Computer and Network Performance: Graduating the “Age of Innocence”

Udi Ben-Porat, *Student Member, IEEE,* Anat Bremler-Barr, *Member, IEEE,* and Hanoch Levy, *Member, IEEE.*

***Abstract*—Performance analysis has been used for over half a century for the design and operations of computer systems and communications networks. This methodology, mostly stochastic in nature, has been based on the underlying paradigm that users are innocent and ”well behaving”. Following the flourish of the internet and the subsequent rise of malicious incidents, research started investigating the effect of malicious attacks aimed at degrading system performance.**

The objective of this work is to understand how system performance is affected by malicious behavior and how performance evaluation should account for it. We do so by considering an array of ”classical” systems taken from the literature and examining their degree of vulnerability. These can also serve in providing some initial insights into what makes a system design vulnerable or resilient.

I. Introduction

Half a century ago L. Kleinrock published [1] where he presented a methodology for evaluating packet based networks performance using the theory of queueing networks. The succeeding 50 years have seen a flourish of research in the areas of computer and network performance. These have been triggered by the fast advances in computer design and computer networks, and was propelled by the continuous needs of the industry for efficient design of their systems and networks.

During most of this period performance analysis of these systems, as well as their design, has been based, to a large extent, on stochastic analysis. The underlying assumption behind this methodology is that customers (alternatively requests, messages sent, etc.) behave as a stochastic process which drives the overall system performance. Such analysis commonly leads to stochastic-type results such as expected values (average/mean), variances and distribution tails. For example, in [1] the interest is in the expected delay experienced by packets traveling across a Wide Area Network. Also, Kleinrock’s textbook [2], which is one of the first textbooks dealing with computer performance and computer networks analysis, is based on that approach; Similar representative examples are Bertsekas and Gallager’s computer networks textbook [3], in its network analysis parts, and Harchol-Balter’s Computer performance textbook [4].

This methodology, shared by many researchers as well as system designers, has been inherently based on the underlying paradigm that all users are innocent, aiming merely at improving their own performance. Perhaps equivalently, the whole incubation, design and birth of the Internet has implicitly shared this “user innocence” underlying assumption.

The Internet, however, not only grew much faster than anyone could imagine[1], it also matured as fast, and recent years have shown that the Internet’s “age of innocence” is over. In the recent period the welfare of the Internet has been threatened by malicious Distributed Denial of Service (DDoS) attacks as well as Cyber attacks. DDoS attackers consume the resources of the victim, a server or a network, causing a performance degradation or even the total failure of the victim. The ease with which such attacks are generated makes them very popular ([5], [6]). In performance evaluation terms, this has broken the underlying paradigm by which all users are ”innocent”.

It took quite some time for the performance community to react to these changes. During that time period computer attacks have been solely under the research domain of ”computer and network security”; an area that is quite disconnected and remote from the area of performance evaluation. The focus of that community has been on attack identification and mitigation, while disregarding the quantitative evaluation of their impact on performance.

The paradigm shift has finally lead to the introduction of a new research direction based on combining knowledge from these two fields: Security and Performance. This research venue has been aiming at viewing the attacks through ”performance eyes”, that is, evaluating system performance in the presence of malicious users. In other words, studying ”malice-based performance”. The findings of these studies have been that the effect of malicious behavior on system performance and design can be, in some cases, quite drastic.

The purpose of this work is to explore the traditional computer and network performance paradigms, and examine them under this new light, namely in the presence of “malicious” behavior. Our objective is to present this new malice-based performance analysis, to contrast it with the traditional analysis, and to demonstrate its impact through reviewing a collection of representative cases (most presented in recent years literature) from the areas of computer and network performance. The approaches presented in this work can assist system designers in evaluating their design subject to malicious behavior as well as in understanding how a system should be designed to be resilient to such attacks. In theoretical

U. Ben-Porat is with the Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland. e-mail: ehud.benporat@tik.ee.ethz.ch.

Anat Bremler-Barr is with the Computer Science Dept., Interdisciplinary Center, Herzliya, Israel. email: bremler@idc.ac.il.

Hanoch Levy is with the Computer Science Dept., Tel-Aviv University, Tel-Aviv, Israel. email: hanoch@post.tau.ac.il.

[1]Had anyone imagined this tremendous growth, we would not be struggling today with the 32 bit limited address space of the Internet.

terms, we would like to give an insight into what makes a design more or less vulnerable.

We start this work with briefly enlisting a sample of performance attacks taken from the literature and classifying them according to the vulnerability point they address (Section 2). We then (Section 3) describe several methodologies for quantitatively evaluating the impact of the attacks on performance. We then (Section 4) move into an in-detail presentation, portraying a representative sample of test cases. For these cases we review their performance under normal conditions, describe the malicious behavior, examine the quantitative effect of the the malicious behavior on the system, and propose how to modify the design to diminish the malice impact on performance. In Section 5 we briefly discuss possible solutions. Finally, concluding remarks are provided in Section 6.

II. How your Performance can be Attacked: Attack Survey and Classification

In this section we review the literature and enlist a representative set of performance attacks.

Roughly speaking, we can classify the methods of launching sophisticated DDoS attacks into four classes: Algorithmic Worst-Case Exploit, Traffic Pattern Exploit, System Architecture Worst-Case Exploit and Protocol Deviation Exploit. We explain and demonstrate below each of the categories.

1) **Algorithmic Worst-Case Exploit or Complexity Attack** - Attacker exploits the worst-case performance of the system's algorithm which differs from the average-case that the algorithm was designed for. Crosby and Wallach [7] were the first to demonstrate the complexity attack on the commonly used Open Hash data structure. They showed that an attacker can design an attack that achieves worst-case complexity of $O(n)$ elementary operations per insert operation instead of the average-case complexity of $O(1)$. Examples of algorithms that are vulnerable to complexity attacks are Closed Hash [8], Quicksort [9], regular expression matcher [10], Intrusion Detection Systems [11], [12] and Linux route-table cache [13].

2) **Traffic Pattern Exploit** - Attacker exploits the (stochastic) worst-case traffic pattern that can be applied to the system. This category is similar to the first one with the distinction that the worst-case scenario involves a specific traffic pattern of requests from multiple users. This type of attack is demonstrated in the Reduction of Quality (RoQ) attacks papers [14], [15], [12]. RoQ attacks target the adaptation mechanisms by hindering the adaptive component from converging to steady-state. This is done by sending - from time to time - a very short burst of surge demand imitating many users and thus pushing the system into an overload condition. Using a similar technique, Kuzmanovic and Knightly [16] presented the Shrew Attack which is tailored and designed to exploit TCP's deterministic retransmission timeout mechanism. Another example of an attack exploiting the stochastic worst-case is given in [17], [18]. There it is shown that Weighted Fair Queueing (WFQ), a commonly deployed mechanism to protect traffic from DDoS attacks, is ineffective in an environment consisting of bursting applications such as the Web client application. The paper [19] shows an attack on the SSL handshake, conducted by repeatedly making requests that require costly decryption operations on the server side.

3) **System Architecture Worst-Case Exploit** - While the previous categories exploit a weak point in the algorithm, attacks belong to this category exploit a weak point in the specific architecture of the system the algorithm runs on, mainly the interaction of the hardware with the software implementation of the algorithm. For example: Moscibroda and Multu [20] discuss a sophisticated DDoS attack which degrades the performance of a Multi-Core System. The attack is conducted by one process (running on one core) targeting the weakest point in such a system - the DRAM shared memory - and degrading the performance of other processes in the system. [21] shows the vulnerability of the pattern matching algorithm of NIDS to traffic which causes high cache miss rate and thus decreases the performance to 14% of the original rate.

4) **Protocol Deviation Exploit** - Attacker deviates from the protocol rules, exploiting the fact that the protocol design is based on the assumption that all the users obey the rules of the protocol. Most traditional DDoS attacks [22], like SYN attacks and DNS attacks, belong to this category. These attacks exploit the fact that performing only part of the flow protocol but not finishing it correctly harms the performance of the protocol significantly. For example, in the SYN attack the attacker sends many SYN requests but does not send the FIN. Another weak point, can be found also in the protocol specification. The papers [23], [24] identify a number of vulnerabilities in the authentication phase in 802.11 standard that can be used by an attacker to de-authenticate other legitimate users.

