Automatically Segregating Greedy and Malicious Internet Flows

José Carlos Brustoloni
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
Email: jcb@cs.pitt.edu

Shuo Chen
Cybersecurity and Systems Management Group
Microsoft Research
Redmond, WA 98052, USA
Email: shuochen@microsoft.com

***Abstract*— In the current Internet, compliance with TCP congestion control rules is voluntary. Noncompliant flows can gain unfair performance advantages or deny service to other flows. We propose a scheme that automatically detects and segregates noncompliant flows and preserves network availability for compliant flows. Our scheme requires modifications only in access routers and is incrementally deployable. Experiments demonstrate that our scheme is effective and has acceptable overhead.**

I. Introduction

Most Internet traffic uses TCP. Interestingly, TCP did not originally provide congestion control [1]. It provided only flow control, which prevents senders from overrunning receivers but not the network. By the mid-eighties, the Internet was on the brink of *congestive collapse* [2]: senders were injecting into it so much traffic that most packets were dropped before reaching their destination. TCP congestion control was introduced at that time, causing senders to increase transmission rate additively while there is no packet loss and decrease transmission rate multiplicatively when there is loss. This simple end-to-end scheme enabled the Internet to grow orders of magnitude larger while averting congestion.

However, even today, many senders on the Internet do not implement TCP's congestion control mechanisms. Internet flows with congestion control compatible to that of TCP are often called *TCP-friendly*, while other flows are called *TCP-unfriendly*. Applications that require a constant bit rate or are ill-served by TCP's error recovery, such as voice and video applications, often use UDP and are TCP-unfriendly. Greedy or malicious applications can also use UDP or ICMP or implement TCP without congestion control, so as to gain an unfairly large, TCP-unfriendly share of the network's bandwidth and thus boost their own performance or deny service to other flows.

TCP-unfriendly flows jeopardize Internet stability and scalability [3]. By enabling targeted or generalized denial-of-service (DoS) attacks, such flows also threaten Internet security. There has been significant progress toward TCP-friendly protocols for applications such as voice and video [4], [5]. Techniques have also been proposed for tracing back or throttling greedy or malicious flows. However, some of these proposals may be impractical because they require massive [6] or even universal [7] deployment to be effective. Many proposals also impose capital or operational costs without economic return to the parties that bear those costs [8], [9], [10], [11].

This paper contributes ASTUF (Automatic Segregation of TCP-Unfriendly Flows), an architecture for a novel *value-added service* that leverages already-deployed quality-of-service (QoS) mechanisms to limit the impact of TCP-unfriendly flows on the performance of TCP-friendly flows. In ASTUF, a subscribing server's and respective client's access routers monitor, classify, and mark traffic between client and server as either TCP-unfriendly (default) or TCP-friendly. Routers between the client and the server forward packets in separate classes of service, such that TCP-friendly flows gain at least a minimum share of the network bandwidth, regardless of concurrent TCP-unfriendly flows. Thus, ASTUF isolates TCP-friendly flows of subscribing servers and their clients from many of the current Internet's vulnerabilities to greedy or malicious flows.

Many TCP-friendly schemes have been analyzed and simulated before. Recent results show, however, that correctly implementing such schemes can be tricky [12]. This paper describes an actual implementation and gives a rare perspective on critical details and actual performance and resource requirements. Unlike many previous proposals, ASTUF is effective even if deployed only by a few service providers, and enables service providers to recover, via server subscriptions, the costs involved in its deployment. ASTUF requires modifications only in the access routers of service providers of subscribing servers and respective clients. Other routers between a subscribing server and respective clients need to have their QoS mechanisms configured, but do not need to be otherwise modified. Remaining routers on the Internet require no reconfiguration or modification.

The rest of this paper is organized as follows. Section II reviews the QoS and TCP modeling techniques used by ASTUF. Section III describes ASTUF's threat model, QoS and revenue handling, and packet marking algorithms. Section IV presents our experimental results. Section V discusses related work, and Section VI concludes.

II. Background

A. Internet QoS mechanisms

The current Internet provides a single (best-effort) class of service (CoS). Differentiated Services (DiffServ) [13] enables Internet routers to support a larger (albeit limited) number of classes of service. DiffServ edge routers mark for some CoS each packet arriving from a host, according to the host's Service-Level Agreement (SLA) and the flow's conformance to that SLA. SLA policing requires per-flow state. Edge routers can maintain such state because of their relatively low bandwidth and number of flows. On the contrary, DiffServ core routers service packets based solely on packet markings, and therefore do not need per-flow state. Because DiffServ requires per-flow state only in edge routers, it scales well.

