IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, July 2022
Impact Factor 6.252

Parasitic Computing

Mr. Pradeep Chavan
Student
Bharati Vidyapeeth, Navi Mumbai, India

Abstract: *Communication on the Internet is guaranteed by a standard set of protocols, used by all computers. Here we show that these protocols can be exploited to compute with the communication infrastructure, transforming the Internet into a distributed computer in which servers unwittingly perform computation on behalf of a remote node. In this model, which we call 'parasitic computing', one machine forces target computers to solve a piece of a complex computational problem merely by engaging them in standard communication. Consequently, the target computers are unaware that they have performed computation for the benefit of a commanding node. As experimental evidence of the principle of parasitic computing, we harness the power of several web servers across the globe, which-unknown to them-work together to solve an NP complete problem.*

Keywords: Plastic Computing

I. INTRODUCTION

The net is a fertile place where new ideas/products surface quite often. We have already come across many innovative ideas such as Peer-to-Peer file sharing, distributed computing etc. Parasitic computing is a new in this category. Reliable communication on the Internet is guaranteed by a standard set of protocols, used by all computers. The Notre Dame computer scientist showed that these protocols could be exploited to compute with the communication infrastructure, transforming the Internet into a distributed computer in which servers unwittingly perform computation on behalf of a remote node.

In this model, known as “parasitic computing”, one machine forces target computers to solve a piece of a complex computational problem merely by engaging them in standard communication. Consequently, the target computers are unaware that they have performed computation for the benefit of a commanding node. As experimental evidence of the principle of parasitic computing, the scientists harnessed the power of several web servers across the globe, which–unknown to them–work together to solve an NP complete problem.

Sending a message through the Internet is a sophisticated process regulated by layers of complex protocols. For example, when a user selects a URL (uniform resource locator), requesting a web page, the browser opens a transmission control protocol (TCP) connection to a web server. It then issues a hyper-text transmission protocol (HTTP) request over the TCP connection. The TCP message is carried via the Internet protocol (IP), which might break the message into several packages, which navigate independently through numerous routers between source and destination. When an HTTP request reaches its target web server, a response is returned via the same TCP connection to the user's browser. The original message is reconstructed through a series of consecutive steps, involving IP and TCP; it is finally interpreted at the HTTP level, eliciting the appropriate response (such as sending the requested web page). Thus, even a seemingly simple request for a web page involves a significant amount of computation in the network and at the computers at the end points.

In essence, a `parasitic computer' is a realization of an abstract machine for a distributed computer that is built upon standard Internet communication protocols. We use a parasitic computer to solve the well known NP-complete satisfiability problem, by engaging various web servers physically located in North America, Europe, and Asia, each of which unknowingly participated in the experiment. Like the SETI@home project, parasitic computing decomposes a complex problem into computations that can be evaluated independently and solved by computers connected to the Internet; unlike the SETI project, however, it does so without the knowledge of the participating servers. Unlike `cracking' (breaking into a computer) or computer viruses, however, parasitic computing does not compromise the security of the targeted servers, and accesses only those parts of the servers that have been made explicitly available for Internet communication.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, July 2022
Impact Factor 6.252

II. THEORY

To solve many NP complete problems, such as the traveling salesman or the satisfiability problem, a common technique is to generate a large number of candidate solutions and then test the candidates for their adequacy. Because the candidate solutions can be tested in parallel, an effective computer architecture for these problems is one that supports simultaneous evaluation of many tests.

Four Notre Dame professors recently discovered a new Internet vulnerability that is commonly known as "parasitic computing." The researchers found a way to "trick" Web servers around the world into solving logic math problems without the server's permission. The researchers found that they could tag a logic problem onto the check sum (the bit amount that is sent when a Web page is requested) and the Web server would process the request. When a Web page was requested without the correct check sum, the server would not respond to the request.

Each of the math problems that were tagged on to the request by the researchers was broken down into smaller pieces that were evaluated by servers in North America, Europe and Asia. The results from each were used to build a solution. Using a remote server, the team divided the problem into packages, each associated with a potential answer. The bits were then hidden inside components of the standard transmission control protocol of the Internet, and sent on their merry way.

The major discovery in this experiment is that other computers are answering logical questions without knowledge of doing so. The work is performed without consent, creating an ethical dilemma. The technique does not violate the security of the unknowing server; it only uses areas that are open for public access. They find it useful because they found a way to use a computer elsewhere to solve a problem.

Here, the computer consists of a collection of target nodes connected to a network, where each of the target nodes contains an arithmetic and logic unit (ALU) that is capable of performing the desired test and a network interface (NIF) that allows the node to send and receive messages across the network. A single home parasite node initiates the computation, sends messages to the targets directing them to perform the tests, and tabulates the results.

Owing to the many layers of computation involved in receiving and interpreting a message, there are several Internet protocols that, in principle, could be exploited to perform a parasitic computation. For example, an IP-level interpretation could force routers to solve the problem, but such an implementation creates unwanted local bottlenecks. To truly delegate the computation to a remote target computer, we need to implement it at the TCP or higher levels. Potential candidate protocols include TCP, HITP, or encryption/ decryption with secure socket layer (SSL).