Another type of protocol deviation attack consists of obeying the protocol flows while "cheating" in the contents of the protocol messages. Recent works in the context of wireless scheduling [25], [26], [27], [28], show that "cheating" by misreporting the channel capacity can harm the system performance significantly.

III. Modeling Methodologies for Malice-Accounting Performance Evaluation

In this section we deal with how one can quantify the damage created by an attack. We start with discussing what is malice-accounting performance. We then describe a performance accounting methodology (Section III-B) and describe in detail a concrete performance accounting metrics (Section III-C). We finalize with describing related evaluation methodologies not based on performance accounting (Section III-D).

A. *Malice-Accounting Performance*

How does "Mal-case" performance differ from worst-case performance and average-case performance? An underlying assumption behind both worst-case and average-case analysis is that in reality there are many cases and realizations; the worst-case performance will evaluate the worst of these realizations while the average-case will average all of them. The mal-case performance can be thought as some type of intermediate paradigm. It assumes that the majority of the population behaves ”normally” (”innocently”) and thus its own performance can be measured via the normal average-case performance. However, this performance is affected by having a fraction of the population (malicious users) behaving as to drive the system into a worst-case realization.

How mal-case performance is defined? Isn’t a malicious user like an adversary? To make it a practical measure, one cannot attribute the malicious user(s) with ”super-powers” and therefore cannot relate to it as an adversary (as commonly done in worst-case analysis). Rather – one has to bind the malicious user to practical limitations. Such limitations can be of several categories: 1) *Knowledge of System Operations* – The attacker may or may not know some of the system internals. For example it may know that the system uses a hash table, but possibly not know the exact hash function. 2) *Knowledge of system state and other users state* – The internal temporal system state (e.g. which buckets in the hash table are occupied at this moment) is commonly hidden from the malicious user. Similarly, knowledge about the temporal behavior of the other users (e.g. which buckets will they address in future requests) is hidden as well. 3) *Resource Limitations* – The attacker has limited ”budget” expressed in limits on the size of the attack he/she can launch.

B. Malice-Accounting Performance: Sophisticated Attacks

From performance point of view, our mere interest is in the so-called *sophisticated attacks*. Under such attacks the attacker sends traffic aiming to hurt a weak point in the system design in order to significantly degrade its performance. The sophistication lies in the fact that the attack is tailored to the system’s design, aiming to increase the attack effectiveness.

The attacker’s incentive is to increase the attack’s performance effect while minimizing the amount of traffic it sends. The use of sophisticated attacks reduces the cost of attacks, i.e., reduces the number of zombie computers the attacker has to operate and and the complexity of attack coordination. Moreover, the use of sophisticated attacks increases the likelihood that the attack will succeed in going unnoticed (going under the radar) by DDoS mitigation mechanisms, which are usually based on detecting higher-than-normal traffic volume.

C. A Performance Vulnerability Metric

How the vulnerability of a system to attacks (malicious behavior) can be evaluated? It is desired that such evaluation will account for the performance effect (damage) caused by the attack as well as for the efforts required in launching the attack. A measure of *vulnerability factor* has been proposed and defined [8] as the maximal performance degradation (damage) that malicious users can inflict on the system using a specific amount of resources (cost) normalized by the performance degradation attributed to regular users using the same amount of resources. In a sense, this evaluates how many innocent users are prevented accommodation in the system due to the activity of one malicious user. In other words, this is the number of innocent users the malicious user ”worth”.

Formally, the Vulnerability Factor can be defined as follows: Let the $usersType$ parameter be equal to either regular users (R) or malicious attackers (M_{st}) with strategy st. Note that we use the plural terms since some of the attack types occur only in specific scenarios of multiple coordinated access of users to the system [14]. Let $budget$ be the amount of resources that the users of $usersType$ spend on producing the additional traffic to the system, i.e., the cost of producing the attack is limited by the $budget$ parameter. The $budget$ can be measured differently in the various models, e.g. as the required bandwidth, or the number of required computers, or as the required amount of CPU and so on. Let $\Delta Perf(usersType, budget)$ be the performance degradation caused by the additional traffic generated by the additional users (of type $usersType$ with a $budget$ resource limit). The performance can be measured by different means such as the CPU consumption, the delay experienced by a regular user, the number of users the system can handle and so on.

We define the *Effectiveness* of a malicious strategy st on the system as

$$E_{st}(budget = b) = \frac{\Delta Perf(M_{st}, b)}{\Delta Perf(R, b)}, \quad (1)$$

and the *Vulnerability Factor* V of the system as:

$$V(budget = b) = max_{st}\{E_{st}(b)\}. \quad (2)$$

In evaluating attack effectiveness one may distinguish between two types of effects: 1) *In-attack* performance, and 2) *Post-attack* performance. In the former one measures the attack’s performance effects while the attack is carried on. In the latter one evaluates the effects on system performance once the attack has completed.

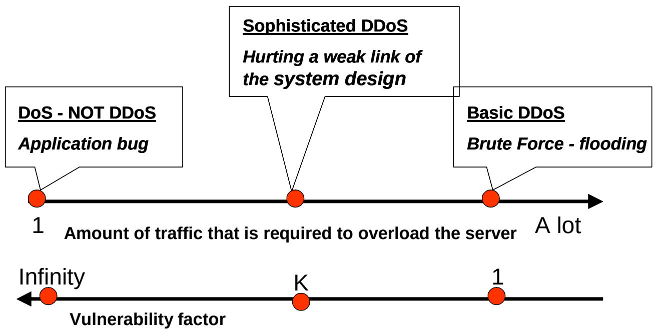


Fig. 1: Attack Effectiveness and Corresponding Vulnerability.

Figure 1 qualitatively demonstrates attack effectiveness in relation to attack sophistication. Brute force attacks (no sophistication) are depicted on the right where a huge amount of traffic is needed to overload the system since the load placed by an attacker operation is identical to that of an innocent user operation (vulnerability is 1). On the left we depict attacks that completely shut down a system and thus the corresponding vulnerability is infinity. The range in between the extreme points belongs to sophisticated attacks (vulnerability is K) where an attacker operation is performance-wise K times more effective than that of an innocent user.

We note that the paper [14] concentrates on related performance measurement, the attack potency. The attack potency, is defined to be the ratio between the damage caused by an attack and the cost of mounting such an attack. Specifically, the paper defined $potency = \frac{damage}{cost^{\frac{1}{\Omega}}}$. The goal of the attacker is to maximize the potency of the attack. Note that the definition for $\Omega = 1$ is basically the $\Delta Perf(M_{st}, budget)$ in the vulnerability factor, without normalizing it to the performance degradation caused by a regular user. In this definition the Ω is introduced to model the aggressiveness of the attacker. A large Ω reflects the highest level of aggression, i.e., an attacker bent on inflicting the most damage and for whom cost is not a concern. Mounting a regular DDoS attack is an example of such behaviour. A small Ω reflects an attacker whose goal is to maximize damage with minimal exposure.

D. Alternative Methodologies: Economics Based and Traffic Based Evaluation Methodologies

Evaluating the financial impact of a DDoS attack is mainly meant to help service providing companies to evaluate their risk and assess the profitability of purchasing and maintaining security measures, backup systems and insurance.

In [29] the authors present a model and methodology to calculate the expected financial loss for a variety of scenarios involving Internet DDoS attack. The financial damage is divided to four categories: *Downtime Loss*, *Disaster Recovery*, *Liability* and *Customer Loss*. The evaluation requires to predict various parameteres such as the down time of the system, cost of the recovery team, productivity degradation during outage, number of actual and potential customers lost and so on.

In [30] the authors aimed at creating DDoS impact-measure factor called MIDAS factor, equals (Cost of DDoS attack) /(Total provider yearly revenue) where *Cost of a DDoS attack = Cost of SLA violations + Cost/Risk of customers leaving*. This metric captures the relative economical impact of a DDoS attack which can be compared between networks of different sizes. This measure is made simple enough to be evaluated by real-time network measurements.

Another metric which uses traffic trace is *DoS impact* propsed by Mirkovic et al. in [31]. The traffic is associated with different user tasks called transactions. These transactions are then classified to different categories, each with its own unique QoS requirements. The metric is defined as the percentage of transactions in each category which failed to achieve their QoS goals.