Most commercially available edge and core routers support DiffServ and one or more packet scheduling algorithms other than FIFO. Two popular alternatives are priority scheduling and deficit round-robin (DRR) [14]. Within a CoS, service is always FIFO. However, if there are packets of different classes, priority scheduling always serves first the highest-priority class. Priority scheduling may cause starvation of lower-priority classes. In DRR, each class i is configured with some $quantum_i$. Given any two classes i and j, DRR serves packets in such order that, on average, $bandwidth_i/bandwidth_j = quantum_i/quantum_j$. DRR guarantees that, if $quantum_i \neq 0$, i does not suffer starvation. Both priority and DRR scheduling are simple to implement and have $O(1)$ complexity. They are also *work-conserving*, i.e., do not adversely impact network utilization.

B. TCP modeling

Padhye et al. [15] have shown that the maximum data transmission rate B_e of a TCP Reno sender (in segments per second) is:

$$B_e = \min(\frac{W_{max}}{RTT}, B'_e) \quad (1)$$

$$B'_e = \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_O \min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \quad (2)$$

where W_{max} is the receiver's maximum advertised window size, RTT is the round-trip time between sender and receiver, p is the packet-loss event rate, b is the number of data segments acknowledged by an ACK segment (typically 2), and T_O is the timeout, with resolution of 0.5 s.

Note that a single packet-loss event may cause several consecutive lost packets. Goyal, Guerin and Rajan have analyzed the relationship between p and the unconditional packet-loss rate p' on the same path [16]. Note also that Padhye et al. validated 1 and 2 using RTT and p values *actually experienced* by the respective TCP flows. He, Dovrolis and Ammar have shown that **large errors can occur** if RTT and p are measured by *ping* probes *before* the TCP flow is injected into the network [12]. This is because the TCP flow can significantly impact RTT and p. **Smaller but still significant errors** can occur if RTT and p are estimated by periodic *ping* probes *during* the TCP flow. These errors are due to the mismatch between the probes' fixed period and TCP's burstiness and variable sampling period [12].

Equations 1 and 2 are for TCP Reno. Similar equations exist for other TCP-friendly protocols [5].

III. Design

This section describes ASTUF's threat model, QoS and revenue handling, and packet marking algorithms.

A. Threat model

ASTUF assumes that greedy or malicious users may arbitrarily modify or use nodes or links they control. They may control any number of hosts connected to any service provider. They may also control routers and links of service providers that do not support ASTUF. However, they cannot control any router or link of any ASTUF service provider or internet exchange that an ASTUF service provider connects to.

B. QoS handling

ASTUF assumes that a host's uplink to and downlink from an ASTUF service provider directly or indirectly connect to a same access or edge router within that service provider (for simplicity, called "access router" in the following). All access routers of ASTUF service providers need to be modified so as to classify and mark packets as either TCP-friendly or TCP-unfriendly, as described in Section III-D. Packet markings use a QoS scheme's identifiers, e.g. DiffServ codepoints [13].

ASTUF also assumes that all routers on paths between any two ASTUF access routers support distinct classes of service for TCP-friendly and TCP-unfriendly flows. Such support may use, e.g., DiffServ with priority or DRR scheduling, as discussed in Section II-A. Typically, such configuration does not require modifications in existing routers.

The QoS schemes used by different ASTUF service providers need not be the same. It suffices that, in each ASTUF service provider, the impact of TCP-unfriendly traffic on the performance of TCP-friendly traffic be limited. Packets arriving from a first ASTUF service provider and being forwarded to a second ASTUF service provider may need to have markings translated to denote equivalent classes in each service provider. Moreover, at Internet exchanges or peering points, packets arriving from a service provider that does not support ASTUF need to be marked TCP-unfriendly, which is the default classification. The ability to perform such translations can be expected to already exist in internet exchanges and peering routers that support QoS schemes such as DiffServ.

ASTUF is incrementally deployable. If a router of an ASTUF service provider does not mark packets or separate classes of service as described above, greedy or malicious users may be able to exploit it to obtain unfair advantage or deny service to TCP-friendly flows. However, ASTUF does not suffer any degradation if deployed in only one or a few service providers.

