The Taming of the Shrew

Srinivas Krishnan Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
{krishnan, jeffay} @cs.unc.edu

Abstract — The Shrew attack is a denial of service attack wherein a rogue end-system periodically generates a high-bandwidth "spike" in order to cause TCP senders to experience loss simultaneously, synchronize their retransmissions, and ultimately experience congestive collapse. Because these spikes are periodic, overall the Shrew is a low bandwidth flow and difficult to detect and police. Currently, the best known counter-measures for Shrew attacks require modifications to the protocol stack on end-systems. In this work we show how a form of active queue management (AQM) can effectively eliminate the threat posed by Shrew attacks. We show that Differential Congestion Notification, an AQM scheme developed to improve response time of interactive traffic, "tames" the Shrew. Using a laboratory network testbed emulating traffic sources found on a campus link, we show that DCN eliminates the ill effects of a variety of Shrew attacks including a distributed attack. This study demonstrates that while Shrew attacks are indeed a real threat, such attacks are naturally countered by using AQM schemes that already improve the performance of the majority of TCP flows.

1. Introduction

The Shrew attack is a denial of service attack developed by Kuzmanovic and Knightly [11]. In a Shrew attack, a source periodically generates short bursts of packets in order to saturate an upstream router. The goal is to keep the router's queue full for a sufficient duration as to ensure that a large number of flows experience a packet drop over a short interval. As explained in more detail in Section 2, a successful Shrew can cause TCP flows to synchronize their response to packet loss and remain in slow-start with a small window size. This ensures these flows transmit at a minimal rate independent of the actual available capacity in the network. A Shrew therefore has the potential to cause TCP senders to experience a form of congestion collapse even though the network does not appear congested.

Because a Shrew transmits its bursts for short durations, overall the Shrew can be a low bandwidth flow that is hard to detect and police. At present, the best defense against a Shrew attack is to modify TCP end-systems to randomize the initial value of their retransmission timer [11, 14]. The developers of Shrew also investigated the use of active queue management (AQM) to police Shrew attacks but concluded these methods were less than effective. More computationally expensive schemes based on a frequency domain analysis of requests have been applied at a server to ameliorate the effects of a related attack known as a reduction-in-quality attack (RoQ) [12].

This work revisits the use of AQM to police Shrew attacks. We show that that an existing AQM scheme developed to provide better response time performance to interactive flows, effectively "tames" the Shrew. The AQM scheme, called Differential Congestion Notification (DCN), heuristically classifies flows by duration and rate and polices high-bandwidth, longer duration flows more aggressively than low bandwidth, short duration flows [13]. Whereas previous attempts to police Shrew attacks with other forms differential AQM were not effective, we show empirically that DCN renders Shrew attacks harmless for the vast majority of TCP flows.

Using a laboratory network testbed, we emulate a peering point between two ISPs. Synthetic traffic is generated on the testbed to faithfully emulate the full range of TCP-based applications and services found on a campus network. In this environment it is first shown that a Shrew attack is easy to launch and is highly effective. Next it is shown that DCN AQM applied at the routers connecting the ISPs eliminates the threat posed by the Shrew for the vast majority of the flows. Moreover, DCN AQM polices the Shrew while still meeting its original design goal of improving response times for short duration/low rate flows. Finally, we show that a variety of modified Shrew attacks designed to circumvent DCN (*i.e.*, redesigns of the Shrew made with the knowledge that DCN would be employed against them), are also ineffective.

This work makes the following contributions. First, we demonstrate the effectiveness of a Shrew attack in a real (unprotected) network under a realistic workload at higher speeds than have previously been demonstrated. Second, we develop a number of new distributed Shrew attacks that are harder to detect than the original Shrew. These attacks represent a tradeoff between effectiveness — they are less effective because of their requirement for precise distributed control and detectability.

We show that in all cases, DCN detects and effectively polices the Shrew(s). In total, this work shows that while Shrew attacks are a real threat, counter to previous results, AQM can be most effective in controlling the attack. We believe this result is significant as it implies that end-systems need not be modified in order combat the Shrew.

The remainder of the paper makes the case for differential congestion notification as an effective counter measure against Shrew attacks. Section 2 presents the Shrew in more detail and describes previous counter measures. Section 3 describes DCN. Section 4 presents our experimental methodology and Section 5 gives the results of using DCN against a variety of Shrew attacks. We conclude with a discussion of these results in Section 6.

2. Background and Related Work

TCP infers congestion from packet loss events and adapts by decreasing its transmission rate aggressively. Loss is detected through two mechanisms that operate on different time scales. First, cumulative ACKs can signal a loss event and this feedback can be provided on the order of a round-trip time (RTT), typically on the order of 100 *ms* or less. Second, loss can be inferred from the lack of an ACK over a period of time covered by a retransmission timer (RTO), typically on the order of seconds. Shrew targets the coarser granularity of the RTO timer to create outages.

In a Shrew attack the attacker sends out “spikes” (bursts) of unresponsive traffic at a set interval. Each traffic spike is meant to overflow a router’s buffers and cause newly arriving packets to be dropped. This causes an artificial signal of congestion to eventually be received at end-systems that experience a loss. Ideally (from the perspective of the attacker), the end-system will wait for the RTO timer to expire before retransmitting the lost packet. The period between each Shrew spike is the expected time an end host waits before retransmitting the previously lost packet. When the end-system retransmits the packet, the Shrew source generates another traffic spike causing the router buffer to again overflow. If the retransmitted packet is also lost then the end-system doubles its RTO value. In this manner, a Shrew attacker can throttle an end-system to a near zero transmission rate. As the RTO timer increases exponentially the end-system throughput goes to zero.

The success of Shrew depends on estimating the RTO value and causing losses at this timescale, thereby preventing the end machine from exiting the timeout state. If a Shrew causes most of the TCP sources on a link to drop packets, then the Shrew will also be successful in synchronizing the sources and create a self-sustaining attack. This is particularly true in networks with a large population of machines with homogeneous TCP implementations and a common minimum RTO value.

Kuzmanovic and Knightly have successfully demonstrated the efficacy of Shrew attacks using ns simulations for a 10Mbps link with both long-lived TCP flows and HTTP traffic. The scale of these experiments however is quite small, as they primarily concentrated on effects to a single flow. They were able to throttle the sender to less than 8% of the original throughput achieved without a Shrew attack. They have also shown that RTT variance will cause the performance of shorter flow to be worse than longer flows.