IV. Computer and Network Vulnerability - Test Cases

A. Computer Performance and Scheduling: Theory

How can the performance of a computer system be attacked? Computer performance in general, and scheduling specifically, have been studied extensively as early as the 60's of the prior century; A common modeling paradigm is a system consisting of an arrival stream of requests, where the processing requirement made by a request is a random variable reflecting the processing times of typical users. For example, for packet networks it is commons to assume that the processing time is proportional to the packets length. The overall system performance then depends on this distribution and on the processing policy.

For example, using the First Come First Served policy, the expected delay experienced by an arbitrary request (assuming an M/G/1 model) is given by the well known formula:

$$E[W] = \frac{\lambda b^{(2)}}{2(1-\rho)}, \tag{3}$$

where λ is the (Poisson) arrival rate, $\rho = \lambda b$ is the system utilization and b, $b^{(2)}$ are the first two moments of the processing time distribution. Under this policy, long jobs and short jobs experience the same delay.

In contrast, the processor sharing (PS) policy, whose first analysis was provided in [32], favors short jobs and (partially) protects them from long jobs; Under PS, the expected delay of a job of size x depends linearly on the job length :

$$E[W(x)] = x/(1-\rho), \tag{4}$$

where the partial effect of long jobs on short jobs is through the denominator.

Note that Generalized Processor Sharing (GPS), which provides sharing between *classes* of customers, and the corresponding well known practical implementation Weighted Fair Queueing (WFQ), are known to *fully protect* short jobs from long jobs (or low load flows from high load flows).

How can malicious users attack the performance of a computer system? Equations (3), (4) suggest that this can be done by either increasing the arrival rate (λ) or the moments of the service times $b, b^{(2)}$. Increasing the arrival rate is done by sending a large number of requests; this forms an "unsophisticated" attack which is easy to detect and requires great effort from the attackers, and is out of scope for this study.

The alternative approach is to increase the processing time experienced by either the attacker's jobs or by innocent users' jobs. The resulting theoretical question is how the performance equations (3), (4) can be affected by controlling the processing requirements of individual jobs or of a group of jobs.

In the next section we deal with this approach, focusing on attacks performed on Hash Tables, which are common and critical data structures used in computer systems as well as networking components (e.g. routers). Our discussion below starts with the traditional (linear) hash tables and then moves to more advanced tables (Peacock, Cuckoo) which became popular in networking technology in recent years.

What makes a system design vulnerable to performance attacks? Our exploration of the various hash systems aims at shedding some light on this design problem.

B. Computer Performance and Scheduling - Attacks on Hash Tables

1) **Vulnerability of "Simple" Hash Tables***:* Many network applications use the common Hash data structure. A Hash table is based on an array of buckets of size M meant to store keys from a space larger than M. A hash function $h(s)$ is used to identify the bucket in which the key s should reside. In [8] the authors analyzed attacks on Open Hash (a.k.a Closed Addressing) and Closed Hash (a.k.a Open Addressing). In the Hash data structure, the Insert operation of a new element is the most time consuming operation. We summarize here the attack on Closed Hash tables with linear probing as an example for an attack that causes both in-attack and post-attack damage.

A Closed Hash consists of an array of size N for storing the keys of the elements. During a key insertion, if the key is hashed into an occupied bucket, then the neighboring bucket is probed, and this process repeats until an empty bucket is found [33] (this is called linear probing; other probings are not analyzed here). The average complexity of performing an insert operation in under regular operation of the Hash is $O(1)$ memory accesses. K insertions can trigger worst-case behavior by having all K keys hashed into the same bucket.

An attacker with the ability to compute the hash function, can produce this behavior and cause *in-attack damage* by requiring $O(K)$ memory accesses per insertion and increase the load on the system [8]. This is compared to $O(1)$ memory accesses per insertion by regular users, as the traditional complexity analysis predicts. Therefore, the in-attack vulnerability is $O(K)$.

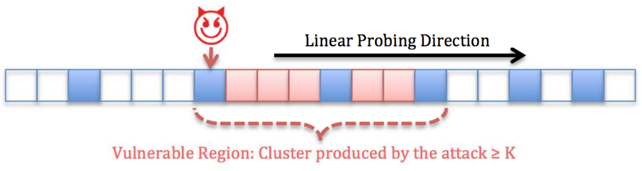


Fig. 2: Closed Hash: Attack creates a vulnerable region.

In addition, after the attack has ended, a cluster (see Figure 2) of at least K consecutive occupied buckets is formed (by the buckets holding the key used in the attack). The existence of such a long cluster increases the insertion complexity of keys searched-for after the attack; therefore, this attack causes also *post-attack damage*. The analysis in [8] shows that, the vulnerability metric of the post-attack insertion-complexity increases linearly with the budget of the attacker and can reach values around $V \approx 90$. That is, the insertions by a malicious user can make the system 90 times slower than after the same amount of insertions by a regular user.

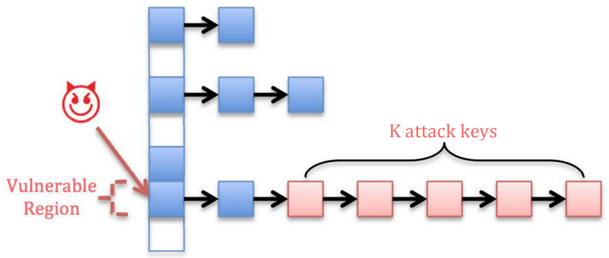


Fig. 3: Open Hash: Vulnerable region is not highly accessible.

Remark 1 (Malice-Based System Design): The importance of malice-based performance analysis for system design can be appreciated by noting that while Closed Hash is extremely vulnerable to post-attack damage, Open Hash is not vulnerable to such damage at all (see [8]) and thus is highly preferred. This major difference is in contrast to the view taken by traditional performance analysis which finds the two hash strategies to be equivalent.

The malice-based performance difference results from the inherent structure of these hash tables: An attacker inserting all keys into the same bucket produces a long chain under Open Hash and a cluster under Closed Hash. Under Open Hash this increases the complexity only of the hash operations targeting a single bucket, namely forming a ”vulnerable region” of one bucket only (see Figure 3). In contrast under Closed Hash it increases the complexity of the operations accessing all of the adjacent buckets, namely forming a ”vulnerable region” whose size is $O(K)$ (see Figure 2).

The average operation complexity in Open Hash depends only on the average chain-length, which depends only on the number of keys stored in the table rather than their distribution in the table. Therefore, regardless of its insertion-strategy, an attacker cannot cause greater damage than a regular user which inserts the same amount of keys.

It is important to note that while the expected post-attack chain-length practically does not change under Open Hash, the *second moment* of the chain length *does change significantly* (since the attacker increased the length of one chain). Thus, if the jobs that access an Open Hash conduct this access by first queueing in an M/G/1 queue, their expected delay *will be significantly affected* due to the second moment of the Hash processing time (see numerator of Equation 3).

*2) **Multiple Choice Hash Tables (MHT):*** Multiple-choice Hash Tables (MHT), and in particular Peacock Hash [34] and Cuckoo Hash [35], are easy to implement and are currently the most efficient and studied implementations for hardware network systems such as routers for IP lookup ([36], [37] and [38]), network monitoring and measurement [39] and Network Intrusion Detection/Prevention Systems (NIDS/NIPS) [40]. For more information about hardware-tailored hash tables we recommend the recent survey by Kirsch et al. [41]. In the following text we bring a summary of the results in [42] where the authors described malicious attacks on Peackok and Cuckoo Hash and evaluated their Vulnerability.

Peacock Hash Tables:

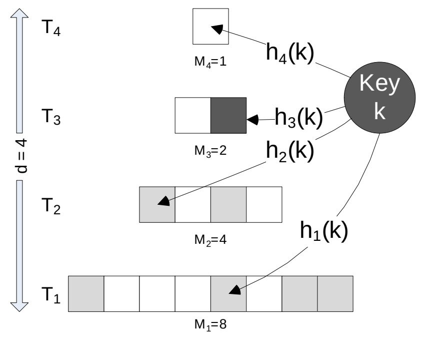
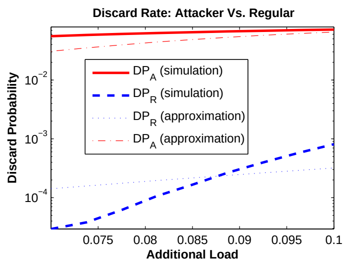
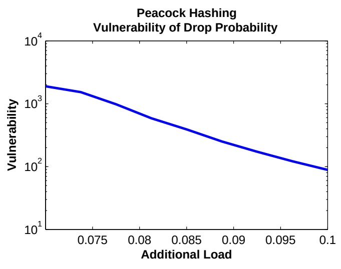


Fig. 4: An insertion of key k into a small Peacock hash table with four tables. Gray buckets are occupied with existing keys and white buckets are free. The black bucket marks the bucket where k is finally placed.

