than any previously discussed DR mechanism. Our cross-sections are in fair agreement with the range of FALP measurements but even our upper-bound cross-section remains ten times smaller than the storage-ring experiments that are expected to be highly reliable. The present calculation has not separated out the *para-* and *ortho-*contributions, which simplifies our theoretical description by allowing us to average over all nuclear permutation symmetries without imposing them explicitly.

One candidate process to consider in the future is the set of DR pathways excited by the inverse of rotational autoionization. Those pathways require a quantized treatment of the ionic rotational channels, which has not yet been incorporated into our present calculations. Rotationally autoionizing *np* resonances are known to have some of the largest widths observed for H_3 Rydberg states[17–19,28], but they do not necessarily generate a high DR contribution because these states probably autoionize back to the electron continuum before they have time to dissociate. An investigation of this pathway may help to understand why current generation storage-ring experiments with rotationally hot target ions appear to have a much larger DR rate than the FALP experiments. For example, in ref. 4, $kT_{\rm rot} = 0.23$ eV, which implies a peak in the rotational population at the $J^+ = 5$ state. For target ions in the $J^+ = 5$ incident channel, electron capture can occur via inverse rotational autoionization at energies up to the $J^+ = 7$ ionic threshold, which is about 0.13 eV.

Although the present study has not solved the problem of why different measurements of the low-energy H_3^+ DR rate give such different results, our hope is that this identification of the dominant DR mechanism will stimulate further progress in both experiment and theory in the near future. □

Received 4 April; accepted 4 July 2001.

