Network and Computer Performance in Malicious Environments: The Good, the Bad and the Ugly

Udi Ben-Porat, Anat Bremler-Barr, Hanoch Levy

Network and Computer Performance in Malicious Environments: The Good, the Bad and the Ugly

Udi Ben-Porat
Computer Engineering and Networks Laboratory (TIK)
ETH, Zurich, Switzerland
Email: ehudb@tik.ee.ethz.ch

Anat Bremler-Barr
Computer Science Dpt.
Interdisciplinary Center, Herzliya, Israel
Email: bremler@idc.ac.il

Hanoch Levy
Balvatnik School of Computer Science
Tel-Aviv University, Tel-Aviv, Israel
Email: hanoch@post.tau.ac.il

***Abstract*—Performance analysis and the design of computer and networking systems have traditionally accounted for the stochastic nature of the problem addressed and been based on stochastic type analysis, mainly expected value ("the good"). In some related disciplines, mainly computer science and algorithmic design, worst-case analysis ("the bad") has been popular. In recent years we have experienced a wave of DDoS and Cyber attacks threatening the welfare of the internet. These are launched by malicious users whose only incentive is to degrade the performance of other, innocent, users. This has triggered a new direction of research aiming at evaluating system performance while accounting for the malicious behavior of the attackers ("the ugly"). The performance metrics in this case differs from both the average-case and the worst-case and can affect system design considerably.**

The purpose of this work is to expose and discuss this new analysis approach as well as to distinguish it from the traditional approaches. We use a wide array of cases and results derived in the literature to demonstrate how such analysis can be carried out. We further use them to show what kind of metrics can be used to evaluate the effect of malicious behavior and the resilience of the system against them.

I. Introduction

Performance analysis of computer and network systems, as well as their design, has traditionally been based on stochastic analysis. The underlying assumption behind this methodology is that customers (alternatively requests, packets 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. One can denote it, in short, as average-case analysis ("the good"). In related fields, such as computer science and algorithmic design, worst-case analysis ("the bad") has been popular. Both these methodologies have been usually based on the underlying paradigm that all users are innocent, aiming merely at improving their own performance.

In recent years 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 (Labovitz et al. [1]). In performance evaluation terms, this has broken the underlying paradigm by which all users are "innocent".

This paradigm shift has lead to the introduction of a new research direction aiming at evaluating system performance in the presence of malicious users ("the ugly"). The effect of malicious behavior on system performance and design can be, in some cases, quite drastic. Our objective in this work is to present this new malicious-based performance analysis, to discuss it and to demonstrate it through reviewing a collection of results derived in recent years.

In the rest of this section we describe malicious performance and a vulnerability metrics. In Section II we classify the attack types and in Section III we review several typical systems and their malicious performance vulnerability.

A. Malicious-Based Performance

How does "malicious-based" 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 malicious-based 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 malicious-based 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.

B. 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 [2] 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 [3]. 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.

C. Malicious 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 (such as Reduction of Quality attacks like [3], [4]). 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 t 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.

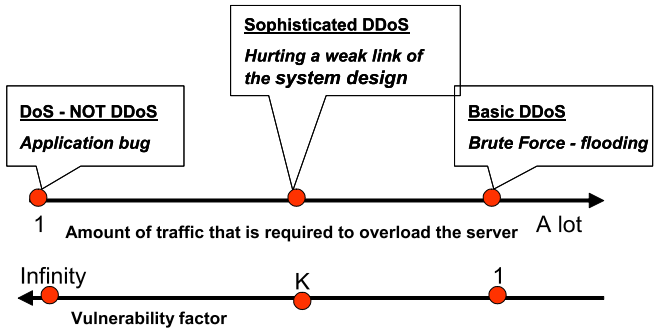

Fig. 1. Attack Effectiveness and Corresponding Vulnerability.

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.

II. Attack Survey and Classification

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[5] were the first to demonstrate the complexity attack on the commonly used Open Hash data structure, where 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 [2], Quicksort [6], regular expression matcher [7] and Intrusion Detection Systems [8], [9].