A Peacock hash table consists of one large main table T_1 (which typically holds 90% of the buckets) and a series of additional sub-tables where collisions caused during insertions are resolved. Its structure is based on the observation that only



(a) Attack Damage



(b) Vulnerability

Fig. 5: Peacock Hash: Plot (a) – The discard probability after insertions by regular (DP_R) and malicious (DP_A) users as a function of the load they add (K/M). Plot (b) – Vulnerability simulation results (V) as a function of the additional load for a table with four subtables, each 10 times smaller than the previous one. Each sub-table has a load of 10% before the insertion of the additional keys.

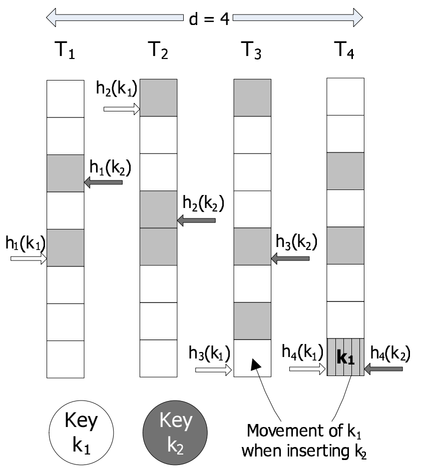
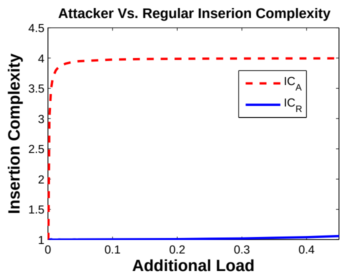
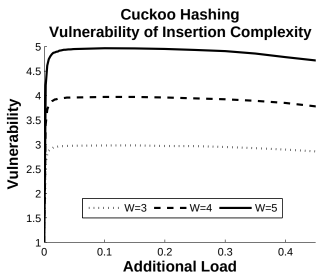


Fig. 6: A move of key k_1 during the insertion of k_2 into a Cuckoo hash table with 4 sub-tables. The white (right) and the dark (left) arrows mark the hash values of k_1 and k_2 in the different sub-tables, respectively.



(a) Attack Damage



(b) Vulnerability

Fig. 7: Cuckoo Hash: Plot (a) – The average insertion complexity of a key insertion by an attacker IC_A and by a regular user IC_R in a system with $W = 4$. Plot (b) – The vulnerability ($V = IC_A/IC_R$) as a function of the additional load (K/M). In both plots the existing load is $\alpha = 0.1$.

a small fraction of the keys inserted into a hash table collide with existing keys (that is, hashed into an occupied bucket) and even a smaller fraction will collide again, etc. These backup tables are usually small enough for their summary (implemented by bloom filters) to be saved on fast on-chip memory which dramatically improves the overall operation performance in Peacock Hash. Starting from T_1, the insertion algorithm probes the sub-tables one after the other until finding a table where the bucket to which the key hashes into is free (see Fig. 4).

An attacker capable of predicting hash values to some degree, can insert small number of keys colliding with each other in the first table(s), flooding the smallest backup tables. Since the backup tables are stored on a fast memory, the in-attack damage in negligible. However, it causes a substantial post-attack damage, since when the backup tables are flooded, the probability for a key to be dropped is much higher than in the normal state (in which all sub-tables have similar load). In [42] the authors evaluated the vulnerability of the drop probability in Peacock hash tables. Figure 5 depicts a sample of the vulnerability results, demonstrating very high vulnerability, ranging between 10^2 to 10^3.

Cuckoo Hash Tables:

A Cuckoo Hash table (see Figure 6) is made of two (or more) sub-tables of the same size . A key is assigned with one hash value in each of the sub-tables and can be placed in any of the corresponding buckets. When a key k finds all its buckets occupied, one of the keys k' residing in those buckets is then kicked out. Then, in the same manner, the algorithm searches a bucket for k'. Note that the number of moves allowed during one insertion is limited to W moves. It is done to avoid (rare) cases in which the insertion is in a loop state. Cuckoo Hash allows achieving a higher table utilization than that achieved by alternative MHT schemes that do not allow moves, while maintaining $O(1)$ amortized complexity of an Insert operation (although the complexity of a single insertion is not bounded by a constant).

The basic idea behind the attack is to insert K keys that all hash into the same small set of buckets B, such that $K > |B|$. Except for the first $|B|$ keys, every attack key causes an insertion loop in which every move triggers another. Note that [42] states that unlike Peacock Hash, Cuckoo Hash is immune to post attack damage, since there is no specific layout of elements in the table that Cuckoo is vulnerable to more than another.

Figure 7 depicts simulation results of such an attack. It shows that the average insertion-complexity of a malicious key is close to W while that of a regular key is very close to 1. This is an important result since the maximal load of a Cuckoo Hash table can be extended either by increasing the number of sub-tables d or by increasing W (both options lead to lower key-dropping probability on the expense of performance). However, as shown in [42], increasing W, which means increasing the system vulnerability, is less safe than increasing d.

3) ***Discussion and Comparison****:* What makes one hash table more vulnerable than the other? While we do not have an analytic quantifiable answer, we try to provide an intuitive qualitative reasoning. Structures which are characterized by ”uniform design”, such as Open Hash and Cuckoo Hash, seem to be significantly less vulnerable than those with ”non-uniform design”, such as Peacock Hash. In the latter, the non-uniformity creates vulnerable regions (such as the top part of the Peacock tables) at which the attack can concentrate its efforts, causing much damage in little effort. In the former, such ”special” regions do not exist, implying that the attack cannot create more damage than innocent users. Note that in the Closed Hash, whose design looks ”uniform” under regular operation, the attacker creates a ”non uniform” structure (a region of $\geq K$ consecutive occupied cells, see Fig. 2) thus making the hash vulnerable at post-attack time.

C. Flow Control in Networks

Flow control is one of the basic and most influential mechanism in computer networks. Using this mechanism the network controls the temporal rate by which each flow can send its data. TCP, the common transport layer in the Internet, defines a smart flow control mechanism, that aims to prevent the endpoints from sending traffic which overwhelms the network, or the other endpoint.

Kuzmanovic and Knightly [16] presented the Shrew Attack which is tailored and designed to exploit TCP’s deterministic retransmission timeout (RTO) mechanism, under which a flow has to wait RTO seconds before retransmitting a packet that was not acknowledged. By using well timed attack-bursts, an attacker can perform an effective low-rate attack on TCP flows. The attack begins with a burst of traffic which causes packet losses to the TCP flows in network, causing them to enter a timeout. This will cause TCP flows to reset the congestion window size and wait RTO seconds before retransmitting. The first retransmission marks the beginning of the *slow start* phase in which TCP flows gradually increase their congestion window in order to recover the same throughput they achieved before the attack. Then, just when the flows are about to saturate the link, the attacker strikes again with another attack burst, long enough to cause the flows to return to RTO state once again. This is a low-rate attack since the bursts are well spaced in time (scale of seconds, based on *minRTO*, the minimal RTO used by the flows) and because they are conducted when the link is already close to saturation. In their Internet experiments, the authors show that a remote attacker with an average rate of only 909 kb/sec can reduce a victim’s throughput from 9.8Mb/sec to 1.2Mb.

In their work, Kuzmanovic and Knightly [16] explored different mechanisms to counter malicious flows conducting Shrew attacks in both end-points and routers. For router-assisted mechanisms, the authors reviewed several per-flow treatment mechanisms and concluded it is a hard task and they did not find an adequate solution. One of the weak points abused by the attacker is the fact that the value of *minRTO* is similar among the flows and hence they all recovering from the RTO at a relatively similar and short time interval. By using a randomized value for *minRTO*, the length of this interval increases which reduces the effectiveness of the attack. However, the authors show that increasing the range of the *minRTO* values means decreasing the TCP throughput. Therefore, the authors conclude that while there are measures to mitigate the attack, there is an inherent trade-off between the TCP performance and the vulnerability to Shrew attacks remains.