1. Larsson, M. Experimental studies of the dissociative recombination of H_3^+. *Phil. Trans. R. Soc. Lond. A* **358,** 2433–2443 (2000).
2. Larsson, M. *et al.* Direct high-energy neutral-channel dissociative recombination of cold H_3^+ in an ion storage ring. *Phys. Rev. Lett.* **70,** 430–433 (1993).
3. Datz, S. *et al.* Branching-processes in the dissociative recombination of H_3^+. *Phys. Rev. Lett.* **74,** 896–899 (1995).
4. Strasser, D. *et al.* Two- and three-body kinematical correlation in the dissociative recombination of H_3^+. *Phys. Rev. Lett.* **86,** 779–782 (2001).
5. Sundström, G. *et al.* Destruction rate of H_3^+ by low-energy electrons measured in a storage-ring experiment. *Science* **263,** 785–787 (1994).
6. Tanabe, T. *et al.* in *Dissociative Recombination: Theory, Experiment and Applications* Vol. IV (eds Larsson, M. Mitchell, J. B. A. & Schneider, I. F.) 170–179 (World Scientific, Singapore, 2000).
7. Jensen, M. J. *et al.* Dissociative recombination and excitation of H_3^+. *Phys. Rev. A* **63,** 052701-1–052701-5 (2001).
8. Laubé, S. *et al.* New FALP-MS measurements of H_3^+, D_3^+ and HCO^+ dissociative recombination. *J. Phys. B* **31,** 2111–2128 (1998).
9. Glosík, J., Plasil, R., Poterya, V., Kurdna, P. & Tichý, M. The recombination of H_3^+ ions with electrons: dependence on partial pressure of H_2. *Chem. Phys. Lett.* **331,** 209–214 (2000).
10. Amano, T. The dissociative recombination rate coefficients of H_3^+, D_3^+, and HCO^+. *J. Chem. Phys.* **92,** 6492–6501 (1990).
11. McCall, B. J., Geballe, T. R., Hinkle, K. H. & Oka, T. Detection of H_3^+ in the diffuse interstellar medium toward Cygnus OB2 No. 12. *Science* **279,** 1910–1913 (1998).
12. Oka, T. Astronomy, physics and chemistry of H_3^+—Introductory remarks. *Phil. Trans. R. Soc. Lond. A* **358,** 2363–2369 (2000).
13. Geballe, T. R. H_3^+ between the stars. *Phil. Trans. R. Soc. Lond. A* **358,** 2503–2512 (2000).
14. Schneider, I. F., Orel, A. E. & Suzor-Weiner, A. Channel mixing effects in the dissociative recombination of H_3^+ with slow electrons. *Phys. Rev. Lett.* **85,** 3785–3788 (2000).
15. Orel, A. E., Schneider, I. F. & Suzor-Weiner, A. Dissociative recombination of H_3^+: progress in theory. *Phil. Trans. R. Soc. Lond. A* **358,** 3293–3293 (2000).
16. Staib, A. & Domcke, W. Analysis of the Jahn–Teller effect in the np^2E' Rydberg series of H_3 and D_3. *Z. Phys. D* **16,** 275–282 (1990).
17. Bordas, M. C., Lembo, L. J. & Helm, H. Spectroscopy and multichannel quantum-defect theory analysis of the *np* Rydberg series of H_3. *Phys. Rev. A* **44,** 1817–1827 (1991).
18. Stephens, J. A. & Greene, C. H. Quantum-defect description of H_3 Rydberg state dynamics. *Phys. Rev. Lett.* **72,** 1624–1627 (1994).
19. Stephens, J. A. & Greene, C. H. Quantum-defect description of H_3 and the Jahn–Teller effect. *J. Chem. Phys.* **102,** 1579–1591 (1995).
20. Bates, D. R. Dissociative recombination when potential energy curves do not cross. *J. Phys. B* **25,** 5479–5488 (1992).
21. Cecchi-Pestellini, C. & Dalgarno, A. H_3^+ in diffuse interstellar gas. *Mon. Not. R. Astron. Soc.* **313,** L6–L8 (2000).
22. Jaquet, R., Cencek, W., Kutzelnigg, W. & Rychlewski, J. Sub-microhartree accuracy potential energy surface for H_3^+ including adiabatic and relativistic effects II. Rovibrational analysis for H_3^+ and D_3^+. *J. Chem. Phys.* **108,** 2837–2846 (1998).
23. Esry, B. D., Lin, C. D. & Greene, C. H. Adiabatic hyperspherical study of the helium trimer. *Phys. Rev. A* **54,** 394–401 (1996).
24. Lin, C. D. Hyperspherical coordinate approach to atomic and other coulombic 3-body systems. *Phys. Rep.* **257,** 2–83 (1995).
25. Zhou, Y., Lin, C. D. & Shertzer, J. Hyperspherical approach to coulombic 3-body systems with different masses. *J. Phys. B* **26,** 3937–3949 (1993).
26. Greene, C. H. & Jungen, Ch. Molecular applications of quantum defect theory. *Adv. At. Mol. Phys.* **21,** 51–121 (1985).
27. Jungen, Ch. *Molecular Applications of Quantum Defect Theory* (Institute of Physics, Bristol, 1996).
28. Mistrík, I. *et al.* Ab initio analysis of autoionization of H_3 molecules using multichannel quantum-defect theory and new quantum-defect surfaces. *Phys. Rev. A* **61,** 033410-1–033410-16 (2000).
29. O'Malley, T. F. Theory of dissociative attachment. *Phys. Rev.* **150,** 14–29 (1966).
30. Giusti, A. A multichannel quantum defect approach to dissociative recombination. *J. Phys. B* **13,** 3867–1894 (1980).
31. Orel, A. E. & Kulander, K. C. Resonant dissociative recombination of H_3^+. *Phys. Rev. Lett.* **71,** 4315–4318 (1993).

Acknowledgements

We thank J. A. Stephens, A. E. Orel and A. Suzor-Weiner for discussions, and M. Larsson, G. Dunn, S. Datz, T. Oka and A. Dalgarno for comments on the experimental and astrophysical issues. Computational work was carried out at the National Energy Research Supercomputer Center at Lawrence Berkeley National Laboratory. This work was supported by the National Science Foundation.

Correspondence and requests for materials should be addressed to C.H.G. (e-mail: chris.greene@colorado.edu).

Parasitic computing

Albert-László Barabási*, **Vincent W. Freeh**†, **Hawoong Jeong*** **& Jay B. Brockman**†

* *Department of Physics; and* † *Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana 46556, USA*

Reliable communication on the Internet is guaranteed by a standard set of protocols, used by all computers[1]. 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[2].

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, that 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)[1]. 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. The success of the Internet rests on the cooperation of and trust between all involved parties.

Here we demonstrate that the trust-based relationships between machines connected on the Internet can be exploited to use the resources of multiple servers to solve a problem of interest without authorization. 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 (see http://www.seti.org), 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.

