victim responses to typical attacks.

TCP FLOW DIFFERENTIATION

A. Connection Establishment

Whether a connection has been established has a significant implication to the receiver. A successfully established connection indicates that both ends have completed the three-way handshaking procedure, which implies that IP spoofing is not employed by the source. For an incomplete connection, on the other hand, the receiver shall be alert, and be conservative in its resource consumption. Possible measures to mitigate potential attacks include (1) tightening the total bandwidth allocated to all incomplete connections, and (2) significantly reducing the timeout value to avoid buffer occupied by half open connections for a long time, or no buffer allocation at all for half-open connections.

B. Benign and Malicious Flows

TCP is an end-to-end solution that requires close orchestration between the sender and the receiver. To characterize the nature of a TCP flow (after a successful connection), the receiver can actively test the response of the sender by delaying the ACK packets intentionally. If the sender is normal, it will take action accordingly and reduce its sending rate. On the contrary, for a DDoS attack, two cases may occur. One is that the sender uses forged source IP addresses, and thus cannot receive the rate-reduction message from the receiver. It has no idea of the proper sending rate.

The other scenario is that the sender does receive the notification, but it neglects it and just keeps sending packets, thus violating the protocol, and it may be punished by the recipient to reduce its share or even block its traffic. This procedure is dynamic.

The protected site can decide the frequency and extent of rate reduction so that no perpetrator can easily fool the system to believe that the traffic from the perpetrator is normal. Fig. 1 depicts the flowchart of the traffic differentiation procedure.

Upon the arrival of a new incoming packet, the receiver first determines to which flow the current packet belongs by checking the tuple of (source IP address, source port number, destination IP address, destination port number). If it is the first packet of a flow, the receiver examines whether the
number of admitted flows reaches the maximum flow count, a threshold set by the receiver to ensure proper provisioning of quality of service.

If this does occur, the packet is dropped. Otherwise, the new packet is admitted after updating the flow table maintained by the receiver, resulting in an increment of the flow count by 1, and initialization of several counters, such as the number of successful tests and the number of failure tests.

The receiver then checks the behavior history of the flow. If the number of failure tests is no less than a threshold, *f*, the packet will be dropped. An integer larger than 1 is selected to prevent our scheme from falsely identifying the behavior of a flow.

A low value of *f* may exacerbate packet dropping. In case of a false identification, subsequent packets from an innocent flow will be blocked.The following figure shows that the pictorial representation of traffic differentiation. Through extensive simulations we found the difference between proper idefication rate and acceptable performance impact

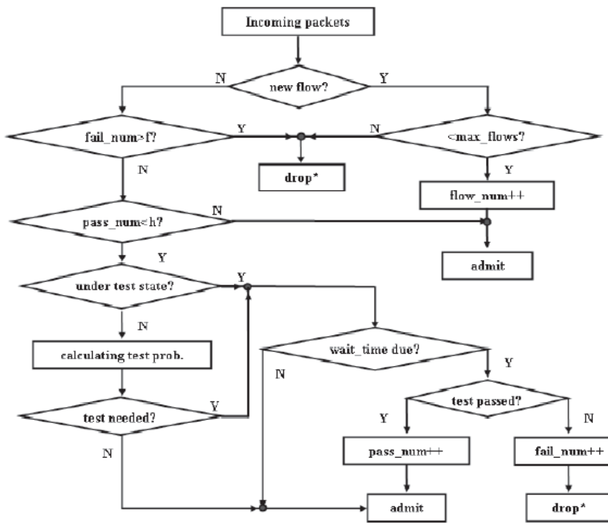


Figure 2. Flowchart of the traffic differentiation.

Selecting a too high value is unwise, either. A high *f* delays the packet dropping decision, and thus subsequent packets of a malicious stream may still consume system resources. Through extensive simulations, we found that *f*=3 provides a good balance between the proper identification rate and the acceptable performance impact.For the flow whose behavior is not so bad in the past, our scheme further examines whether the flow has passed a certain number of tests, *h*. The receiver will admit directly any packet of flows having passed *h* tests successfully (Similarly, some tradeoff has to be made to determine a proper value of *h*. We set *h* to 6 by trials and errors). For other flows, we further check the current state of the flow. If the flow is under a test, its current rate shall not exceed one half of its previous one (the receiver enforces this constraint by manipulating the reverse ACK rate). If the flow conforms to that constraint, the flow passes the current test and its *pass num* is incremented by 1.