Two forms of counter measures were proposed for a Shrew attack: a router-based AQM and modifications to TCP implementations on end-systems [11]. For AQM schemes, RED-PD and CHOKe (described below) were considered and found in simulation to be less than completely effective. Both reduced the effectiveness of the Shrew attack but not to levels that are likely to be acceptable to an Internet user.

The proposed modifications to TCP implementation were increasing initial window size and randomizing the minimum RTO value (minRTO). Increasing the window size prevents bias against shorter flows, as the probability of a loss for a short flow is much higher when Shrew is operating. Randomizing minRTO has a considerable effect in mitigating a Shrew attack but it still does not remove the vulnerability of a given flow to the attack.

An alternate approach to detecting a Shrew is based on a frequency-domain analysis of packet arrivals [15]. Here, a Shrew is detected by classifying every arriving packet based on the flow’s history. The behavior of each flow is monitored by counting packet arrivals over a given set of intervals and converting each time series into the frequency domain via a Discrete Fourier Transform. This method was applied at server machines (ideally equipped with special purpose DSPs) and is not applicable for use in routers. This method is computationally intensive and requires the maintenance of per-flow state.

In summary, we are unaware of any effective network-based means of detecting and policing a Shrew attack. Changes to host protocol stacks will be effective but is a not a practical solution as it will change the behavior of TCP. Moreover, it is unknown if there is a sufficient diversity in the range of minRTO values that can be applied in networks consisting of tens of thousands of hosts. Such solutions will also not help UDP-based applications. For these reasons we are motivated to investigate router-based counter measures.

3. Differential AQM

Active queue management refers to the process of running a decision procedure when a packet arrives at a router to determining whether or not to "mark" the arriving packet. Originally, the goal of AQM was to provide end-systems with an "early" indication that a router was becoming congested. For example, the common implementation of marking is to simply drop (discard) the packet. A responsive end-system will eventually detect the loss and (ideally) reduce its transmission rate. If a router does a good job of marking packets then enough sources will reduce their rates and the router will never actually become congested.

Differential AQM is a means of approximating packet scheduling in a router. In this case flows are typically classified heuristically and different decision procedures are run on packets from different classes. In this manner, differential AQM provides a means for routers to provide different classes of service to classes of flows. Classes of flows receive different service by being more or less immune to the effects of packet marking during times of incipient congestion.

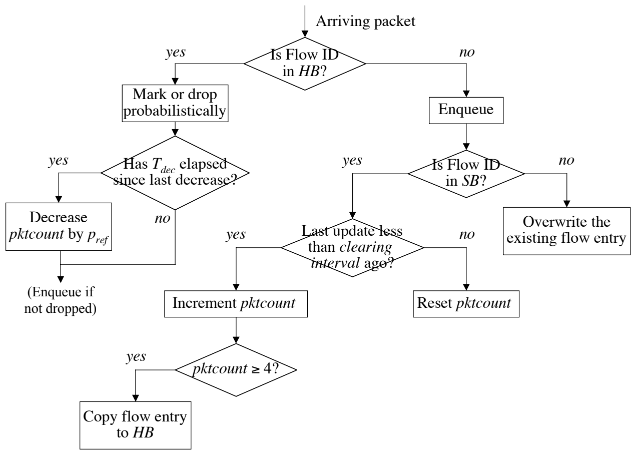
Arriving packet
yes
Is Flow ID in *HB*?
no
Mark or drop probabilistically
Enqueue
yes
Has T_{dec} elapsed since last decrease?
yes
Is Flow ID in *SB*?
no
Decrease *pktcount* by p_{ref}
no
yes
Last update less than *clearing interval* ago?
no
Overwrite the existing flow entry
(Enqueue if not dropped)
Increment *pktcount*
Reset *pktcount*
yes
pktcount ≥ 4?
Copy flow entry to *HB*

Figure 1: High-level DCN flowchart.

The design of DCN is based on the observation that on many networks, a small number of flows produce a large percentage of the traffic. For example, recent measurement studies have shown that while approximately 85% of flows transmit 100K bytes or less, these flows account for less than 15% of the bytes transferred across a link [16]. Flows that are "large" and "fast" (*i.e.*, high-bandwidth, greater than 10KB/sec) account for less than 10% of all flows, but carry more than 80% of all the bytes.

These data suggest that providing early congestion notification to short flows would have little effect on congestion. These flows often are too short to have a transmission rate that is adaptable. By the time they receive a congestion notification signal they have either already completed or have only a small number segments remaining to be sent. Furthermore, since short flows have a small congestion window, they have to resort to timeouts when experiencing a packet loss. Thus giving these flows a congestion signal does not significantly reduce congestion and can only hurt the flows' performance by significantly delaying their completion. In contrast, high-bandwidth flows carrying large amount of data are capable of reducing their transmission rate and hence can have an impact on congestion. Unlike short flows, high-bandwidth flows do not have to resort to timeouts and instead can use TCP mechanisms for fast retransmission and fast recovery to recover from their packet losses.

DCN is a form of differential AQM that was previously shown to improve the response time for more than 90% of all flows while effectively increasing the overall efficiency of the network (by providing early congestion indications) [13]. In particular, DCN was shown to provide significant performance improvements over other differential AQM schemes such as CHOKE and RED-PD.

DCN's approach to identifying high-bandwidth, long-lived flows is based on the idea that packets of high-bandwidth flows are closely paced (*i.e.*, their interarrival times are short) [3]. DCN tracks the number of packets that have been recently seen from each flow. If this count exceeds a threshold, the flow is considered to be a "long-lived and high-bandwidth" flow. The flow's rate is then monitored and its packets are eligible for dropping. As long as a flow remains classified as high-bandwidth, it remains eligible for dropping. If a flow reduces its transmission rate, it is removed from the list of monitored flows and is no longer eligible for dropping.

DCN uses two hash tables for classifying flows: HB ("high bandwidth") and SB ("scoreboard"). The HB table tracks flows that are considered high-bandwidth and stores each flow's flow ID (IP addressing 5-tuple) and the count of the number of forwarded packets. The SB table tracks a fixed number of flows not stored in HB. For these flows SB stores their flow ID and "recent" forwarded packet count.