Another related avenue of attacking the TCP is the RoQ attack [14]. RoQ attacks present a general framework for targeting adaptation mechanisms employed in current computing systems and networks. The attacks keep an adaptive mechanism oscillating between over-load and under-load conditions. While Shrew attacks exploit the timeout mechanism of TCP resulting the RoQ attacks do not target this specific protocol setting, but they are a general class of dynamic exploits that target adaptation mechanisms wherever they are present: transport layer [14], admission controls [15], Moblie ad hoc network [43] and so on. Specifically to TCP, the paper [14] focuses on attack techniques that would hinder an Active Queue Management (AQM), such as RED and drop-tail, from stabilizing its queue. Thus resulting in a noisy feedback signal to the end-system transmission controllers, which in turn lead to high jitters due to oscillations, as well as inefficiencies which result in lower TCP throughput. Moreover it was shown [44] that RoQ can be used not only for attacks, but also for flows to acquire an unfair share of bandwidth.

D. Multiple Access Protocols

Multiple Access Protocols [45] deal with a shared channel, where a set of nodes sends and receives packets over the same channel. All the nodes can distinguish between an idle and a busy link, and have a “collision detect” mechanism implying that a node listens as it transmits, and can therefore detect whether a packet it transmits collides with a frame transmitted by another node. The multiple access protocols deal with controlling the access to the channel, namely with the ”decision” of which node uses the channel when. Applications include satellite networks, packet radio networks and wired local area networks.

Research started with the early papers of Abramson [46] who introduced Pure Aloha and offered the well known throughput formula (as a function of the offered load) $S = Ge^{-2G}$, and Roberts [47] who introduced Slotted Aloha; Under both schemes users use *very simple* access and collision resolution schemes. Later research moved to more sophisticated schemes accounting for carrier sensing and other access improvements (e.g. CSMA/CD by Kleinrock and Tobagi [48] and other studies). Sophisticated techniques for collision resolution have started with the work of Capetanakis [49] who offered a tree based collision resolution algorithm, leading to significant throughput improvements.

1) ***Vulnerability of Simplistic Collision Resolution - Aloha****:* In Slotted Aloha time is divided into fixed length time-slots, each suitable for one packet transmission. If a collision between two or more users occurs at slot k, all the users whose packets collide, independently schedule their retransmission of the packet to a random time in the future.

In a multiple access network a sophisticated attacker will try to disturb the network by sending traffic aiming at creating new collisions. However, due to the simplicity of Aloha, there is no information or knowledge of the system that the attacker can exploit for her benefit. Therefore, a sophisticated attacker has the same effect as a regular user and the vulnerability of Aloha can be shown to be 1.

Note that under Slotted Aloha the maximal achievable throughput is 0.368, namely 63.2% of the channel bandwidth is wasted due to packet collision losses.

*2) **Vulnerability of Sophisticated Binary-Tree Based Resolution**:* In sophisticated collision resolution protocols users base their future transmissions on past collisions they observed. Therefore, collisions are resolved more efficiently and throughput can reach values of up to 0.5. Under such protocols the sophisticated attacker can observe the history of collisions and predict, with high probability, other users behavior, namely their retransmission attempts. It can use this knowledge to create more collisions than a regular user. We show here that by using a very simple strategy the malicious attackers can conduct effective sophisticated attack (with *effectiveness* ranging between 2.5 to 5).

In the following text we show an attack which targets the coin-flipping mechanism of the Binary Tree Protocol. This mechanism is also the basis for many of the enhanced protocols[2], which makes them vulnerable to such an attack as well. For ease of presentation we concentrate on the Binary Tree Protocol.

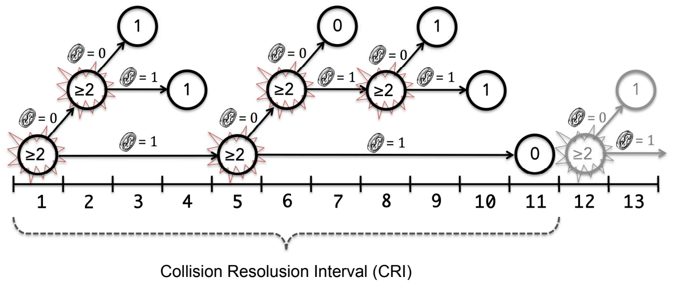


Fig. 8: Collisions resolution under Binary Tree Protocol. The circled number denotes the number of transmissions in each time slot. The arrows point when should users retransmit based on their coin-flipping.

In the Binary Tree Protocol [49] [45] the underlying model and the assumptions used are identical to those assumed for Slotted Aloha. Let S_k denote the k-th time slot. If a collision between two or more users occurs in S_k, all the users not involved in the collision wait until the collision is resolved. The users involved split randomly into two subsets, by each flipping a coin. The users that flipped 0 retransmit in S_{k+1} while those that flipped 1 wait until all those in the first subset transmit successfully their packets. If S_{k+1} is either idle or contains a successful transmission, the users of the second subset retransmit at S_{k+2}. If S_{k+1} contains another collision, then the procedure is repeated, i.e., the users whose packets collided in slot S_{k+1} flip a coin again and operate according to the outcome of the coin flipping, and so on. In Figure 8, a collision of four packets in S_1 is resolved in time slots S_1 to S_{11}.

[2]The basic Binary Tree Protocol has a throughput of 0.34, and the optimized binary tree protocol has throughput of 0.43 [49]. The Modified Binary Tree protocol and the Clipped Modified Binary Tree Protocol have throughput values of 0.381 and 0.487 respectively.

A malicious user in this model will try to collide with an otherwise successful packet transmission of an innocent user. In the strategy we consider, to be denoted the $LEAF$ strategy, the attacker transmit in S_{k+1} only in two cases:

- *Pattern A:* S_{k-1} contains a collision of existing users (the malicious users were not involved) and S_k contains a successful packet transmission.
- *Pattern B:* S_{k-1} contains a collision where the malicious users were involved and S_k is idle.

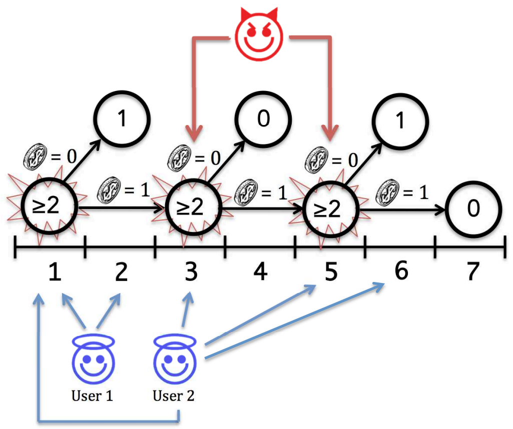


Fig. 9: An attack on coin-flipping based collision resolution. The attacker transmits in time slots with high probability for an otherwise successful transmission (by a regular user).

Figure 9 depicts an attack on the collision resolution of two packets. The attacker disrupts otherwise successful transmissions of User 2 in time slots 3 and 5 after identifying Pattern A in time S_1 and S_2 and identifying Pattern B in time slots S_3 and S_4. Observe that due to the attack, User 2 transmits its packet 4 times instead of 2 and the collision resolution takes 7 time slots instead of 3.

The basic idea behind the $LEAF$ strategy is that a large fraction of the collisions consist of exactly two packets. Patterns A and B provide good indication that there was indeed a collision of two packets, and that in the next slot a packet will be transmitted successfully if the attacker would not disturb.

The $LEAF$ strategy is not proved to be the most effective attack strategy. Therefore, its *Effectiveness* provides a lower bound for the *Vulnerability* value of this model. Our preliminary analytic results and simulations (omitted from this text) show that the probability for a malicious transmission to collide with an otherwise successful transmission (by another user) in larger than 82%. At rates below 0.2 packets/time-slot this probability increases to more than 90%.