To solve many NP complete problems, such as the travelling 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. An example of such a machine is illustrated in Fig. 1a. 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).

During package transfer across the Internet, messages can be corrupted, that is, the bits can change. TCP contains a checksum that provides some data integrity of the message. To achieve this, the sender computes a checksum and transmits that with the message. The receiver also computes a checksum, and if it does not agree with the sender's, then the message was corrupted (see Fig. 2). One property of the TCP checksum function is that it forms a sufficient logical basis for implementing any Boolean logic function, and by extension, any arithmetic operation[3].

To implement a parasitic computer using the checksum function we need to design a special message that coerces a target server into performing the desired computation. As a test problem we choose to solve the well known 'satisfiability' (or SAT) problem, which is a common test for many unusual computation methods[4,5]. The SAT problem involves finding a solution to a Boolean equation that satisfies a number of logical clauses. For example, (x_1 XOR x_2) AND ($\sim x_2$ AND x_3) in principle has 2^3 potential solutions, but it is satisfied only by the solution $x_1 = 1$, $x_2 = 0$, and $x_3 = 1$. This is called a 2-SAT problem because each clause, shown in parentheses, involves two variables. The more difficult 3-SAT problem is known to be NP complete, which in practice means that there is no known polynomial-time algorithm which solves it. Indeed, the best known algorithm for an n-variable SAT problem scales exponentially with n, following $(1.33)^n$ (ref. 6). Here we follow a brute-force approach by instructing target computers to evaluate, in a distributed fashion, each of the 2^n potential solutions.

A possible parallel approach to the SAT problem is shown schematically in Fig. 1. The parasite node creates 2^n specially constructed messages designed to evaluate a potential solution. The design of the message, exploiting the TCP checksum, is described in Fig. 3. These messages are sent to many target servers throughout the Internet. Note that while we choose the simpler 2-SAT case to illustrate the principle behind the coding, the method can be extended to code the NP complete 3-SAT problem as well, as explained in the Supplementary Information (see also http://www.nd.edu/~parasite). The message received by a target server contains an IP header, a TCP header, and a candidate solution (values for x_i). The operators in the Boolean equation determine the value of the checksum, which is in the TCP header. The parasite node injects each message into the network at the IP level (Fig. 1), bypassing TCP. After receiving the message, the target server verifies the data integrity of the TCP segment by calculating a TCP

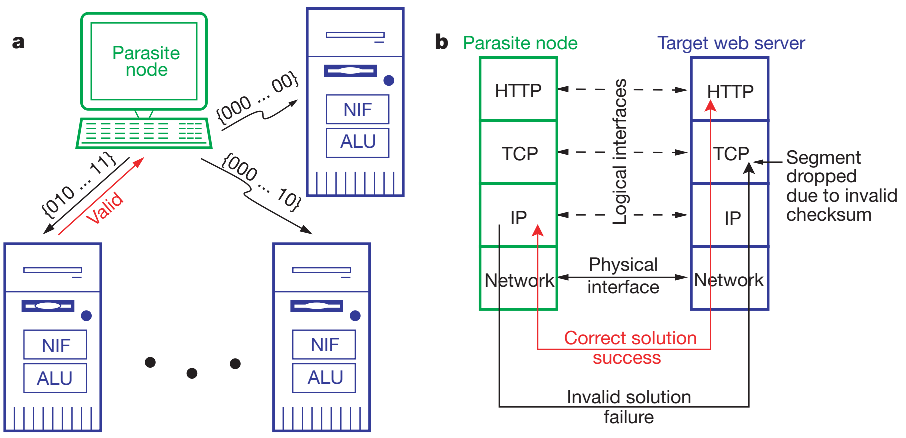
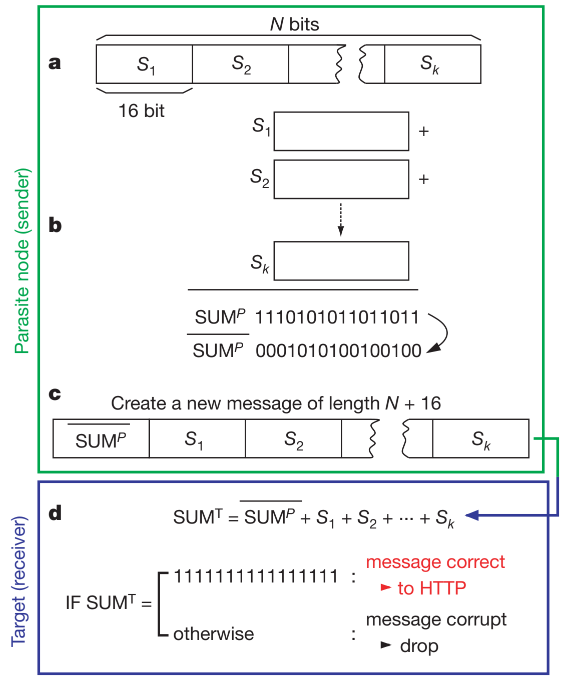