When a packet arrives at a DCN router, the HB table is checked to see if this packet belongs to a high-bandwidth flow. If the packet's flow is found in HB, then it is handled as described below. If the packet's flow ID is not in HB, then the packet is enqueued and its flow is tested to see if it should be entered into HB. The SB table is searched for the flow's ID. If the flow ID is not present, it is added to SB. If the flow ID is present in SB, the flow's packet count is incremented.

A flow is classified as long-lived and high-bandwidth if the number of packets from the flow arriving within a "clearing interval," exceeds a threshold. Once the flow's packet count in SB has been incremented, if the count exceeds the threshold, the flow's entry in SB is added to HB. If no packets have been received for the flow within a clearing interval, the flow's packet count is reset to 0.

A high-level flow chart of the DCN algorithm is given in Figure 1. All operations on the SB table are performed in $O(1)$ time. Since the number of flows identified as high-bandwidth is small, hash collisions in HB are rare for a table size of a few thousand entries. Thus, operations on the HB table are also usually executed in $O(1)$ time.

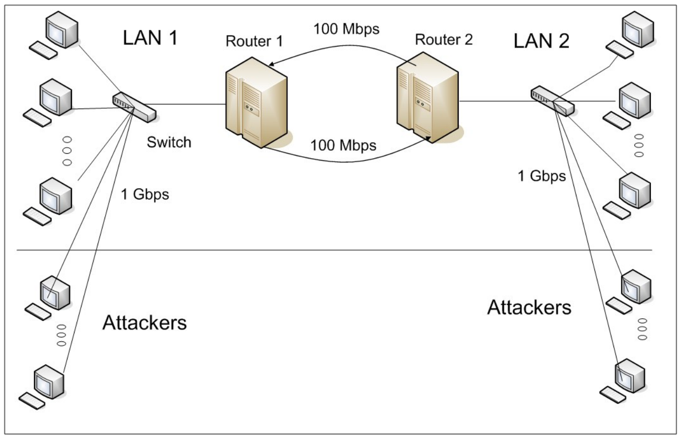

Figure 2: Laboratory testbed.

Packets from a high-bandwidth flow are dropped with a probability $1 - p_{ref} / pktcount$, where *pktcount* is the number of packets from that flow that have arrived at the router within a period of T_{dec}, and p_{ref} is the current "fair share" of a flow on a congested link. When congestion is suspected in the router we target high-bandwidth flows for dropping in proportion to their deviation from their fair share (p_{ref} packets within an interval T_{dec}) [16, 19].

DCN uses a simple control theoretic algorithm based on the well-known proportional integral controller to compute p_{ref}. The instantaneous length of the queue in the router is periodically sampled with period T_{update}. A flow's fair share of the queue at the k^{th} sampling period is given by:

$$p_{ref}(kT_{update}) = p_{ref}((k-1)T_{update}) + a \times (q(kT_{update}) - q_{ref}) - b \times (q((k-1)T_{update}) - q_{ref})$$

where a and b, $a < b$, are control coefficients (constants) that depend on the average number of flows and the average RTT of flows (see [7] for a discussion), $q()$ is the length of the queue at a given time, and q_{ref} is a target queue length value for the controller. Since $a < b$, p_{ref} decreases when the queue length is larger than q_{ref} (an indication of congestion) and hence packets from high-bandwidth flows are dropped with a high probability. When congestion abates, the queue length drops below q_{ref}, p_{ref} increases and the probability of dropping becomes low.

The flow ID of a high-bandwidth flow is kept in the HB table as long as the flow's counter *pktcount* is positive. After each interval T_{dec}, the counter *pktcount* is decreased by p_{ref}. If a high-bandwidth flow's packet count becomes negative, the flow is deleted from HB.

4. **Experimental Methodology**

The effectiveness of DCN in detecting and policing Shrew attacks was tested in a laboratory testbed. The testbed emulates a peering link between two Internet service provider (ISP) networks connected at either 1 Gbps or 100 Mbps. The testbed consists of approximately 50 Intel processor based machines running FreeBSD 4.5. Machines at the edge of the network execute one of a number of synthetic traffic generation programs described below. These machines have 100 Mbps Ethernet interfaces and are attached to switched VLANs with both 100 Mbps and 1 Gbps ports on 10/100/1000 Ethernet switches. At the core of this network are two router machines running the ALTQ extensions to FreeBSD [9]. ALTQ extends IP-output queuing at the network interfaces to include alternative queue-management disciplines. We used the ALTQ infrastructure to implement DCN. The efficiency of DCN to police flows depends on developing hashing function that will prevent state-space attacks. The hash-function that we used in our experiments prevented the onset of state-space attacks for hash table sizes of 16K and 8K.

Each router has sufficient network interfaces to create either a point-to-point 100 Mbps or 1 Gbps Ethernet network between the two routers. The Gigabit Ethernet network is used to conduct calibration experiments to benchmark the load induced by traffic generators on an unloaded network. To evaluate the Shrew and compare the effects of Shrew on a drop-tail FIFO and a DCN-controlled FIFO queue, we create a bottleneck between the routers by altering the (static) routes between the routers so that all traffic flowing in each direction uses a separate 100 Mbps Ethernet segment. This setup allows us to emulate the full-duplex behavior of a typical wide-area network link. A high-level view of the laboratory network is shown in Figure 2.

Synthetic Generation of TCP Traffic

A synthetically derived TCP workload will be used to study the effects of Shrew. The workload is derived from measurements of the full-range of TCP connections found in the UNC connections. Using the methods described in [10, 18], a multihour packet header trace is collected of all connections flowing on the link between the UNC campus and its upstream service provider. This trace is "reverse compiled" into a source level model of all the TCP connections flowing over the measured link. The model in essence captures the network invariant aspects of each connections behavior: the pattern of socket layer reads and writes performed by each end-point of the connection. This model is used as an input to synthetic traffic generation system called *tmix* that emulates the source-level behavior of each TCP application instance that was active during the measurement period. This allows us to effectively recreate the full range and complexity of application communication patterns found on a real network. In doing so we are freed from the need to make arbitrary decisions about the nature of synthetic traffic used in experiments such as the mix between short-lived and long-lived flows. Instead, we simply reproduce exactly the mix found on the measured link. The result is synthetically generated TCP traffic that matches statistics of real traffic such as file and object-size distributions, burstiness, and number of simultaneously active connections [10].