We will use the *number of transmissions per packet*, indicating the efforts to be spent by regular users to reach successful transmission, as the performance metric by which we measure the vulnerability of the system. Note that the performance

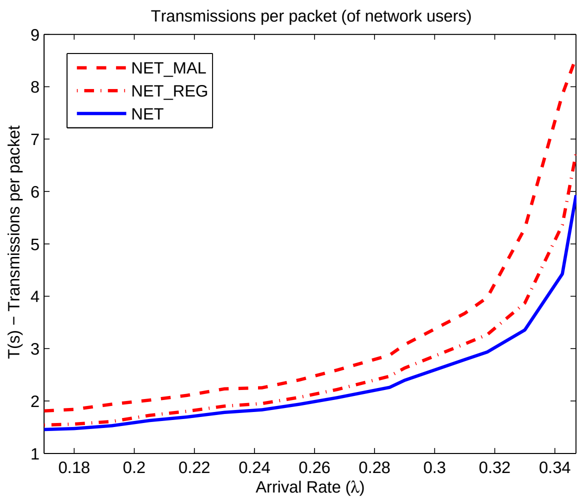


Fig. 10: Average transmissions per packet in the presence of additional malicious users (*NET_MAL*), additional regular users (*NET_REG*) and before the arrival of additional users (*NET*).

degradation in throughput and delay is derived directly from this metric.

Let *NET* be a scenario when the system serves regular users with arrival rate λ. Let *NET_REG* and *NET_MAL* be scenarios in which we add to *NET* transmissions at rate γ of regular and malicious user respectively. Figure 10 depicts the average number of transmissions per packet in each of the scenarios. The effectiveness of the attack (for a certain λ) is given by the proportion between the the difference between *NET_MAL* and *NET* to the difference between *NET_REG* and *NET*. Our simulations demonstrate that the effectiveness of the $LEAF$ strategy is quite significant, ranging between 2.5 to 5.

3) ***Discussion and Comparison:*** the reader may note an inherent difference between the vulnerability of Slotted Aloha and the Binary Tree protocol. While the former is not vulnerable at all, the latter is highly vulnerable. The resilience of Aloha seems to stem from its simplicity and ”uniform structure”. Since all points of the protocol are identical to each other, the attack cannot address a specific, more vulnerable point, and thus an attacker is as harmful as an innocent user. In contrast, the Binary tree protocol is characterized by highly ”structured” process, resulting with specific points which are more vulnerable than others, allowing a sophisticated attacker to be more effective than an innocent user, making the system highly vulnerable.

E. Network Routing - Amplified Vulnerability

Routing protocols are the protocols by which routers communicate with each other, disseminating information that allows them to select paths between any two nodes on the network. Border Gateway Protocol (BGP), is the de facto standard and the only inter-domain routing protocol of the internet.

A stable routing protocol is critical to the performance of the whole network, and any period of disfunction can lead to dramatic network performance degradation. As such, a routing protocol can be used as an *amplifier* of performance attacks: Hurting the performance of the traffic carrying the routing messages, even mildly, can cause routing disfunction and drastic network performance degradation.

Such amplification can be achieved by exploiting The vulnerability of TCP to a low-rate TCP-targeted DoS attack. The Shrew attack [16], can be used to attack many protocols that use TCP as their underlying transport protocol, such as the BGP routing protocol [50]. Zhang [50] shows empirically that the default configurations of routers are vulnerable to conducting the Shrew attack on the TCP session of BGP. The attack can reset the BGP session, thus the router loses the routing information, until the attack is ended. Specifically, [50] shows that the attackers can bring down the targeted BGP session in less than 216 seconds. While the attack raises the utilization of the bottleneck link only to 42%, it’s effect is highly amplified since it can increase the BGP session reset probability to as high as 30%.

Note that the TCP session of BGP is vulnerable also to conventional attacks such as TCP RST and SYN flood attacks. However, router vendors have provided protection against these known attacks [51], [52], [53]. Note that these attacks are easier to mitigate since the destination of the attacked traffic in this case must be the router itself, and the traffic attacked must be the BGP session specifically. Therefore, a simple useful remedy is to allow the router to receive traffic only from a neighbor router [53]; this remedy works since under E-BGP information is commonly exchanged only between neighboring routers.

In contrast, Shrew attacks on BGP sessions are harder to mitigate due to the fact that the attack can be launched using *any packets* passing through the routers without the need to send packets destined to the router, and to the BGP session specifically.

The amplifying effect, whereby mild attack on arbitrary traffic of the link hurts BGP drastically, serves also as the key for mitigating the attack. A simple queueing prioritization mechanism which will protect the BGP session resources from being affected by the other traffic will do the job. This can be easily portrayed by considering an M/G/1 queueing system with two traffic classes where non-preemptive priority is given to class 2 over class 1, and where the mean delay, W_p, of class p is given, as a function of the utilization of the classes ρ_p, $p = 1, 2$, by [54] (see Equation (3.31) in [2]):

$$W_p = \frac{W_0}{(1-\sigma_p)(1-\sigma_{p+1})}, p = 1, 2; \qquad \sigma_p = \sum_{i=p}^{2} \rho_i \tag{5}$$

The equation shows that the BGP traffic whose volume/utilizaion (ρ) is low would experience negligible delay if served at high priority and significant delay if served at low priority.

Note that under the common deployments [50], routing protocols usually *receive no priority* over other traffic ([55]) – from which this amplified vulnerability results.

F. Wireless Network Scheduling

Channel-aware scheduling strategies - such as the CDF Scheduler (CS) [56] and the popular *Proportional Fairness Scheduler* (PFS) [57] - provide an effective mechanism for utilizing the channel data rate for improving throughput performance in wireless data networks by exploiting channel fluctuations. In the down-link wireless scheduling model, the users are waiting for their data to be sent from a base station. Assume that time is slotted and that only one user can be served in each time slot. Before each time slot, each user reports to the base station his channel condition, which determines the data transfer speed from the base station to the user. Based on this information and different statistics from the past, the wireless scheduler decides on the user to be scheduled for transmission in the next time slot.

Attacks on channel aware schedulers were investigated in recent studies such as [25] and [58] which revealed the vulnerability of PFS to delays/jitter and loss of throughput and time share caused to regular users by malicious users providing false channel capacity reports.

In the following text we outline two attacks on CS and on PFS.

1) ***CDF Scheduler and Coordinated Attacks****:* The CS algorithm [56] schedules (for transmission) the user whose transfer speed is the most *exceptional* among all users (compared to his past reports). The scheduler uses the CDF value of the current reported speed to determine how exceptional it is. Formally, let $r_i(t)$ be the data transfer speed of user i at time t and let R_i be a random variable of the transfer speeds of user i. Before time slot t, the scheduler calculates for each user $F_i(t) = Prob(r_i(t) > R_i)$. The user with the highest CDF value $F_i(t)$ is scheduled. The fairness CS enforces is derived from a fundamental property of CDF functions which states that regardless of the distribution of a random variable, its CDF function is distributed uniformly between zero and one. Since in every time slot the probability to be scheduled is equal among all users, CS enforces time share fairness. This unique property of CS also immunes it to malicious or selfish behavior, since regardless of the rates a user reports, his CDF value will always be uniformly distributed between zero and one as for all the others.

In [27] the authors show that while CS is immune to an attack by a single user, it is vulnerable to attacks by a coordinated group of malicious and selfish users. The vulnerability abused by the attack is based on the following observation: A malicious user who can predict the scheduling decisions, can report a fake low channel condition in time slots where he knows that he is not going to be scheduled anyway. These fake reports make his future reports, where he report his *real* channel condition, look more exceptional (his R_i is lower and therefore his F_i is higher) and thus increase his probability to be the one scheduled for transmission. While it is not assumed that a malicious user can predict the future, by cooperating with other users that share their future CDF value (in the next time slot) the user can predict some of the time slots where he is not going to be scheduled anyway. This way, such user breaks the fairness CS tries to enforce and gains a larger time share on the expense of other users (who are not in the group). One of the results presented in [27] is that when there are malicious users and regular users in equal numbers, the ratio of allocated time slots between a coordinated user and a regular one converges to $e - 1 \approx 1.7$ (instead of 1, since they are in equal numbers and CS maintains time share fairness). This attack belongs to the category of *Traffic Pattern Exploit* and causes in-attack damage. The fact that the time share of a malicious user can be 70% larger than this of a regular one, demonstrate the importance of the vulnerability analysis since it exposed the fact that CS allows large deviation from time share fairness while traditional performance analysis (which does not consider malicious environment) claims it is completely fair.

2) ***PFS and Retransmission Attack****:* Under PFS, before time slot t, the scheduler calculates a priority value for each user and schedules the one with the maximal priority. The priority value is given by $r_i(t)/A_i(t)$, where $A_i(t)$ is the throughput average of user i until time slot t (not including) measured in $bits/slot$ (and $r_i(t)$ is its channel rate as defined before).