ASTUF service providers may offer SLAs for best-effort traffic, e.g., gold, silver, and bronze levels, each with different

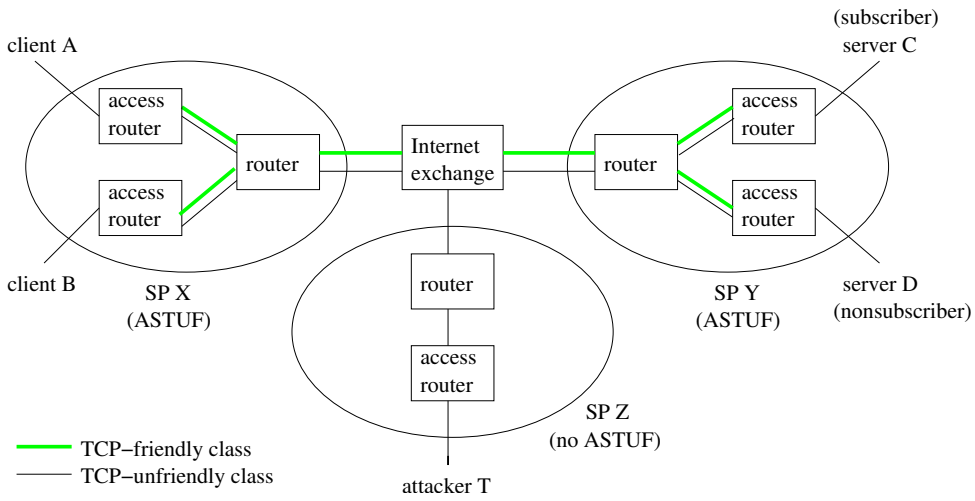

Fig. 1. Incremental deployment of ASTUF

bandwidth limits. If present, such best-effort SLAs form subclasses within ASTUF's TCP-friendly and TCP-unfriendly classes. On the other hand, SLAs for traffic other than best-effort (e.g., video or voice) are subject to admission control and SLA enforcement that prevents their use for DoS attacks. They are handled separately from ASTUF.

C. Revenue handling

An ASTUF service provider X can offer ASTUF protection to any Internet server C that has Internet access via some ASTUF service provider Y (where possibly $X \neq Y$). This is illustrated in Fig. 1. Typically, ASTUF would be offered on a subscription basis. ASTUF service providers (e.g., X and Y) can share such revenues much like they currently share access fees (e.g., via peering agreements).

A packet that arrives on an ASTUF access router's access link is *eligible* for the TCP-friendly CoS only if: (1) the packet's destination is an ASTUF subscribing server or a client of such a server, and (2) the sender has specified the packet to be TCP-friendly. A packet's sender uses the IP header's protocol field (sometimes augmented by a port number) to specify the packet's transport-layer protocol (e.g., a TCP-friendly protocol). The protocol (e.g., TCP) and source and destination IP addresses and port numbers in a packet's header identify what flow the packet belongs to. A connection is a pair of flows, one in each direction, between a client and a server.

For example, in Fig. 1, TCP connections between clients A or B and server C are eligible for the TCP-friendly class. On the other hand, ASTUF marks TCP-unfriendly any flows (TCP or other) between clients A or B and server D, because D is not a subscriber. ASTUF also marks TCP-unfriendly any flows involving attacker T, because T's access provider does not support ASTUF.

Each ASTUF access router needs to keep state proportional to the number of connections in the TCP-friendly CoS that go through it. Since these routers are in the network's edge, the number of such connections can be expected to be reasonably small. If this number becomes excessive, the service provider can reduce it by increasing the ASTUF subscription price (which can be expected to be inversely related to the number of subscribing servers). ASTUF does not require state in other routers.

D. Packet marking

An ASTUF access router marks a packet for the TCP-friendly CoS only if: (1) the packet is *eligible*, as discussed in the previous section, *and* (2) the router's measurements verify that the packet belongs to a connection that indeed is TCP-friendly. Otherwise, the router marks the packet TCP-unfriendly. ASTUF access routers make three types of measurements for verifying TCP friendliness: (1) flow data transmission rate; (2) flow overhead; and (3) number of connections between a given client and server. These measurements are used for packet marking and affect packet scheduling only via such markings.