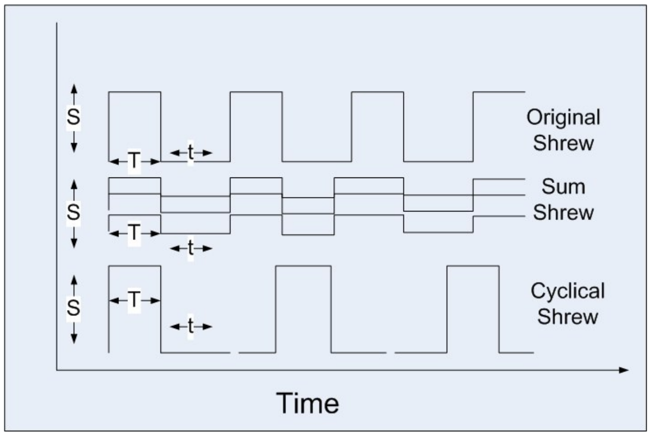

Figure 3: Patterns of Shrew attacks.

In addition to the source level model, we also acquire network path properties for each measured connection such as minimum RTT, and transport layer parameters such maximum window size. These parameters are used in the *tmix* traffic generation system on the end-systems to emulate per-flow path properties. Thus, while the laboratory network is ultimately a dumbbell network, each synthetically generated flow will have a unique minimum RTT derived from measurement data. By combining the source-level and network-level models, we can effectively emulate a wide-area network in our laboratory [10].

Generation of Shrew Attacks

To generate Shrew attacks we built a tool that could generate attacks using 1 to n_1 processes on 1 to n_2 hosts. In the case of either multiple processes or multiple hosts, two forms of Shrew were possible: a "cyclical Shrew" and a "sum Shrew." These are described in more detail below. In general, we experimented with 5 types of Shrew attacks:

- A single process on a single machine,
- A cyclical Shrew using multiple processes on a single machine,
- A sum Shrew using multiple processes on a single machine,
- A cyclical Shrew using multiple machines machine, and
- A sum Shrew using multiple machines.

The main goal of the tool is to generate traffic spikes in a waveform fashion as described in [11]. This is achieved by using UDP traffic as illustrated in Figure 3. In Figure 3, t represents the period of time in between each spike and T is the duration of a spike. The sum of $t+T$ should equal the estimate of the minRTO value on the victim machines.

The core of the Shrew attack depends on estimating the end systems minRTO value. This requires tight co-ordination amongst the Shrew client machines and synchronization of their clocks. Thus the simplest, and most accurate Shrew attack is achieved by using a single process on a single machine. We term this Shrew variant the *single source Shrew attack*. This attack should generate the most well-timed spikes and hence be the most effective. The downside of this attack is that since it originates from one machine and one application (and hence one source port number), in principle this attack should e the easiest to detect.

To make the Shrew harder to detect can use multiple processes on a single machine. By using multiple processes the attack is spread out over multiple source port IDs and hence a flow classifier such as DCN that classifies solely based on flow ID will have a harder time identifying this Shrew attack. In addition, because all the processes execute on the same machine, timing and coordination is easy and hence the resulting attack should be effective.

Within this structure, two Shrew control paradigms were investigated: *cyclical Shrew* and *sum Shrew*. In the former case the processes take turns generating the spike in a round-robin fashion. In the latter case, all processes generate a smaller spike simultaneously such that the small spikes combine into an appropriately sized larger spike. These two forms of Shrew attacks are called *multithreaded cyclical Shrew* and *sum Shrew* respectively.

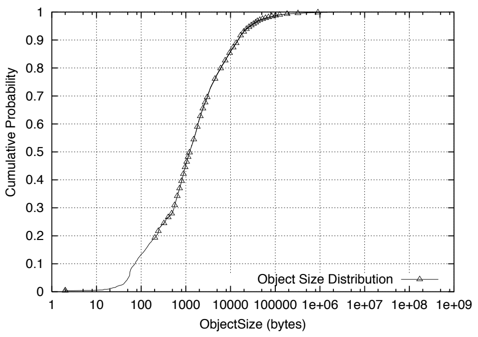

Figure 4: Distribution of bytes transmitted within request/response exchanges.

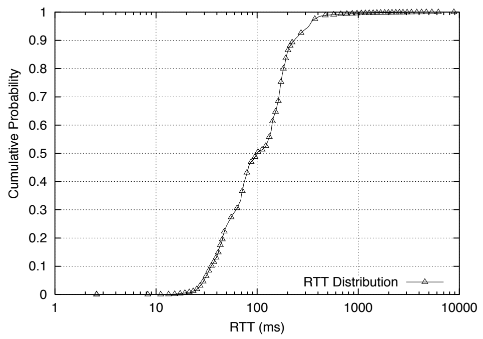

Figure 5: Distribution of minimum RTT of connections.

Another obvious method for making the Shrew harder to detect is to distribute the attack across a set of machines. This should make the attack hardest to detect, however, it almost makes it harder to perform the attack effectively because it introduces a distributed timing a control problem. We experimented distributed cyclical and sum attacks to test their effectiveness. In the following section these attacks are called *multi-machine cyclical* Shrew and *multi-machine sum* Shrew attacks.

Experimental Procedures

To evaluate Shrew we performed experiments on the two emulated ISP networks in our testbed connected with 100 Mbps links. Each experiment was run using an offered load of 90% of the capacity of the 100 Mbps link connecting the two router machines. We targeted our Shrew attacks to generate spikes of 30 Mbps of UDP traffic.

Our primary measure of performance is the distribution of response times experienced by TCP connections. Response time is defined as the time for an application-level “request/response” exchange to complete. More precisely, it is the time required for a TCP client to send an application-level data unit to a server and for the server to respond with a second application-level data unit. In general the synthetic connections we generate consist of a number of these request response exchanges. We will understand the effect of the Shrew attack on various types of connections by examining conditional cumulative distribution functions (CDFs) of response times. In our case we will separately consider the distribution of response times experienced by request/response exchanges transmitting 1-1K bytes, 1K-10K bytes, 10K-100K bytes, 100K-1M bytes, and exchanges transmitting more than 1M bytes.