Figure 1 Schematic diagram of our prototype parasitic computer. **a**, A single parasite node (green) coordinates the computations occurring remotely in the Internet protocols. It sends specially constructed messages to some number of targeted nodes (blue boxes), which are web servers consisting of an arithmetic and logic unit (ALU) and a network interface (NIF). **b**, Levels of communication between the parasitic node and the target in our proof-of-principle implementation. The packet carrying the problem to be solved is inserted into the network at the IP level, bypassing the parasitic node's TCP. The construction of the message is such that an invalid solution fails the TCP checksum and is dropped, which is illustrated by the black path labelled 'failure'. Consequently, only valid solutions propagate up through the TCP protocol layer to HTTP, illustrated by the red path labelled 'success'. Targeted web servers respond to all requests that reach HTTP, even invalid HTTP requests. Thus, the parasite node sends out a message for each possible solution (black arrow in **a**), but only receives a response for each solution that is valid (red arrow in **a**).

checksum. The construction of the message (Fig. 3) ensures that the TCP checksum fails for all messages containing an invalid solution to the posed SAT problem. Thus, a message that passes the TCP checksum contains a correct solution. The target server will respond to each message it receives (even if it does not understand the request). As a result, all messages containing invalid solutions are dropped in the TCP layer. Only a message which encodes a valid solution 'reaches' the target server, which sends a response to the 'request' it received.

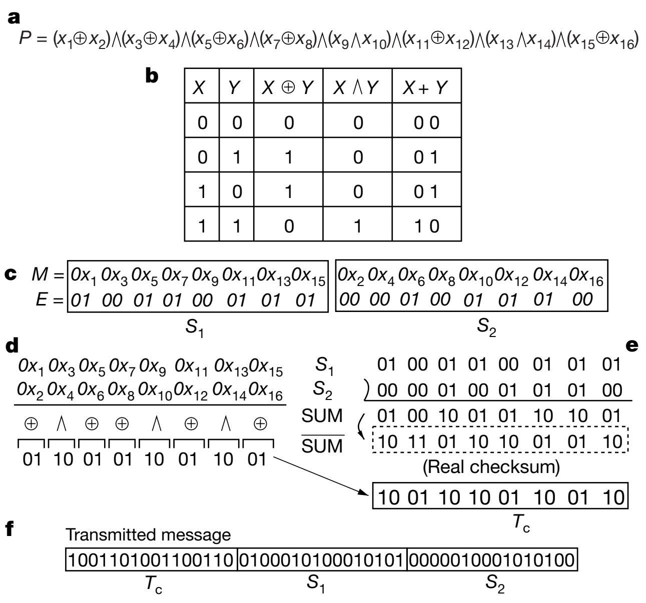
We have implemented the above scheme using as a parasitic node an ordinary desktop machine with TCP/IP networking. The targeted computers are various web servers physically located in North America, Europe, and Asia, each of which unwittingly participated in the experiment. As explained in Fig. 1, our parasite node distributed 2^n messages between the targets. Because only messages containing valid solutions to the SAT problem pass through TCP, the target web server received only valid solutions. This is interpreted as an HTTP request, but it is of course meaningless in this context. As required by HTTP, the target web server sends a response to the parasitic node, indicating that it did not understand the request. The parasite node interprets this response as attesting to the validity of the solution. As expected and by design, incorrect solutions do not generate responses for the web server. A typical message sent by the parasite, and a typical response from a target web server are included in the Supplementary Information.