ASTUF access routers use Eqs. 1 and 2 (or similar equations, depending on the specific TCP-friendly protocol) to verify that a flow's data transmission rate is TCP-friendly. These equations require estimates of the flow's p and connection's RTT. ASTUF access routers cannot rely on transport-layer headers set by hosts to make such estimates, because hosts can spoof these headers. ASTUF access routers do not use periodic *ping* probes, either, because the latter can cause large errors, as discussed in Section II-B. Therefore, ASTUF access routers estimate p and RTT by **inserting measurement probes into the TCP-friendly segments themselves**.

ASTUF measurement probes are carried as transport-layer (e.g., TCP) options. After inserting a probe into a segment, an ASTUF access router updates the transport-layer checksum, so that the latter remains valid. To avoid fragmentation of maximally-sized packets, ASTUF access routers reduce by the size of their measurement probes the maximum transmission unit (MTU) of their backbone links. Senders that discover

and adjust to path MTUs as specified in [17] learn the ASTUF-reduced MTUs. Therefore, when a path is protected by ASTUF, these senders automatically leave room in each TCP-friendly segment for insertion of an ASTUF measurement probe. A sender cannot itself insert a (possibly spoofed) ASTUF measurement probe into a packet because (1) ASTUF access routers replace such probes in TCP-friendly packets, and (2) ASTUF access routers ignore measurement probes in TCP-unfriendly packets (including any packets received from service providers that do not support ASTUF).

ASTUF measurement probes are 12 bytes long and contain: (1) the ingress router's current timestamp, (2) the latest timestamp the ingress router received from the egress router, (3) the count of flow's segments forwarded by ingress router, and (4) the count of reverse flow's packet-loss events detected by the ingress router. The egress router looks for gaps in the ingress router's segment counts to detect the flow's packet-loss events. A sliding-window bitmap is used to allow for out-of-order segment reception. The egress router obtains a sample r_s of the connection's round-trip time by taking the difference between the egress router's current timestamp and the latest timestamp the ingress router received from the egress router. The egress router also obtains a sample p_s of the reverse flow's packet-loss event rate by taking the difference between the counts of the reverse flow's packet-loss events detected by the ingress router at different egress router timestamps, and dividing the result by the difference in the counts of the reverse flow's segments forwarded by the egress router at those timestamps. Exponentially-weighted moving averages of the samples r_s and p_s are then used as estimates of RTT and p in Eqs. 1 and 2. Because ASTUF estimates RTT and p between access routers, instead of between the end hosts, the resulting value of B_e can be expected to be an upper bound for the data transmission rate of a TCP-friendly sender under the prevailing network conditions.

For each flow that is eligible for the TCP-friendly CoS, ASTUF ingress routers count the flow's number of data bytes sent, and obtain samples b_a of the flow's actual data transmission rate by taking the difference in these counts at different ingress router timestamps and dividing the result by the interval between those timestamps. An exponentially-weighed moving average of the samples b_a is then used as an estimate of the flow's actual data transmission rate B_a. ASTUF access routers mark a flow's packets TCP-friendly only if the flow satisfies $B_a < \phi_1 B_e$, where $\phi_1 \geq 1$ is a safety margin. In our experiments, we set $\phi_1 = 3$.

Transport-layer **control segments** (e.g., TCP ACKs) **are not limited** by B_e, **because they can be sent without any data**. Therefore, ASTUF access routers also count and limit flow overhead (e.g., TCP SYN, ACK, FIN, and RST segments). A TCP flow's limits for SYN, FIN, and RST segments are small constants. In our experiments, we use the value 10. A TCP flow's limit for ACK segments is equal to the ceiling of the reverse flow's count of segments times ϕ_2, where $\phi_2 \geq 1/b$ is a safety margin. In our experiments, we use $\phi_2 = 1.125$.

Greedy or malicious users might attempt to **circumvent**

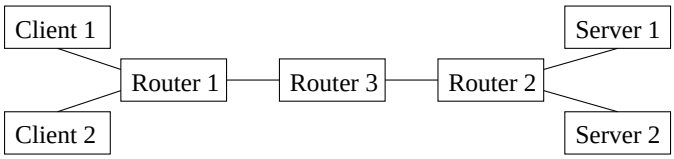

Fig. 2. Experimental setup.