To better understand the nature of the synthetic traffic we generate, Figure 4 shows the distribution of bytes transmitted across all request/response exchanges of all connections initiated in our experiments. As these data derive from measurements of actual TCP traffic, they exhibit the expected pattern of 90% of all exchanges transmit 10K bytes or less. For completeness, Figure 5 shows the distribution of minimum RTTs experienced by connections in our experiments. These data derive directly from measurements as well.

Each experiment was run for 120 minutes to ensure very large samples (over 10,000,000 TCP data exchanges), but data were collected only during a 90-minute interval to eliminate startup effects at the beginning and termination synchronization anomalies at the end.

5. Experimental Results

In this section we describe the results we obtained by comparing the effects of Shrew on a drop tail FIFO queue and a FIFO queue managed by DCN. In order to gather reference data, we ran the first set of experiments without Shrew traffic, with FIFO and DCN. Figure 6 describes the result, where we observe that without Shrew FIFO and DCN perform comparably.

The section is organized by the type of Shrew attack we ran: single source, multithreaded and multi-machine. In each plot we show the effect of Shrew on response time broken down by the amount of data transmitted during a request/response exchange.

Single Source Shrew Results

The plots in Figures 7-11 describe the effects of a basic Shrew attack, in which we use a single pair of machines sending out spikes of 30Mbps for a duration 200ms with 1s intervals. In Figure 7 we can see that ~99% of the request/response exchanges

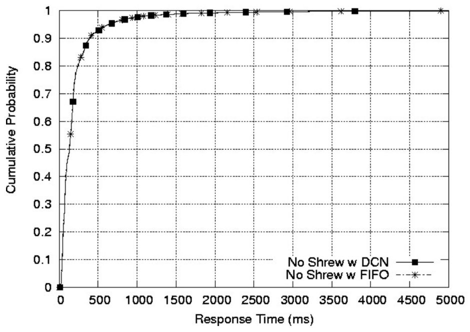

Figure 6: Response Time of DCN and FIFO with No Shrew traffic

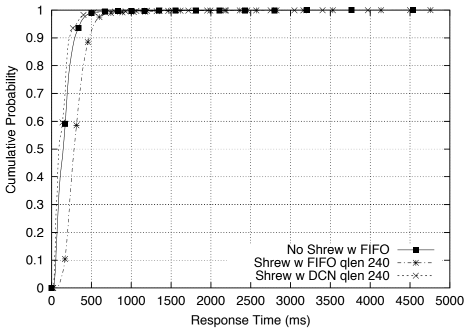

Figure 7: Response time for object sizes < 1K with single source Shrew attack.

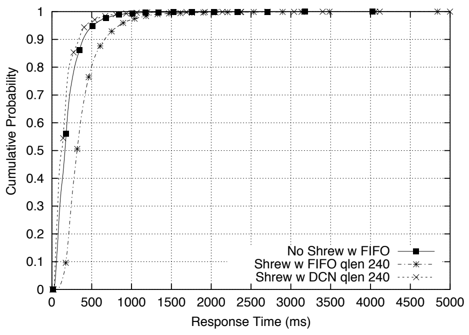

Figure 8: Response time for object sizes < 10K with single source Shrew attack.

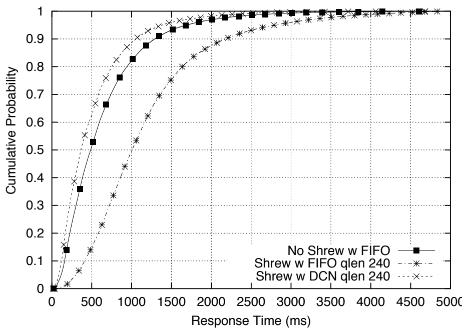

Figure 9: Response time for object sizes < 100K with single source Shrew attack.

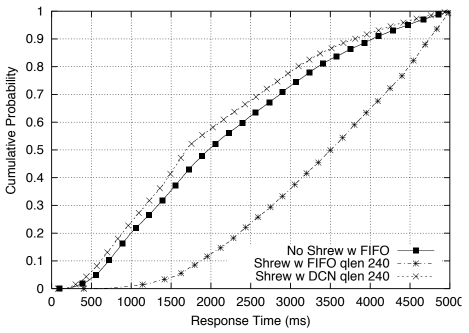

Figure 10: Response time for object sizes < 1M with single source Shrew attack.

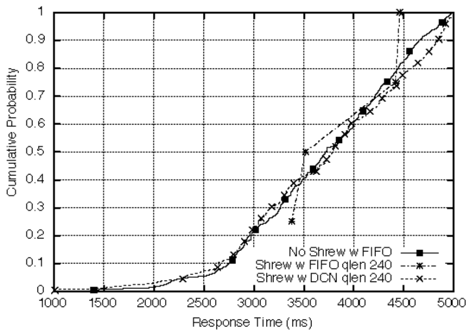

Figure 11: Response time for object sizes > 1M with single source Shrew attack.

transmitting less than 1K bytes have a higher response time during a Shrew attack when the router is using a drop-tail FIFO queue.[1] The response time for 90% of these objects is increased by over 100ms. However, DCN is able to police the unresponsive Shrew flow to allow objects 1K bytes or less to have similar or better response time to a network with no Shrew flows.

This trend is seen in Figure 8 as well, for objects of size 10K bytes or less. 98% of these objects experience significantly increased response times due to a Shrew attack. DCN is able to detect and throttle the Shrew flow and the achieved response times are even better than a network with no Shrew flows.

For larger objects, those of sizes up to 100K and 1M bytes, we see that the Shrew increases response times dramatically (Figures 9 and 10). 90% of the objects experience delays over 500ms. However, with DCN these objects get a fair share of the router buffer and again achieve significant performance improvements over a no-Shrew network. This trend is particularly striking in 10 where the transfer of larger objects experience clear congestion collapse in the face of a Shrew attack. Once again DCN is able to detect and police the Shrew flow and cause response times to be better than the non-Shrew case.