Otherwise, the flow fails one test. In the case that the flow is not in the state of testing, its sending rate is compared with that of the fair share of each flow. The result of the comparison is used to determine the test probability for that flow. Obviously, a flow with less bandwidth consumption is subject to less number of tests. The probability *p* for a high-rate flow (over the fair share) is 1/(*pass*
num+1). At the very beginning, *pass num* is 0 for all flows. Therefore, as long as a high-rate flow has not passed a test, its chance of being tested is 100%. As the number of successful tests of a flow increases, its test probability reduces. The test probability *p* for the less resource-consumption flow is 1/*max*(*m*, 2***h*), where *m* is the total number of flows. For the normal case, *m* is far greater than 2*h*; thus, *p*=1/*m*. We use the *max*(.) function to address the case that only a few flows exist in the system and ensure that the test probability for a low-rate flow is at most 1/2 of that of a high-rate one.

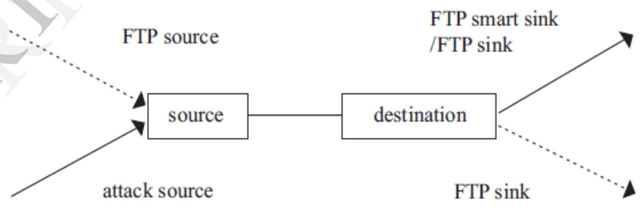


Figure 3. Simulation setup for comparison study of the effectiveness of traffic differentiation.

The rate of a flow is calculated according to the following formula, *num pkt*sz pkt/t*, where *t* is the time interval (window), *num pkt* the number of packets received during this period, and *sz pkt* the packet size. It is worth mentioning that the flow rate calculated here is not the average rate of a flow, as normally used by others, because we update the starting time of a flow once it passes a test. In so doing, we can effectively thwart a low-rate DoS attack which sends a burst of attack packets to incite congestion and keeps silence for a much longer period to significantly lower its average rate
in order to escape detection and filtering.
Four scenarios may happen. 1) An attack source always behaves well, and thus the effect of an attack is greatly diminished. 2) An attack source behaves well initially and misbehaves later. When tested, the constraint that the current rate is at most 1/2 of the previous rate will not be satisfied, and the source fails the test. 3) An attack source always misbehaves, that may be easily thwarted by the fail count. 4) An attack source misbehaves at first and behaves well later. In this case, the attack source is exposed to more chances of being tested because its

pass num is off-setted by the *fail num* once it fails a test. Note also that a low-rate flow is also subject to test, though at a lower probability in our design. As time passes by, the chance that a low-rate flow has never been tested by the receiver is very low. We enforce this policy to contain the case that some low-rate streams are malicious[1][5].

IV. SIMULATIONS

To test the effectiveness of our proposed traffic differentiation, we set up a simulation scenario including 1 FTP source and an attack source, as shown in Fig.3. These flows pass through the same bottleneck link. The difference is that one simulation uses a normal FTP sink to accept packets from both flows, and the other uses our developed TCP sink, called TCP smart sink. The simulation results are shown in Fig.4 and Fig. 5.

Fig. 4 shows the throughput of the attack traffic using the FTP sink while Fig. 5 presents the throughput of the attack traffic using our proposed TCP smart sink, in which the throughput of attack traffic drops drastically after 3.2s. After 42.3s, the attack traffic is totally blocked. In contrast, using the FTP sink as the receiver, the attacker may keep the highest throughput during its lifetime. The result demonstrates the effectiveness of our proposed traffic differentiation.