Our technique does not receive a positive acknowledgement that a solution is invalid because an invalid solution is dropped by TCP. Consequently, there is a possibility of false negatives: cases in which a correct solution is not returned, which can occur for two reasons. First, the IP packet could be dropped, which might be due to data corruption or congestion. Normally TCP provides a reliability mechanism against such events[7], but our current implementation cannot take advantage of this. Second, because this technique exploits the TCP checksum, it circumvents the function the checksum provides. The TCP checksum catches errors that are not caught in the checks provided by the transport layer, such as errors in



Parasite node (sender)
a
N bits
S_1 S_2 S_k
16 bit
b
S_1 +
S_2 +
S_k
SUM^P 1110101011011011
$\overline{SUM^P}$ 0001010100100100
c
Create a new message of length $N + 16$
$\overline{SUM^P}$ S_1 S_2 S_k
Target (receiver)
d
$SUM^T = \overline{SUM^P} + S_1 + S_2 + \cdots + S_k$
IF SUM^T = 1111111111111111 : message correct ► to HTTP
otherwise : message corrupt ► drop

Figure 2 TCP checksum. **a**, The checksum is a simple function performed by all web servers (in TCP), which allows a recipient to check if the received message has been corrupted during transmission. The sender (parasitic node) breaks the message consisting of N bits into 16-bit words, shown as $S_1, S_2, \ldots, S_k$. **b**, The k words are added together using binary one's-complement arithmetic, providing the sum denoted as SUM^P. Next, the sender performs a bit-wise complement on the checksum, so that every bit is flipped: a 0 becomes a 1 and a 1 becomes a 0, obtaining $\overline{SUM^P}$. An example of a checksum and its complement are shown in the figure. **c**, The sender incorporates the complement of the checksum into the header of the message. **d**, The receiving computer (target) again breaks the received message into 16-bit segments and adds them together. The value of the checksum SUM^T calculated by the target is $\overline{SUM^P} + SUM^P$, the first term coming from the header and the second term being the contribution from $S_1 + S_2 + \ldots + S_k$. As SUM^P and $\overline{SUM^P}$ are complementary, the checksum obtained by the receiver has to be 1111111111111111. If any bit along the message has been corrupted during transmission, the checksum obtained by the target will be different from all ones, in which case the target drops the message. A non-corrupted message is passed to the HTTP protocol, which will attempt to interpret its content.



a
$P = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4) \wedge (x_5 \oplus x_6) \wedge (x_7 \oplus x_8) \wedge (x_9 \wedge x_{10}) \wedge (x_{11} \oplus x_{12}) \wedge (x_{13} \wedge x_{14}) \wedge (x_{15} \oplus x_{16})$
b
X Y $X \oplus Y$ $X \wedge Y$ $X + Y$
0 0 0 0 00
0 1 1 0 01
1 0 1 0 01
1 1 0 1 10
c
$M = 0x_1\, 0x_3\, 0x_5\, 0x_7\, 0x_9\, 0x_{11} 0x_{13} 0x_{15}$ $0x_2\, 0x_4\, 0x_6\, 0x_8\, 0x_{10} 0x_{12}\, 0x_{14} 0x_{16}$
$E =$ 01 00 01 01 00 01 01 01 00 00 01 00 01 01 01 00
S_1 S_2
d
$0x_1\, 0x_3\, 0x_5\, 0x_7\, 0x_9\; 0x_{11}\, 0x_{13} 0x_{15}$
$0x_2\, 0x_4\, 0x_6\, 0x_8\, 0x_{10} 0x_{12}\, 0x_{14} 0x_{16}$
$\oplus$ $\wedge$ $\oplus$ $\oplus$ $\wedge$ $\oplus$ $\wedge$ $\oplus$
01 10 01 01 10 01 10 01
e
S_1 01 00 01 01 00 01 01 01
S_2 00 00 01 00 01 01 01 00
SUM 01 00 10 01 01 10 10 01
$\overline{SUM}$ 10 11 01 10 10 01 01 10
(Real checksum)
10 01 10 10 01 10 01 10
T_c
f
Transmitted message
1001101001100110 0100010100010101 0000010001010100
T_c S_1 S_2