DCN is primarily meant to remove the bias against short flows and smaller objects, hence for objects greater than 1 MB, we see about 20% of the objects having a minor increase in response time ~20-40ms with DCN when compared to the no-Shrew case. The discrete nature of the Shrew with FIFO curve is due to a lack of connections finishing within 5000ms (there are only around 20 data points in this case). This is in contrast the DCN and non-Shrew cases where over 250 connections finish within 5000ms. This shows that the Shrew attack is particularly devastating to larger, longer-lived flows.

Multithreaded Shrew

Figures 12-16 show the results for the multithreaded Shrew attack. Recall that this attack is implemented with multiple processes on a single machine that coordinate their spike transmissions to emulate the behavior of the Single Source attack. Because DCN classifies connections solely based on IP flow ID (source/destination IP address/ source/ destination port number,

[1] For ease of presentation, hereafter we refer to individual request/response exchanges as simply "objects."

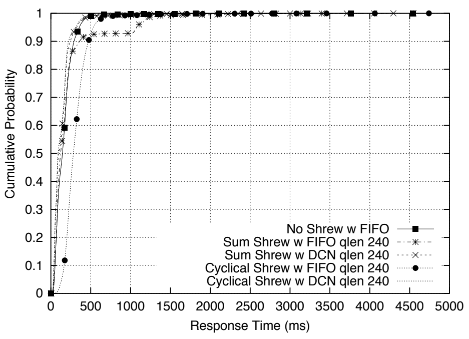

Figure 12: Response time for object sizes < 1K for multithreaded Shrew.

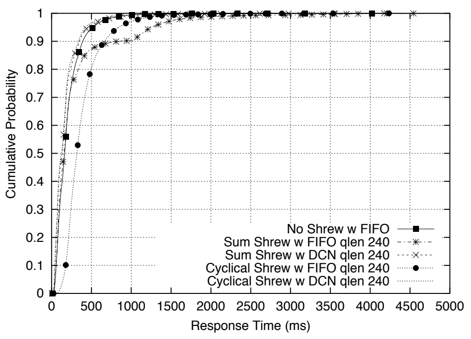

Figure 13: Response time for object sizes < 10K for multithreaded Shrew.

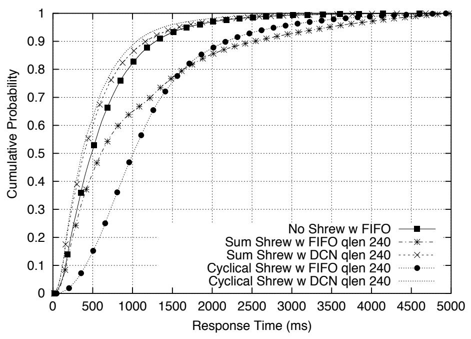

Figure 14: Response time for object sizes < 100K for multithreaded Shrew.

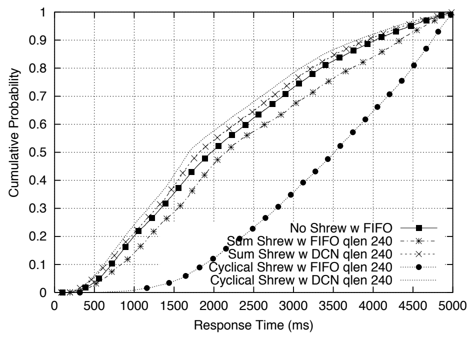

Figure 15: Response time for object sizes < 1M for multithreaded Shrew.

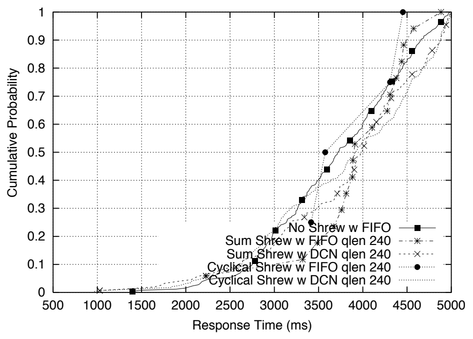

Figure 16: Response time for object sizes > 1M for multithreaded Shrew.

and protocol number), although these Shrew flows originate from the same machine, DCN will treat them as completely separate flows because they have different source port numbers. Although it would be trivial to modify DCN to bias its classification scheme to weight flow ID components such as source IP address more heavily, the current implementation does not do so. However, nonetheless, as the results show, DCN still effectively detects and polices the Shrew attack even when it is spread out over 10 processes generating 3 Mbps each.

In Figures 12-16, we show the effects of a Shrew attack from a single machine with 10 processes. The two types of attacks compared are the cyclical (round robin) Shrew and sum (cumulative) Shrew. As the threads share a common system clock, there are no complex synchronization issues in these attacks and hence they represent the best-case behavior for a "distributed" attack. One complexity here, however, is the fact that the processes share a common network interface which can induce some serialization effects in the case of the sum Shrew and stagger the individual spikes by a couple of milliseconds.

For the case of objects 1K bytes or less (Figure 12), we see a similar trend as was observed in single source Shrew attacks. For the cyclical Shrew we see a similar trend as before: 99% of the objects experience delays of ~50ms. The sum Shrew case

however is interesting with 90% of the objects experiencing no delays due to Shrew. The attack however is quite effective for the 20% of flows near the tail, increasing their response time by ~500ms. Note that in these and subsequent plots, DCN universally detects and polices the cyclical and sum attacks effectively. In fact DCN provides nearly identical response time distributions under both attacks for all object sizes. For this reason, in the remaining plots, where the "cyclical Shrew DCN" curve is not discernable on the plot, this is because it lies exactly on top of the "sum Shrew DCN" curve.

In Figure 13 we once again see cyclical Shrew affecting 10K objects with 80% of the objects having response times increased by approximately 50ms. In the sum Shrew once again only ~20% of the flows near the tail were affected, but the effect is quite pronounced with delays up to ~500ms. The 100K and 1M cases for the cyclical attack are similar to the response times described for the single source Shrew. The sum attack seems to lose potency as we increase the object sizes with 1M objects experiencing a uniform delay of ~50ms.