A malicious user can use false reports in order to achieve a very high priority value in consecutive time slots. It is done by reporting a low channel rate $r_i(t)$ which leads to a low $A_i(t)$ value and then reporting a high channel rate. This allows the malicious user to obtain the highest priority value $r_i(t)/A_i(t)$ for consecutive time slots (until the growth of $A_i(t)$ reduces its priority value below those of others). This attack causes short term starvation of innocent users, that even if they will be compensated later for it, they suffer from delays that decrease their overall throughput. Such an attack can decrease TCP throughput by 25%-30% [25] and leave VoIP applications unable to operate [58].

The retransmission algorithm defines how the scheduler handles a report of a lost frame (NACK). While PFS describes how to transmit new pending transmissions, the way it should schedule retransmissions is open to interpretations. Should the scheduler retransmit the frame immediately? Should a transmission that was reported by a user as lost count in his throughut average $A_i(t)$? The vast majority of existing studies on wireless schedulers ignore the retransmission mechanism by simplifying the model to one where packet losses do not occur, while the rest adopt straight forward implementations of the retransmission algorithm.

In [28], the authors present the common straight forward retransmission algorithms for PFS, expose their vulnerability to malicious attacks and present an immune variation that maintain the original fairness of PFS. The vulnerability common to these algorithms lies in the assumption that a user will never report a failed transmission if he received it successfully. After all, even a selfish user cannot benefit from receiving the same data frame twice. The problem is that malicious users can abuse that and require more retransmission than others, hence occupying more transmission time on the expense of others. This attack belongs to the category of *Algorithmic Worst-Case Exploit* since the number of retransmissions required by the attacker is the maximal number of retransmissions allowed (which is the worst-case). The attack hurts other users only during the attack itself, hence it causes in-attack damage. The analysis in [28] of the vulnerability to this malicious behavior shows that in typical settings[3] with malicious and regular users in equal numbers, the regular users lose 50% of their time share.

*3) **Discussion and Comparison**:* From the retrasnsmission attack on PFS we can learn that it is not enough to make a system immune only to selfish behavior, but also a malicious behavior must be mitigated. Unlike PFS, CS is proved to be immune to malicious attacker as well. However, as shown earlier, coordination between malicious or selfish users can enable new attack capabalities and should not be overlooked.

In addition, the PFS case shows how analyzing an algorithm in a simplified model, while leaving the some of its aspects open to interpretations, can open the door to unexpected attack strategies. Since CS does not keep track of the throughput average a user receives, one does not have to decide if and how to count restransmissions that were reported as lost. Therefore, CS can be proved to be immune to such an attack (in the steady state). However, this unique property (of scheduling without keeping track of the scheduling history) allows attacks, such as the coordinated attack we described, in which users can gain additional time share and througput on the expense of others.

[3]Where the maximal number of frame retransmissions is 5.

V. Solutions

In this section we discuss general approaches to sophisticated DDoS. We note that many of the specific attacks mentioned above have tailor made solutions [59], [60], [61]. However, due to diversity of possible attacks, we are more interested here in understanding whether there are general approaches that take the bull by the horns.

A key factor in all the attacks discussed earlier, is the fact that there is a huge difference between the common case performance to the worst-case of the system/protocol/algorithm. An obvious solution approach is to design a better algorithm, system or protocol that avoids this worst-case performance. However, this is not always feasible. In many cases problems seem to be characterized by inherent trade-offs, where algorithmic solutions that work the best in the worst-case, underperform in the common case (in comparison to algorithms that were designed to deal best with the common case). Thus in many cases it might be reasonable to prefer the common-case good performance, at the expense of leaving the system vulnerable to attack as a calculated risk.

One solution to this dilemma is to design a system that gets the best of both worlds, by running different algorithms in the different cases. MCA^2 [62], is a such solution. The paper takes advantage of the emerging multi-core computer architecture to design a general framework for mitigating network-based complexity attacks. In this architecture, cores quickly identify packets with packets crafted as heavy that aim hurting the system performance, and divert them to a fraction of the cores that are dedicated to handle all the heavy messages. These cores run algorithms that handle the worst-case efficiently. Thus, the rest of the cores remain relatively unaffected and free to provide the legitimate traffic with the same quality of service as if no attack took place.

We note that detecting whether an algorithm/system/protocol is vulnerable is not so easy in many cases. Complexity attack is easy to discover, since it exploits in a straight forward manner the the difference between the worst-case and the average-case complexity. However, finding the worst-case traffic pattern (or the system weakest point or the protocol worst-case) - is a more challenging task.

The paper [63] deals with this problem and develops a method to help discover manipulation attacks in protocol implementations, using compilation tools. The method is based on a novel combination of static analysis with symbolic execution and dynamic analysis with concrete execution. The former finds code paths that are likely vulnerable, and the latter emulates adversarial actions that lead to effective attacks. By discovering the potential protocol attacks, the paper performs the first step towards finding solutions to the attack.

VI. Concluding remarks and Discussion

We presented the subject of system performance under malicious behavior and argued that such performance may differ drastically from that derived using the traditional approaches. We explored an array of traditional computer and network systems and demonstrated their vulnerability to performance attacks, which, depending on the system involved, could be quite drastic.

Our work tried to convey some insights into the important question of what distinguishes one design from another, making the former highly vulnerable and the latter resilient. The cases we addressed suggest the following insight: A ”simple” design whose rules are ”simple” and whose structure is ”uniform” is likely to be more resilient. The reason is that all transactions under such a structure are usually simple and identical to each other and therefore an attacker cannot benefit (in comparison to a regular user) from conducting specific transactions. Examples include the Open Hash and the Aloha system which are quite resilient and whose transactions are very simple and uniform.

In contrast, a more ”sophisticated design” may make the system more vulnerable since the attacker can hand-pick the expensive transactions (or an expensive series of transactions). An example is the Binary Tree algorithm where the transaction sequencing is sophisticated and is exploited by the attacker.

It remains as an open research question, whether a more exact characterization of system design could be made to assist in predicting the vulnerability level of the design, and guidelines can be provided to yield resilient designs. Another open question is how one can benefit from all worlds by making a design whose average-case performance as well as malice-case performance are both good.

References

[1] L. Kleinrock, *Comunications Nets: Stochastic Message Flow and Delay*. McGraw Hill (New York), 1964. Out of print. reprinted by Dover Publications, 1972.

[2] ——, *Queueing Systems*. Wiley Interscience, 1976, vol. II: Computer Applications.

[3] D. Bertsekas and R. Gallager, *Data Networks*. Prentice-Hall International, 1992.

[4] M. Harchol-Balter, *Performance Analysis and Design of Computer Systems*. Cambridge University Press, 2013.

[5] C. Labovitz, D. McPherson, and F. Jahanian, "Infrastructure Attack Detection and Mitigation," in *ACM SIGCOMM Conference on Applications*, Aug. 2005.

[6] "Digital attack map," http://www.digitalattackmap.com/understanding-ddos/.

[7] S. A. Crosby and D. S. Wallach, "Denial of Service via Algorithmic Complexity Attacks," in *USENIX*, Aug. 2003.

[8] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "Evaluating the Vulnerability of Network Mechanisms to Sophisticated DDoS Attacks," in *INFOCOM*, Apr. 2008.

[9] M. D. McIlroy, "A Killer Adversary for Quicksort," *Software–Practice and Experience*, pp. 341–344, 1999.

[10] T. Peters, "Algorithmic Complexity Attack on Python," May 2003, http://mail.python.org/pipermail/python-dev/2003-May/035916.html.

[11] M. Fisk and G. Varghese, "Fast Content-Based Packet Handling for Intrusion Detection," UCSD, CA, USA, Tech. Rep., 2001.

[12] R. Smith, C. Estan, and S. Jha, "Backtracking Algorithmic Complexity Attacks Against a NIDS," in *ACSAC*, Dec. 2006.

[13] F. Weimer, "Algorithmic Complexity Attacks and the Linux Networking Code," http://www.enyo.de/fw/security/notes/linux-dst-cache-dos.html.

[14] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources," in *ICNP*, Mar. 2004.

[15] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of Quality (RoQ) Attacks on Internet End-Systems," in *INFOCOM*, Mar. 2005.