Figure 3 Deciding satisfiability using checksum. The 2-SAT problem shown in **a** involves 16 variables $\{x_1, x_2, \ldots, x_{16}\}$ and the operators AND ($\wedge$) and XOR ($\oplus$). **b**, The logical table of XOR, AND and the binary sum (+). In order to get a TRUE answer for P, each clause shown in separate parentheses in **a** needs to be independently TRUE. **c**, To evaluate the value of P we generate a 32-bit message M that contains all 16 variables, each preceded by a zero. As an illustration, we show a possible solution E. TCP groups the bits of the received message in two 16-bit segment and adds them together (Fig. 2a). As shown in **d**, this will result in adding each (x_i, x_{i+1}) pair together where i is odd. The sum can have four outcomes. Comparing the sum with the ($\oplus$) column in the table in **b**, we notice that a TRUE answer for the XOR clause $(x_i \oplus x_{i+1})$ coincides with the (01) result of the (x_i, x_{i+1}) sum. Similarly, if the clause has an AND operator, $(x_i \wedge x_{i+1})$ is true only when the checksum is (10). This implies that for a set of variables $\{x_1, x_2, \ldots, x_{16}\}$ that satisfies P the checksum will be determined by the corresponding operators only (that is, a $\oplus$ should give (01) for the sum check, and for $\wedge$ the sum is (10)). For illustration, in **d** we show the formal lineup of the variables, while in **e** we show an explicit example. The correct complemented checksum for E should be $\overline{SUM} = 10110110100110$. In contrast, the parasitic computer places in the header the transmitted checksum $T_c = 1001101001100110$, which is uniquely determined by the operators in P, as shown in **d**. To turn the package into a parasitic message the parasitic node prepares a package, shown in **f**, preceded by a checksum T_c, and continued by a 32-bit sequence (S_1, S_2), which represent one of the 2^{16} potential solutions. If S_1 and S_2 do not represent the correct solution, then the checksum evaluated by the target TCP will not give the correct sum (111…11). The TCP layer at the target concludes that the message has been corrupted, and drops it. However, if S_1 and S_2 contain the valid solution, the message is sent to HTTP. The web server interprets the solution as an HTTP request; however, because it is not a valid HTTP request, the web server responds with a message saying something like 'page not found' (red arrow in Fig. 1a). Thus, every message to which the parasite node receives a response is a solution to the posed SAT problem (see Fig. 2d).

intermediate routers and the end points[8]. Measurements show that the TCP checksum fails in about 1 in 2^{10} messages[9]. The actual number of TCP checksum failures depends on the communication path, message data, and other factors. To test the reliability of our scheme, we repeatedly sent the correct solution to several host computers located on three continents. The rate of false negatives with our system ranged from 1 in about 100 to less than 1 in 17,000.

The implementation offered above represents only a proof of concept of parasitic computing. As such, our 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, we emphasize that 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.

It is important to note that parasitic computing is conceptually related but philosophically different for cluster computing[10], which links computers such that their cumulative power offers computational environments comparable to the best supercomputers. A prominent example of cluster computing is the SETI program, which has so far enlisted over 2.9 million computers to analyse radio signals in search of extraterrestrial intelligence. In cluster computing, the computer's owner willingly downloads and executes software, which turns his computer into a node of a vast distributed computer. Thus, a crucial requirement of all cluster computing models is the cooperation of the computer's owner. This is also one of its main limitations, as only a tiny fraction of computer owners choose to participate in such computations. In this respect, parasitic computing represents an ethically challenging alternative for cluster computing, as it uses resources without the consent of the computer's owner.

Although parasitic computing does not compromise the security of the target, it could delay the services the target computer normally performs, which would be similar to a denial-of-service attack, disrupting Internet service[11,12]. Thus, parasitic computing raises interesting ethical and legal questions regarding the use of a remote host without consent, challenging us to think about the ownership of resources made available on the Internet. Because parasitic computation exploits basic Internet protocols, it is technically impossible to stop a user from launching it. For example, changing or disrupting the functions that are exploited by parasitic computing would simply eliminate the target's ability to communicate with the rest of the Internet.