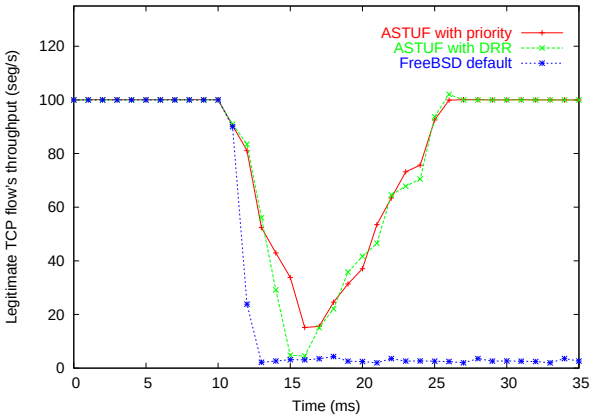

Fig. 3. Throughput of legitimate TCP flow from client 1 to server 1, with concurrent DoS attack from client 2 to server 2, starting at $t = 10$. Unlike FreeBSD's default protocol stack, ASTUF segregates the DoS flow and enables throughput of the legitimate TCP flow to recover.

limits on flow data transmission rate or control overhead by creating a large number of connections. Therefore, ASTUF access routers also limit the number of connections in the TCP-friendly CoS between a client and a server to a configurable value L_f. ASTUF access routers also monitor such connections' activity, and mark for eviction a connection that has been inactive for longer than a configurable interval T_i. If a client requests a new TCP-friendly connection C_2 (e.g., sends a TCP SYN segment) that would exceed L_f, and there is another connection C_1 between the same endpoints marked for eviction, then the ASTUF access router deletes C_1's state and starts monitoring C_2; otherwise, the router does not keep state for C_2. An ASTUF access router marks for the TCP-unfriendly CoS any packet whose flow the router does not have the state of. (For example, if C_1 resumes transmission after its eviction, then it will be forwarded in the TCP-unfriendly CoS.) Appropriate values for L_f and T_i may depend on the server. In our experiments, we use $L_f = 10$ and $T_i = 5$ minutes.

IV. Experimental results

This section reports results of experiments performed to evaluate ASTUF.

We implemented ASTUF on FreeBSD's TCP/IP protocol stack. FreeBSD's stack supports natively only one CoS. We modified the stack such that there are two IP input queues and two network driver output queues, separating TCP-friendly and TCP-unfriendly classes of service. We could not duplicate the

hardware transmit FIFO; to **reduce interference between the two classes of service**, we configured this **FIFO to hold at most only a few** (in our experiments, 4) packets. We also added priority-based and DRR scheduling options to the IP input and network driver output dequeue operations. Finally, we added options to configure a system as an ASTUF access router, including lists of incident access links and of subscribing servers. These options implement the packet marking algorithms described in Section III-D. The per-connection state required by this implementation is 164 bytes.

In our experiments, we connected two clients C_1 and C_2 to an ASTUF access router R_1 and two servers S_1 and S_2 to another ASTUF access router R_2, as illustrated in Fig. 2. In this testbed, all nodes are PCs running FreeBSD and all links are FastEthernet crossover cables operating at 100 Mbps. The TCP maximum segment size (MSS) was set to 1448 bytes.

We implemented a well-behaved TCP application L that sends about 100 MSS segments per second from C_1 to S_1 if the network is otherwise idle, but backs off according to TCP's congestion control rules if the packet-loss event rate increases. We also implemented a DoS attack application A that transmits over 8,000 TCP MSS segments per second from C_2 to S_2, without TCP's error, flow, or congestion control mechanisms and regardless of round-trip time or packet-loss event rate. We set A to start only well after L reaches steady state.

Fig. 3 shows L's throughput, in segments per second, when R_1 and R_2 were configured with the default FreeBSD stack, ASTUF with priority-based scheduling, or ASTUF with DRR scheduling. In the latter case, TCP-friendly and TCP-unfriendly classes of service were both configured with quantum equal to 1000. The figure shows that after the DoS attack starts, throughput of the legitimate TCP connection drops precipitously. With the default FreeBSD stack, there is no automatic recovery, and the legitimate TCP flow continues to have very little throughput as long as the attack continues. With ASTUF, however, the access routers eventually detect that A is TCP-unfriendly and mark A's segments accordingly. Because A's segments are then segregated in another CoS, L's throughput recovers to its pre-attack level, even though the attack continues. Similar recovery is observed regardless of the particular scheduling algorithm used (priority or DRR).