2) **Traffic Pattern Exploit** - Attacker exploits the (stochastic) worst case traffic pattern that can be applied to the system. This case 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 [3], [4], [9]. 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 [10] 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 [11], [12]. 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.

3) **System Architecture Worst-Case Exploit** - While the previous categories exploit a weak point in the algorithm, this category exploits 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 [13] 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. [14] shows vulnerability in the pattern matching of NIDS by using traffic that 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 [15] of SYN attack and DNS attack types, 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 amounts of SYNs but does not send the FIN.

Another type of a protocol deviation attack, is to obey the protocol flows but to "cheat" in the messages of the protocols. Recent works in the context of wireless scheduling [16], [17], [18], [19], show that "cheating" by misreporting the channel capacity can harm the system performance significantly.

III. Practical systems and attacks: Performance Degradation and Vulnerability

Having described the vulnerability measures in section I and the attacks classification in section II, we will now demonstrate the performance effect and vulnerabilities of a selected sets of attacks on wireless schedulers and hash tables.

Channel-aware scheduling strategies - such as the CDF Scheduler (CS) [20] and the popular Proportional Fairness Scheduler (PFS) [21] - 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.

A. Coordinated Attack on CDF Scheduler

The CS algorithm [20] 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.

In [18] 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 (or 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 [18] 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.

B. Retransmission Attack on PFS

The retransmission algorithm defines how the scheduler handles a report of a lost frame (NACK). Should the scheduler retransmit the frame immediately? should he first send to others before he tries to retransmit the lost frame? 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 [19], the authors present the common straight forward retransmissions algorithms for PFS, expose their vulnerability to malicious attacks and present immune variations 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 [19] of the vulnerability to this malicious behavior shows that in typical settings[1] with malicious and regular users in equal numbers, the regular users lose 50% of their time share.

C. Attack on 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 [2] 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. During an insertion in Closed Hash with linear probing, if a key is hashed into an occupied bucket, then the neighboring bucket is probed, and this process repeats until an empty bucket is found [22]. The average complexity of performing an insert operation is $O(1)$ memory accesses while the worst case complexity is $O(N)$ memory accesses, where N is the number of existing elements in the Hash table. The worst case is that an insertion of K keys are all 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 [2]. 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 $O(K)$. In addition, after the attack has ended, a cluster 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 after the attack, therefore, this attack causes also post-attack damage. The analysis in [2] 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. In addition, while Closed Hash is extremely vulnerable to post-attack damage, [2] shows that Open Hash table does not suffer from post-attack damage, while according to traditional performance analysis they are almost identical.

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

IV. Summary

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 described a performance metrics that evaluates the vulnerability of system performance to such attacks. We classified the attack types and the various performance effects they can have. We demonstrated the methodology on wireless systems and hash tables. We showed that malicious behavior can affect drastically system performance and that systems that otherwise are considered to be well designed may be quite vulnerable, that is, their performance under malicious behavior can degrade severely.

References

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

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

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

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

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

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

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

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

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

[10] 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.

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

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

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

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

[15] Y. Afek and H. Nussbacher, "Ripe-41 conference amsterdam 1/2002, ddos tutorial."
[16] S. Bali, S. Machiraju, H. Zang, and V. Frost, "A Measurement Study of Scheduler-Based Attacks in 3G Wireless Networks," in *PAM*, 2007.
[17] R. Racic, D. Ma, H. Chen, and X. Liu, "Exploiting Opportunistic Scheduling in Cellular Data Networks," in *NDSS*, Feb. 2008.
[18] U. Ben-Porat, A. Bremler-Barr, and H. Levy, "On the Exploitation of CDF Based Wireless Scheduling," in *INFOCOM*, Apr. 2009.
[19] ——, "On the Vulnerability of the Proportional Fairness Scheduler to Retransmission Attacks," in *INFOCOM*, Apr. 2011.
[20] 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.
[21] 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.
[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw-Hill, 2001.