In summary, parasitic computing moves computation onto what is logically the communication infrastructure of the Internet, blurring the distinction between computing and communication. We 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 behaviour, drastically altering the current use of the Internet[13,14]. □

Received 21 May; accepted 6 July 2001.

1. Peterson, L. L. & Davie, B. S. *Computer Networks, A Systems Approach* 2nd edn (Morgan Kaufmann, San Francisco, 2000).
2. Garey, M. & Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-completeness* (Freeman, San Francisco, 1979).
3. Boole, G. *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities* (Dover Advanced Mathematics, Macmillan, London, 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 problem. *Science* **278,** 446–449 (1997).
6. Schöning, U. in *Proc. 40th Annu. IEEE Conf. Found. Comp. Sci. (FOCS)* 410–414 (IEEE Comp. Sci., Los Alamitos, California, 1999).
7. Stevens, W. R. *TCP/IP Illustrated* 144–147 (Addison-Wesley, Reading, Massachusetts, 1994).
8. Stone, J. & Partridge, C. in *Proc. ACM SIGCOMM* 309–319 (2000).
9. Stone, J., Greenwald, M., Partridge, C. & Hughes, J. Performance of checksums and CRCs over real data. *IEEE Trans. Networking* **6,** 529–543 (1998).
10. Foster, I. Internet computing and the emerging grid. *Nature Web Matters* at http://www.nature.com/nature/webmasters/grid/grid.html (7 Dec. 2000).
11. Cohen, R., Erez, K., ben-Avraham, D. & Havlin, S. Resilience of the Internet to random breakdowns. *Phys. Rev. Lett.* **85,** 4626–4629 (2000).
12. Cohen, R., Erez, K. ben-Avraham, D. & Havlin, S. Breakdown of the Internet under intentional attack. *Phys. Rev. Lett.* **86,** 3682–3685 (2001).
13. Lawrence, S. & Giles, C. L. Accessibility of information on the web. *Nature* **400,** 107–109 (1999).
14. Lawrence, S. & Giles, C. L. Searching the World Wide Web. *Science* **280,** 98 (1998).

Supplementary information is available on *Nature*'s World-Wide Web site (http://www.nature.com) or as paper copy from the London editorial office of *Nature*.

Acknowledgements

We thank M. Crovella, G. W. Flake, L. Giles, S. Lawrence, D. Peenock and L. Peterson for comments on the manuscript. This work was supported by the NSF.

Correspondence and requests for materials should be addressed to A.-L.B. (e-mail: alb@nd.edu).

Universal behaviour in compressive failure of brittle materials

Carl E. Renshaw* & Erland M. Schulson†

* *Department of Earth Sciences; and † Thayer School of Engineering, Dartmouth College, Hanover, Hampshire 03755, USA*

Brittle failure limits the compressive strength of rock and ice when rapidly loaded under low to moderate confinement. Higher confinement or slower loading results in ductile failure once the brittle–ductile transition is crossed. Brittle failure begins when primary cracks initiate and slide, creating wing cracks at their tips[1–3]. Under little to no confinement, wing cracks extend and link together, splitting the material into slender columns which then fail. Under low to moderate confinement, wing crack growth is restricted and terminal failure is controlled by the localization of damage along a narrow band. Early investigations proposed that localization results from either the linkage of wing cracks[1–3] or the buckling of microcolumns created between adjacent wing cracks[4,5]. Observations of compressive failure in ice[6] suggest a mechanism whereby localization initiates owing to the bending-induced failure of slender microcolumns created between sets of secondary cracks emanating from one side of a primary crack. Here we analyse this mechanism, and show that it leads to a closed-form, quantitative model that depends only on independently measurable mechanical parameters. Our model predictions for both the brittle compressive strength and the brittle–ductile transition are consistent with data from a variety of crystalline materials, offering quantitative evidence for universal processes in brittle failure and for the broad applicability of the model.

Beginning at 20–30% of the terminal failure stress, high-resolution images of brittle compressive failure in ice[6] reveal both wing cracks and secondary cracks emanating from one side of a sliding primary crack (Fig. 1). The secondary cracks create parallel sets of slender microcolumns which are fixed on one end and free on the other. In contrast to previous failure models which have invoked microcolumns as a key element in the failure process[3–5], the