To measure ASTUF's overhead, we replaced L by TTCP, a well-known TCP benchmarking utility. Table I shows the TTCP throughput between C_1 and S_1, with or without concurrent DoS attack between C_2 and S_2 and with or without R_1 and R_2 configured as ASTUF access routers with priority scheduling. The table shows that without the DoS attack, ASTUF reduces TTCP throughput only slightly (about 2%). This overhead is due to ASTUF's TCP option, which is inserted, carried, and processed in each TCP segment. As the table also shows, this overhead pays off handsomely when the network is under attack: with ASTUF, the DoS attack reduces TTCP throughput by less than 10%, while without ASTUF, the DoS attack reduces TTCP throughput by more than 99%. ASTUF has moderate impact on the access routers' CPU utilization, increasing the latter from 26.3% to 39.4% under DoS attack.

TABLE I

ASTUF IMPOSES LITTLE OVERHEAD (2%) ON THROUGHPUT OF TCP-FRIENDLY FLOWS AND CAN PROTECT THEM FROM DOS ATTACKS ($> 90\%$)

Access router configuration	TTCP throughput (KB/s)		CPU utilization with attack
	no attack	with attack	
FreeBSD default	10716	82	26.3%
ASTUF/priority	10487	9455	39.4%

TABLE II

DRR WITH EQUITABLE QUANTA PREVENTS STARVATION OF TCP-UNFRIENDLY FLOWS, BUT GUARANTEES LESS BANDWITH FOR TCP-FRIENDLY FLOWS UNDER ATTACK

Scheduling (quanta: TCP-friendly, TCP-unfriendly)	TTCP throughput [σ] (KB/s) under DoS attack
Priority	9455 [22]
DRR (2000, 50)	9106 [63]
DRR (2000, 500)	7562 [21]
DRR (3000, 1000)	7208 [24]
DRR (1000, 1000)	5191 [8]

To better characterize ASTUF performance with priority or DRR scheduling, we measured TTCP throughput with DoS attack and various scheduling settings. Averages and standard deviations of ten repetitions are shown in Table II. The table shows that DRR performance can be very similar to that of priority scheduling if the quantum of the TCP-friendly CoS is much larger than that of the TCP-unfriendly CoS. However, DRR with such a setting may nearly starve UDP, ICMP, and other TCP-unfriendly flows (as does priority scheduling). With more equitable quantum settings, DRR avoids such starvation, but does not guarantee as much bandwidth to TCP-friendly flows when the network is under attack (e.g., if the quota are the same for both classes, ASTUF saves only about 50% of the bandwidth to TCP-friendly flows under DoS attack).

V. RELATED WORK

Floyd and Fall have proposed that congested routers attempt to identify TCP-unfriendly flows and preferentially drop packets from such flows so as to encourage conformance to TCP congestion control [10]. Their scheme assumes that round-trip times and packet-loss rates of congested links approximate the corresponding end-to-end values, and uses such measures to estimate the maximum transmission rate that would be expected in a TCP-friendly flow under such conditions. They use such estimates to regulate (by packet dropping) the flows that consume the most bandwidth in a congested router. Their approach differs from ours in several important ways. First, it requires modifications in backbone routers, which may be less feasible than modifying access routers. Second, its round-trip time and loss rate measurements may more significantly differ from end-to-end values, resulting in less precise control. Third, it controls only the heaviest flows. Fourth, attackers may evade

detection simply by using more connections and/or control segments. Fifth, it does not compensate service providers.

Brustoloni has proposed VIPnet, whereby e-commerce sites pay service providers to carry traffic of the sites' most important clients in a privileged class [18]. To send packets to a subscribing site in such a class, a client needs to have a VIP right issued by the site. VIP rights are time- and usage-limited, and need to be renewed after expired. A site may choose to renew VIP rights only of well-behaved clients that bring in the most revenue. VIPnet's strategy for incremental deployment and service provider compensation is similar to that of ASTUF. VIPnet can protect any transport-layer protocol, but is most appropriate for e-commerce applications, whereas ASTUF can protect only TCP-friendly protocols, but is appropriate for a much wider range of applications.