[16] A. Kuzmanovic and E. W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew VS. the Mice and Elephants)," in *ACM SIGCOMM Conference on Applications*, Aug. 2003.

[17] A. Bremler-Barr, H. Levy, and N. Halachmi, "Aggressiveness Protective Fair Queueing for Bursty Applications," in *IWQoS*, Jun. 2006.

[18] E. Doron and A. Wool, "Wda: A Web Farm Distributed Denial of Service Attack Attenuator," *Comput. Netw.*, vol. 55, pp. 1037–1051, April 2011.

[19] C. Castelluccia, E. Mykletun, and G. Tsudik, "Improving Secure Server Performance by Re-balancing SSL/TLS Handshakes," in *USENIX*, Apr. 2005.

[20] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *USENIX*, Jun. 2007.

[21] A. Bremler-Barr, Y. Harchol, and D. Hay, "Space-time tradeoffs in software-based deep packet inspection," in *IEEE HPSR*, 2011.

[22] Y. Afek and H. Nussbacher, "Ripe-41 conference amsterdam 1/2002, ddos tutorial."

[23] J. Bellardo and S. Savage, "Abstract 802.11 Denial-of-Service Attacks: Real Vulnerabilities and Practical Solutions," in *USENIX*, Jun. 2003.

[24] "Advanced 802.11 attack," black Hat Briefings, July 2002.

[25] S. Bali, S. Machiraju, H. Zang, and V. Frost, "A Measurement Study of Scheduler-Based Attacks in 3G Wireless Networks," in *PAM*, 2007.

[26] R. Racic, D. Ma, H. Chen, and X. Liu, "Exploiting Opportunistic Scheduling in Cellular Data Networks," in *NDSS*, Feb. 2008.

[27] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "On the Exploitation of CDF Based Wireless Scheduling," in *INFOCOM*, Apr. 2009.

[28] ——, "On the Vulnerability of the Proportional Fairness Scheduler to Retransmission Attacks," in *INFOCOM*, Apr. 2011.

[29] T. Dubendorfer, A. Wagner, and B. Plattner, "An economic damage model for large-scale internet attacks," in *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004. WET ICE 2004. 13th IEEE International Workshops on*, 2004, pp. 223–228.

[30] R. Vasudevan, Z. Mao, O. Spatscheck, and J. Van der Merwe, "Midas: An impact scale for ddos attacks," in *Local Metropolitan Area Networks, 2007. LANMAN 2007. 15th IEEE Workshop on*, 2007, pp. 200–205.

[31] J. Mirkovic, S. Fahmy, P. Reiher, R. Thomas, A. Hussain, S. Schwab, and C. Ko, "P.: Measuring impact of dos attacks," in *In: Proceedings of the DETER Community Workshop on Cyber Security Experimentation. (2006*.

[32] L. Kleinrock, "Analysis of a time shared processor," *Naval Research Logistics Quarterly*, vol. 11, no. 1, pp. 59–73, March 1964.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.

[34] S. Kumar, J. S. Turner, and P. Crowley, "Peacock hashing: Deterministic and updatable hashing for high performance networking." in *INFOCOM*. IEEE, 2008, pp. 101–105.

[35] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *Journal of Algorithms*, 2001, p. 2004.

[36] M. Mitzenmacher and A. Broder, "Using multiple hash functions to improve ip lookups," in *In Proceedings of IEEE INFOCOM*, 2000, pp. 1454–1463.

[37] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *SIGCOMM Comput. Commun. Rev.*, vol. 35, pp. 181–192, August 2005.

[38] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed ip routing lookups," in *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '97. ACM, 1997, pp. 25–36.

[39] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 245–256, August 2004.

[40] T. N. Thinh and S. Kittitornkun, "Massively parallel cuckoo pattern matching applied for nids/nips," in *Electronic Design, Test and Application, 2010. DELTA '10. Fifth IEEE International Symposium on*, jan. 2010, pp. 217 –221.

[41] A. Kirsch, M. Mitzenmacher, and G. Varghese, *Hash-Based Techniques for High-Speed Packet Processing*. SPRINGER, 2010.

[42] U. Ben-Porat, A. Bremler-Barr, H. Levy, and B. Plattner, "On the vulnerability of hardware hash tables to sophisticated attacks," in *Networking (1)*, 2012, pp. 135–148.

[43] U. Barros, M. Bouet, A. Santos, and M. Nogueira, "Assessing roq attacks on manets over aware and unaware tpc techniques," in *Network and Service Management (CNSM), 2011 7th International Conference on*, 2011, pp. 1–5.

[44] M. Guirguis, "Bandwidth stealing via link targeted roq attacks," in *in Proceedings of CCN04: The 2nd IASTED International Conference on Communication and Computer Networks*, 2004.

[45] R. Rom and M. Sidi, *Multiple Access Protocols: Performance and Analysis*. Springer Verlag, New York, 1990.

[46] N. Abramson, "The aloha system: Another alternative for computer communications," in *Proceedings of the November 17-19, 1970, Fall Joint Computer Conference*, ser. AFIPS '70 (Fall). New York, NY, USA: ACM, 1970, pp. 281–285.

[47] L. G. Roberts, "Extensions of packet communication technology to a hand held personal terminal," in *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, ser. AFIPS '72 (Spring). New York, NY, USA: ACM, 1972, pp. 295–298.

[48] L. Kleinrock and F. Tobagi, "Carrier sense multiple access for packet switched radio channels," in *Conference Record, International Conference on Communications*, Minneapolis, Minnesota, June 1974, pp. 21B/1–21B/7.

[49] J. Capetanakis, "Tree algorithms for packet broadcast channels," *IEEE Transactions on Information Theory*, vol. 25, no. 5, pp. 505–515, 1979.

[50] Y. Zhang, "Low-rate tcp-targeted dos attack disrupts internet routing," in *In Proc. 14th Annual Network and Distributed System Security Symposium*, 2007.

[51] "Cisco security advisory: Tcp state manipulation denial of service vulnerabilities in multiple cisco products," http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20090908-tcp24, advisory ID: cisco-sa-20090908-tcp24.

[52] "Cisco security advisory: Tcp vulnerabilities in multiple ios-based cisco products," http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20040420-tcp-ios, advisory ID: cisco-sa-20040420-tcp-ios.

[53] V. Gill, J. Heasley, D. Meyer, P. Savola, and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)," RFC 5082 (Proposed Standard), Internet Engineering Task Force, October 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5082.txt

[54] A. Cobham, "Priority assignment in waiting line problems," *J. Oper. Res. SOc. Am.*, vol. 2, p. 70, 1954.

[55] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and analysis of bgp behavior under stress," in *In Proc. ACM SIGCOMM Internet Measurement Workshop*, 2002.

[56] D. Park, H. Seo, H. Kwon, and B. G. Lee, "Wireless Packet Scheduling Based on the Cumulative Distribution Function of User Transmission Rates," *IEEE Transactions on Communications*, pp. 1919 – 1929, 2005.

[57] R. P. A. Jalali and R. Pankaj, "Data throughput of CDMA-HDR: A high efficiency high data rate personal communication wireless system," in *IEEE Vehicular Technology Conference*, 2000.

[58] H. C. Radmilo Racic, Denys Ma and X. Liu, "Exploiting opportunistic scheduling in cellular data network," in *NDSS*, 2008.

[59] Y. Chen, K. Hwang, and Y.-K. Kwok, “Filtering of shrew ddos attacks in frequency domain,” in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*. IEEE, 2005, pp. 8–pp.

[60] Y. Chen and K. Hwang, “Spectral analysis of tcp flows for defense against reduction-of-quality attacks,” in *Communications, 2007. ICC’07. IEEE International Conference on*. IEEE, 2007, pp. 1203–1210.

[61] ——, “Collaborative detection and filtering of shrew ddos attacks using spectral analysis,” *Journal of Parallel and Distributed Computing*, vol. 66, no. 9, pp. 1137–1151, 2006.

[62] Y. Afek, A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, “Mca 2: multi-core architecture for mitigating complexity attacks,” in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2012, pp. 235–246.

[63] N. Kothari, R. Mahajan, T. Millstein, R. Govindan, and M. Musuvathi, “Finding protocol manipulation attacks,” *SIGCOMM-Computer Communication Review*, vol. 41, no. 4, p. 26, 2011.