III. IMPLEMENTATION

In our implementation a single master node controls the execution of the algorithm. There are several ways to implement the basic algorithm discussed above. The two major choices are (a) concurrency and (b) connection reuse. Regarding (a), the master node can have many computations occurring in the web concurrently. Each concurrent computation requires a separate TCP connection to a HTTP host. Regarding (b), before a TCP connection can be used, it must be established. Once established, TCP segments can be sent to the remote host. When multiple guesses are sent in one connection, it is impossible to know to which guess a correct solution refers to. For example, suppose guess <b1 , c1 > and < b2 , c2> are sent one after the other in a single connection. Further suppose that only one solution is correct. We expect to get one response back. But we cannot tell to which solution the response refers.

The implementation used in this paper is a prototype that is not designed for efficiency of execution. In our prototype implement there is no concurrency and each connection is used for exactly one computation.

IV. RELIABLE COMMUNICATIONS

Any message can get lost. In a reliable system, the sender of a message saves a copy of the message and waits for an acknowledgement of the message. If after some time, the sender has not received an acknowledgement, it will re-send the message (from the copy). The sender will continue to do this until an acknowledgement is received.

In general, there is no upper bound on how long a message might take to be delivered. Consequently, in a distributed system, it is not possible to distinguish between a lost message and a delayed message. Therefore, a message is assumed lost after some time-out period. A time-out value that is too small declares too many delayed messages as lost. On the other hand, a value that is too large unnecessarily slows down the system.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 1, July 2022
Impact Factor 6.252

Our exploit circumvents the reliability mechanism in TCP. Furthermore, because an invalid solution fails the checksum, it is as if it never arrived. Therefore, the receiver will not send an acknowledgement of the message. There are two undesirable outcomes that could occur:

A false negative occurs when a packet for a valid solution is dropped due to a data corruption or congestion.

A false positive occurs when a bit error changes an invalid result into an valid result

The latter is very rare statistically and all but impossible in practice. Although the former is also unusual, it is frequent enough that it should be considered further.

First, let's consider the errors that are caught by the TCP checksum. Every transmission link (hardware devices such as ethernet) computes a checksum on its packets. The TCP checksum catches errors that pass the link checksum, but still have some data corruption. Because the data was not damaged in transmission (where it would have been caught by the transmission link checksum), it must have occurred in an intermediate system

V. CONCLUSION

Parasitic computing moves computation onto what is logically the communication infrastructure of the Internet, blurring the distinction between computing and communication. The Notre Dame scientists have shown that the current Internet infrastructure permits one computer to instruct other computers to perform computational tasks that are beyond the target's immediate scope. Enabling all computers to swap information and services they are needed could lead to unparalleled emergent behavior, drastically altering the current use of the Internet.

The implementation offered above represents only a proof of concept of parasitic computing. As such, the solution merely serves to illustrate the idea behind parasitic computing, and it is not efficient for practical purposes in its current form.

Indeed, the TCP checksum provides a series of additions and a comparison at the cost of hundreds of machine cycles to send and receive messages, which makes it computationally inefficient. To make the model viable, the computation-to-communication ratio must increase until the computation exported by the parasitic node is larger than the amount of cycles required by the node to solve the problem itself instead of sending it to the target.

However, these are drawbacks of the presented implementation and do not represent fundamental obstacles for parasitic computing. It remains to be seen, however, whether a high- level implementation of a parasitic computer, perhaps exploiting HTTP or encryption/ decryption could execute in an efficient manner.

REFERENCES

[1]. Kogge, P.M. The Architecture of Symbolic Computers. McGraw-Hill, New York, 1991.

[2]. Peterson, L.L. and Davie, B.S. Computer Networks, a Systems Approach, Second Edition. Morgan Kaufmann, San Francisco, California, 2000.

[3]. Boole, G. An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities. 1854.

[4]. Alderman, L.M. Molecular computation of solutions to combinatorial problems. Science, 266, 1021-1024 (1994).

[5]. Ouyang, Q., Kaplan, P.D., Liu, S., Libchaber, A. DNA solution of the maximal clique prob- lem. Science,278, 446-449 (1997).

[6]. Sch¨oning, U. in Proc. 40th Ann. IEEE Conf. Found. Comp. Sci. (FOCS) 410-414 (IEEE Comp. Sci. Los Alamitos, California, 1999).

[7]. Stevens, W. R. TCP/IP Illustrated. (Addison-Wesley, Reading, Massachusetts, 1994).

[8]. Stone, J. and Partridge, C. When the CRC and TCP checksum disagree. In SIGCOMM 2000, September 2000.

[9]. Stone, J., Greenwald, M., Partridge, C., and Hughes, J. Performance of checksums and CRCs over real data. IEEE Trans. on Networks, October 1998.

[10]. Foster, I. Internet computing and the emerging grid. Nature web matters (http://www.nature.com/nature/webmatters/grid/grid.html) (2000).