Yaar, Perrig, and Song have proposed SIFF, a scheme whereby routers on the path between a client and a server verify or modify a capability field in packets' headers [11]. Each router's key changes periodically, causing the path's capability to change. A server returns the current capability to a client only if the client is well-behaved. Routers drop packets whose capability is incorrect, and forward packets with correct capability with priority strictly higher than that of packets without capability. SIFF's use of capabilities and classes of service makes it similar to VIPnet. However, SIFF differs from VIPnet and ASTUF in several important ways. First, SIFF requires modifying backbone routers, whereas VIPnet and ASTUF modify only access routers. Second, SIFF can directly cause packet dropping and therefore its parameters' tuning may be more critical. Third, SIFF uses a priority scheme that may starve packets without a capability. Fourth, a SIFF client may experience great difficulty to get authenticated and get an initial capability if the server is under attack, whereas a VIPnet client gets authenticated by its access router and therefore can more easily break through such attacks (ASTUF clients do not need authentication and therefore may be less prone to such difficulties). Fifth, SIFF does not compensate service providers. Sixth, SIFF can be ineffective if attackers have control of a host near a victim (e.g., S_2 in Fig. 2). In SIFF, the colluding host can send capabilities to an attacker that can then clog links shared by the victim and colluding host. On the contrary, in VIPnet and ASTUF, flows destined to a colluding host are segregated in a separate class of service if the colluding host is not a subscribing server, or, in ASTUF, if the flows are not actually TCP-friendly. If the colluding host is a VIPnet or ASTUF subscribing site, the attack is also limited by the attacker's need to keep paying service providers during the attack, while preventing tracing of such payments.

VI. Conclusions and future work

Greedy or malicious users often disregard TCP congestion control in order to maximize their own performance or deny service to other users. We described Automatic Segregation of TCP-Unfriendly Flows (ASTUF), a novel scheme whereby access routers measure certain parameters of subscribing servers' flows and use these measurements to verify such flows' TCP-friendliness. Flows that declaredly (e.g., UDP or ICMP) or measurably disregard TCP congestion control rules are forwarded in a class of service separate from that of subscribing servers' verified TCP-friendly flows. Therefore, ASTUF protects the latter from many vulnerabilities of the current Internet. ASTUF is incrementally deployable and enables service providers to recover the costs involved in its deployment. We implemented ASTUF and evaluated it on an emulated network testbed. Experimental results show that ASTUF is effective and imposes little overhead. Future work should include replicating these results on larger simulated or production networks for more extensive validation.

Acknowlegments

Part of this research was performed while the authors were with Networking Software Research Department, Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733. José Brustoloni was partly supported by NSF grant ANI-0325353.

References

[1] J. Postel, "Transmission Control Protocol," RFC 793, IETF, Sept. 1981.
[2] V. Jacobson, "Congestion avoidance and control," in *Proc. SIGCOMM'88*, ACM, Aug. 1988.
[3] B. Braden et al., "Recommendations on queue management and congestion avoidance in the Internet," RFC 2309, IETF, Apr. 1998.
[4] D. Sisalem and H. Schulzrinne, "The Loss-Delay Adjustment algorithm: a TCP-friendly adaptation scheme," in *Proc. Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Cambridge, UK, July 1998.
[5] M. Handley, S. Floyd, J. Padhye and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448, IETF, Jan. 2003.
[6] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *Proc. SIGCOMM'01*, ACM, 2001, pp. 15-26.
[7] P. Ferguson and D. Senie, "Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing," RFC 2827, IETF, May 2000.
[8] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *Proc. SIGCOMM'2000*, ACM, 2000, pp. 295-306.
[9] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer, "Hash-based IP traceback," in *Proc. SIGCOMM'01*, ACM, 2001.
[10] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, 7(4):458–472, Aug. 1999.
[11] A. Yaar, A. Perrig and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *Proc. Security and Privacy Symposium*, IEEE, 2004.
[12] Q. He, C. Dovrolis and M. Ammar, "On the Predictability of Large Transfer TCP Throughput," in *Proc. SIGCOMM'05*, ACM, Aug. 2005.
[13] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," RFC 2475, IETF, Dec. 1998.
[14] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Transactions on Networking*, 4(3):375–385, June 1996.
[15] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proc. SIGCOMM'98*, ACM, Aug. 1998.
[16] M. Goyal, R. Guerin and R. Rajan, "Predicting TCP Throughput From Non-invasive Network Sampling," in *Proc. INFOCOM'2002*, IEEE, 2002.
[17] J. Mogul and S. Deering, "Path MTU discovery," RFC 1191, IETF, Nov. 1990.
[18] J. Brustoloni, "Protecting electronic commerce from distributed denial-of-service attacks," in *Proc. 11th Int'l World Wide Web Conf. (WWW2002)*, ACM, May 2002. [Online] `http://www2002.org/CDROM/refereed/528/`