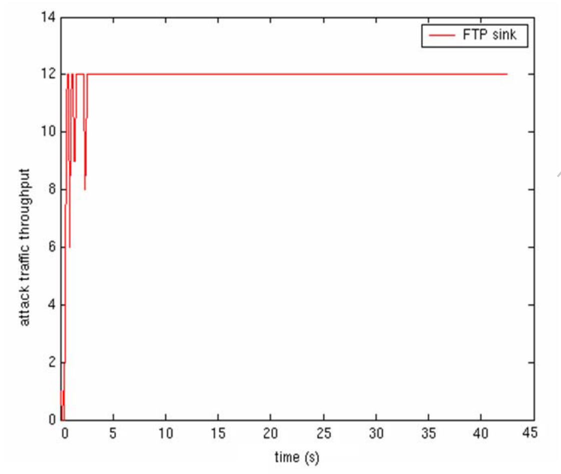


Figure. 4. Attack traffic throughput using FTP Sink

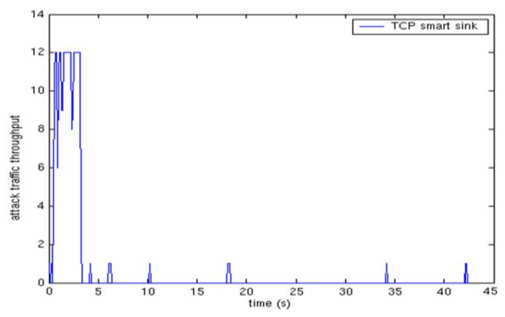


Figure 5. Attack traffic throughput using TCP Smart Sink

V. CONCLUSION

Denial-of-service attacks remain a significant problem despite the widespread deployment of perimeter security devices such as firewalls and IDS. This paper presented a novel DDoS defense scheme The salient benefits of this proposal mainly lie in its capability of identifying malicious TCP flows by proactive tests. Preliminary simulation results have validated our design.

REFERENCES

[1] Zhiqiang Gao, *Member, IEEE,* and Nirwan Ansari, *Senior Member, IEEE* "Differentiating Malicious DDoS Attack Traffic from Normal TCP Flows by Proactive Tests" *IEEE communication letter, Vol. 10, NO. 11, Nov 2006 793*

[2] P. Mutaf, "Defending against a denial of service attack on TCP," in *Proc.International Symposium on Recent Advances in Intrusion Detection (RAID 99)*

[3] J. Haggerty, T. Berry, Q. Shi, and M. Merabti, "DiDDeM: a system for early detection of TCP SYN flood attacks," in *Proc. IEEE GLOBECOM 2004*, pp. 2037-2042.

[4] J. Xu and W. Lee, "Sustaining availability of Web services under distributed denial of service attacks," *IEEE Trans. Comp.*, special issue on reliable distributed systems, vol. 52, no. 2, pp. 195-208, Feb. 2003.

[5] D. Yau, J. Lui, F. Liang, and Y. Yan, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles," *IEEE/ACM Trans Networking*, vol. 13,
29-41, Feb. 2005.

[6] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks (the shrew vs. the mice and elephants)," in *Proc. ACM SIGCOMM 2003*, pp. 75-86.

[7] D. Moore, G. Voelker, and S. Savage, "Inferring Internet denial-of-service
activity," in *Proc. 10th USENIX Security Symposium*, pp. 9-22.

[8] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," Internet RFC 2414, 1998.

[2] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *ACM SIGCOMM*, Vancouver, BC, Canada, Sep. 1999, pp. 263–274.

[9] F. Anjum and L. Tassiulas, "Fair bandwidth sharing among adaptive and non-adaptive flows in the Internet," in *Proc. IEEE INFOCOM*, New York, NY, Mar. 1999, pp. 1412–1420.

[10] R. L. Carter and M. E. Crovella, "Measuring bottleneck link speed in packet-switched networks," *Perform. Eval.*, vol. 27, no. 28, pp. 297–318, 1996.

[11] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *J. Internetworking: Res. Exp.*, vol. 1, pp. 3–26, Sep. 1990.

[12] C. Dovrolis, P. Ramanathan, and D. Moore, "What do packet dispersion techniques measure?," in *Proc. IEEE INFOCOM*, Anchorage, AK,
Apr. 2001, pp. 905–914.