In all the cases we see a trend of cyclical Shrew to be more effective as each thread has sole control of the interface for the burst period of 200ms, while in the sum attack we have sharing of the interface by the multiple threads, which leads to the spikes being staggered and the cumulative effect is rarely equals 30Mbps at the router. The objects that were affected seem to

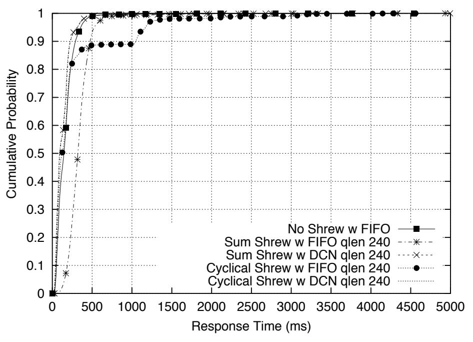

Figure 17: Response time for object sizes < 1K for multi-machine Shrew.

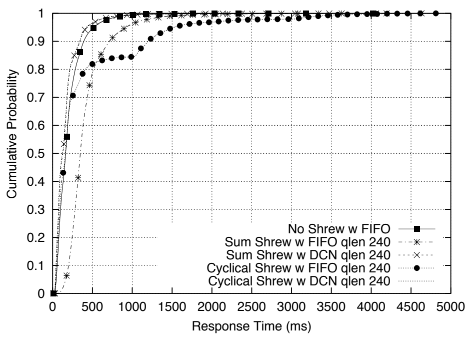

Figure 18: Response time for object sizes < 10K for multi-machine Shrew.

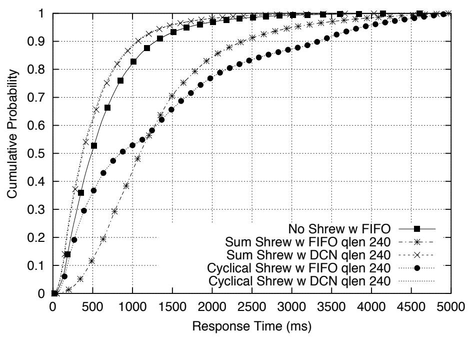

Figure 19: Response time for object sizes < 100K for multi-machine Shrew.

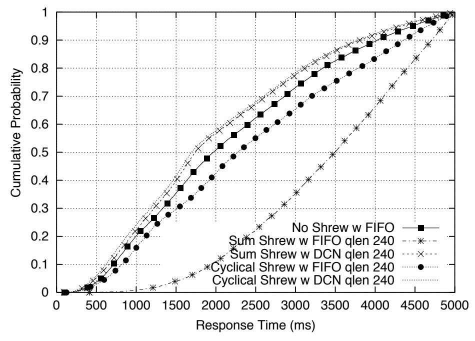

Figure 20: Response time for object sizes < 1M for multi-machine Shrew.

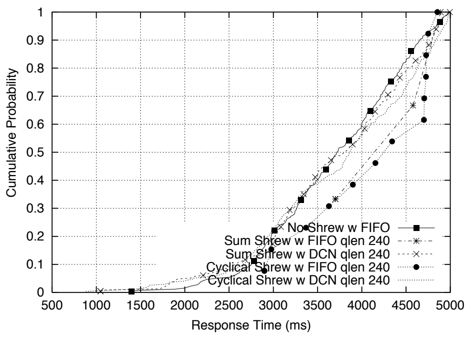

Figure 21: Response time for object sizes > 1M for multi-machine Shrew.

be short flows, whose RTO value was approximated by at least 1 cycle of attackers, increasing the response time by ~500ms for the 1K and 10K object size cases.

DCN however is able to detect and throttle the Shrew flows, as shown in Figures 12-15, leading to a comparable or better response time than a network with no-Shrew. However, for objects greater than 1MB, we observe that over 60% of the objects have response times elongated by ~500 ms with DCN for the cyclical attack and for the sum case the upper 30% of the objects experience delays between 50-200ms. This is inherent in the operation of DCN as it explicitly is attempting to provide lower response times to shorter flows (over 99% of all objects as seen from Figure 3) at the expense of a small number of very large flows. Also, due to the sum attack not being very effective we have 150 Shrew with FIFO connections finishing within 5000ms, almost a tenfold increase compared to previous cases of single source and cyclical (~25 for Shrew w FIFO compared to 250 for no-Shrew and 320 w DCN).

Multi-Machine Shrew Results

In this section we describe results from running a distributed Shrew attack with the cyclical and sum variants. A total of 3 machine pairs were used to simulate a distributed attack. In the multi-machine sum case, we have the 3 client zombies sending out spikes of 10Mbps each, cumulating to 30 Mbps. In the cyclical case we have the three machines sending out spikes in a round-robin as described in the previous section.

In Figures 17-21 we observe a trend opposite to what was observed for the multithreaded case. Here the sum attack causes more damage than it did in the cyclical attack. For all objects less than 1MB (Figures 17-20), the sum attack is as effective as the single source Shrew attack, with 99% of the 1K and 10K objects experiencing increase in response time over 20ms. In the 100K and 1M cases the Shrew with FIFO experiences ~500ms for over 70% of the objects. As the distributed cyclical depends on close synchronization between the *N* attackers, we see that for the 1K and 10K cases ~90% of objects for Shrew with FIFO were not affected by Shrew. The 100K objects that constitute 50% of the object size distribution have 99% of the objects with increased response times with a maximum of 500ms. In the case of 1M objects we have uniform increase in response times of ~ 50ms. This effect is similar to the multithreaded sum case where we had similar problems of spikes not able to estimate the minRTO value of the majority of flows. The objects that were once again affected were the short flows, whose RTO value was approximated by at least 1 cycle of attackers, increasing the response time by ~500ms, in the 1K and

	SingleSource	Sum Multi-threaded	Cyclical Multi-Threaded	Sum Multi-Machines	Cyclical Multi-Machines
FIFO	67	75	74	75	76
DCN	70	73	73	72	77
No Shrew	87	87	87	87	87

Table 1: Observed Throughput

	SingleSource	Sum Multi-threaded	Cyclical Multi-Threaded	Sum Multi-Machines	Cyclical Multi-Machines
FIFO	.1411	.1391	.128	.1167	.121
DCN	.0696	.0567	.0645	.0612	.0512
No Shrew	.003	.003	.003	.003	.003

Table 2: Loss rates

10K cases. Due to the synchronization problems any staggering between the Shrew flows will cause the attackers to not send out spikes during the period when the flows which experience losses during the previous spike are coming out of a timeout.

The success of the sum Shrew attack is due to the fact the interfaces are not shared. Thus unlike the multithreaded sum attack case, we do not have the staggering of flows. Also as each flow is guaranteed 10Mbps, only a coarse granularity is needed for the arrival of these flows at the router to cause the router buffer to overflow. Once again DCN detects and polices all the Shrew flows, and the response times are comparable or better than a network with no Shrew. In the case of object sizes greater than a MB, over 60% of objects for the cyclical attack with DCN have response time increases between 10ms-50ms.

Network Centric Performance Results

Finally, we comment on some network centric performance results such as achieved throughput and loss rates. In all experiments, as the Shrew flow(s) were policed, the bandwidth consumed by these flows was consumed instead by the TCP flows. This is a reflection of the stressful environment the experiments were conducted in, where link utilization was already high. For these reasons, we have chosen to emphasize response time distributions as our primary performance measure. Response times show the effects on individual classes of flows and captures the aspect of performance that is likely most important to the user. Data on loss rates across all the experiments is given in Table 2. The loss rate ranges from as high as 14% for drop tail FIFO under the single-source Shrew attack to 3% without the Shrew attack. This again indicates that we are operating in a highly utilized network environment. It must be noted that these loss figures are from data gathered at the router and represent total loss from all sources (responsive and non-responsive sources combined).

DCN reduces the loss rate to 6% during a Shrew attack. As these rates include packets dropped from the TCP flows and the Shrew flows, we cannot draw any direct conclusions from them. However, when looked at in conjunction with the response times, we can conclude that a majority of packets dropped by DCN were the Shrew flows. This confirms the efficacy of DCN in detecting and throttling Shrew flows. The various attempts to change the flow ids to “hide” from DCN, by using a multi-threaded and distributed attack are also detected by DCN, as we can see from the reported loss rates for the respective attacks.

6. Summary and Conclusions

The Shrew attack has the potential to adversely affect all flows by synchronizing senders’ responses to congestion. In particular, the longer the flow, the more seriously the flow will be affected. We have shown the potential of a Shrew attack to be detrimental on a scale not previously demonstrated. The experiments we conducted show that on a 100 Mbps link connecting two ISPs, and carrying realistic TCP traffic representing the full variety of application traffic found on a major university campus, the Shrew can cause congestion collapse for long-lived flows.

Moreover, we have shown that it is possible to detect and police a variety of Shrew attacks using an existing AQM scheme designed to provide differential congestion notification to flows based on their assumed rate and duration. The algorithm, DCN, detects and polices single and multi-source Shrew attacks and in all cases, ameliorates its ill-effects for more than 99% of the TCP flows present, while achieving its original design goal of providing better response time performance for short flows (those that have request/response exchanges of 100K bytes or less.)

These results have been demonstrated in a laboratory testbed under realistic workloads and traffic conditions and as such, we believe, represent the most comprehensive study of the Shrew attack and counter measures to date. In conclusion, this work demonstrates that the previous best counter measure for the Shrew, the modification of protocol stacks on end-systems can be augmented or replaced with a router-based AQM approach.

7. References

[1] D. D. Clark and W. Fang, *Explicit Allocation of Best-Effort Packet Delivery Service*, IEEE/ACM Trans. on Networking, vol. 6(4), pp. 362-373, August 1998.

[2] C. Estan and G. Varghese, *New Directions in Traffic Measurement and Accounting*, Proc., ACM SIGCOMM 2002, Aug. 2002, pp. 323-336.

[3] W. Feng, D. D. Kandlur, D. Saha, and K. G. Shin, *Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness*, Proc., IEEE INFOCOM 2001, April 2001, pp. 1520-1529.

[4] L. Guo and I. Matta: *The War between Mice and Elephants*, Proc., ICNP 2001, Nov. 2001, pp. 180-188.

[5] C.V. Hollot, Vishal Misra, Don Towsley, and W. Gong, *On Designing Improved Controllers for AQM Routers Supporting TCP Flows*, Proc., IEEE Infocom 2001, pp. 1726-1734.

[6] L. Le, J. Aikat, K. Jeffay, F. D. Smith, *The Effects of Active Queue Management on Web Performance*, Proc., ACM SIGCOMM 2003, Aug. 2003, pp. 265-276.

[7] R. Mahajan, S. Floyd, and D. Wetherall, *Controlling High-Bandwidth Flows at the Congested Routers*, Proc., ICNP 2001, pp. 192-201.

[8] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, *Approximate Fairness through Differential Dropping*, ACM CCR, April 2003, pp. 23-39.

[9] R. Pan, B. Prabhakar, and K. Psounis, CHOKE, *A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation*, Proc., IEEE INFOCOM 2000, March 2000, pp. 942-951.

[10] F. Hernández-Campos, F.D. Smith, K. Jeffay, Generating Realistic TCP Workloads, Proc. Computer Measurement Group Intl. Conf., Las Vegas, NV, December 2004.

[11] A. Kuzmanovic and E. Knightly, *Low-Rate TCP-Targeted Denial of Service Attacks and Counter Strategies,* IEEE/ACM Transactions on Networking, vol. 14(5)

[12] Mina Guirguis, Azer Bestavros, Ibrahim Matta, and Yuting Zhang, *Reduction of Quality (RoQ) Attacks on Internet End-Systems*, Proc. , IEEE INFOCOM 2005, March 2005

[13] K. Jeffay J. Aikat and D. Smith. *Differential congestion notification: Taming the elephants*. Proc., IEEE ICNP 2004, October 2004.

[14] Guang Yang, Gerla, M. and Sanadidi, M.Y. *Defense against low-rate TCP-targeted denial-of-service attacks,* Proc., ISCC 2004, July 2004, pp. 345-350

[15] Yu Chen, Kai Hwang, and Yu-Kwong Kwok, "Filtering of Shrew DDoS Attacks in Frequency Domain," *the First IEEE LCN Workshop on Network Security (WoNS)*, 2005.

[16] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, *On the Characteristics and Origins of Internet Flow Rates*, Proc., ACM SIGCOMM 2002, Aug. 2002, pp. 161-174.

[17] F. Hernández-Campos, A.B. Nobel, F.D. Smith, K. Jeffay, Understanding Patterns of TCP Connection Usage with Statistical Clustering, Proc. IEEE MASCOTS 2005, Atlanta, GA, September 2005.