DISTRIBUTED RC5 DECRYPTION AS A CONSUMER FOR IDLE-TIME BROKERAGE

MICHAEL MAY

LRR-TUM, Institute of Computer Science
Technical University of Munich
Michael.May@in.tum.de
http://wwwbode.in.tum.de/~maym

Abstract

RSA Data Security, Inc. is sponsoring contests to prove the (in-)security of miscellaneous cryptographic algorithms as DES or RC5. We have joined RSA's RC5 contest by combining idle CPU power from users visiting a web site. Via an embedded Java RC5 applet down-loading and testing RC5 keys from the web server it was originating from, visitors can participate and donate idle CPU cycles without hassle for the secret key contests. This example of distributed computing with Java applets served as a base to further explore the possibilities Java is offering for the brokerage of idle CPU resources on the Internet. Our new concept – provider orientation, low overhead, working incentives and new, modest applications – solves deficiencies of related full-grown, user-oriented distributed computing environments and gives way to a propagation of an electronic idle-time market focused on the mainstream Internet user as providers of idle computing time.

Keywords: computing time brokerage, java based distributed computing, RC5 decryption, electronic market

1 JAVA BASED DISTRIBUTED COMPUTING

We have written Java applet code to down-load, test and return RC5 key blocks from a Master Servlet located on the Web server. We have further developed a Java based computing system called Locust (LOW cost Computing Utilizing Skimmed idle Time) incorporating concepts quite contrary to current approaches - provider orientation, low overhead, openness, working incentives and 'modest' applications. We will present Locust's concepts more detailed in paragraph Computing Time Brokerage. In contrast to other approaches oriented towards consumers of idle computing time Locust does not need the potential providers to come to a specific Web location in order to donate computing time. Instead we make sure the RC5 applet or any other idle-time collecting application autonomously comes to the potential provider by embedding it in attractive Web pages. By placing the RC5 applet on a frequented web site and joining its visitors into an instant RC5 testing team we have created a sensible distributed consumer application for idle-time delivering encouraging key rates. Combining the web and the key server on the same machine might form a potential bottleneck but is necessary to suffice the restrictive security aspects of the Java Virtual Machine.

The RC5 Secret Key Challenge In 1995 the RSA Data Security Inc. [18] formally announced a new encryption algorithm, the RC5 [15] [6] designed to serve as an alternative to the Data En-
encryption Standard (DES). Proudly claiming RC5's fundamental superiority over DES, RSA Laboratories [18] has challenged a lot of people to quantify its security level. RSA has further heated the efforts by establishing a series of cryptographic contests around RC5 as the RSA Secret-Key Challenge launched in January 1997 with money prizes up to 10,000 Dollars. The challenge was composed by a series of 13 contests to quantify the security offered by DES and RC5 with keys of various sizes and was accepted by several entities. The DES encryption and the RC5 encryption with keys size until 56-bits are already broken. Several efforts are being developed to crack the 64-bit key length.

**Related Approaches**  In contrast to other works on Java based distributed computing we emphasized our approach to be as simple as possible and to make up the simplicity and the performance lack of Java by the sheer number of concurrent applets. We have made yet no further efforts to support extended scalability, load balancing or further communication routing.

Among the more elaborate projects using features of the Java language are Javelin [7], SuperWeb [5] and ATLAS [13]. They all offer lots of additional features - as direct applet to applet communication or special directory services - in order to deliver a full-grown parallel computing environment (so-called infrastructure, framework, toolkit) while considerably adding overhead to the already pending penalties of wide area distributed computing as slow Java byte code interpretation and poor network latency and bandwidth.

Besides, all of them implement distributed computing systems from the client's or customer's point of view paying much less or no attention to the possible providers of such an environment. All the features extending or rather diluting the Java sand-box model serve to create a more powerful parallel programming environment from the programmer's or user's point of view keeping out of mind that it takes several magnitudes more providers than customers if such an idle-time computing framework shall work out and propagate in practice.

**Consumer applications for Idle-time**  The RC5 decryption is an example of a embarrassingly parallel or brute-force application: every possible key has to be tested but, fortunately, is completely independent and can be tested in parallel. Embarrassingly parallel brute-force is the prototypical problem class for every distributed meta-computing scenario and is characterized by (almost) perfect parallelism, high computation to communication ratio and exhaustion of the solution space. These properties are best implemented by using a master/slave computation model. The master splits the main problem into sub-problems and distributes them to the slaves that he keeps track of. Upon completion of their sub-tasks slaves return the sub-result to the master and trigger a new sub-problem being sent. In the RC5 case the RC5 key space is divided into key blocks and distributed to the slaves which check them against the known plain text. Their return value for a completed key block is either Key found or No key found. The Master code we used was implemented by Laurence Vanhelsuwe [19].

**Java Implementation**  One of the features of the RC5 algorithm implementation is its simplicity and independence from hardware acceleration, hence it is straight-forward to formulate the algorithm in C and with a little more effort in an object-oriented language as Java. Due to the lack of mathematical functions support in Java only moderate speeds were achieved by this naive approach with the initialization, setup and encryption methods implemented straight-forward by means of loops. In order to achieve acceptable performance compared to native implementation all loops were unrolled in above methods. Implementation and benchmarking was done by Greg Hewgill.
The benchmarks show that the RC5 applet can achieve – under an efficient implementation or plug-in of the Java virtual machine with just-in-time compilation – up to one fifth of the key rates of native implementations with optimized assembly core. So by just using more than five worker applets any native RC5 application can be recovered and outperformed.

2 COMPUTING TIME BROKERAGE

This example of distributed computing with Java applets will serve as a base to further explore the possibilities Java is offering for the brokerage of idle CPU resources (idle-time) on the Internet. The main drawbacks of current computing time brokerage system we see are their large infra-structural overhead and complete orientation towards the potential customer and user of idle computing time requiring potential providers to actively join the system to donate idle-time.

Outlined in the following we present key features of Locust (LOw cost Computing Utilizing Skimmed idle Time), a brokerage system introducing new aspects we think important for the establishment and management of an electronic idle-time market. In this section we will not differentiate between idle-time brokerage and distributed computing systems as they are both characterized by the trial to combine a multitude of heterogeneous computing nodes into a high performance computer offered to customers via a simple homogeneous interface.

2.1 Related Work

It is idle-time that brokerage and meta-computing efforts are targeting for their systems. Among these are ReGTime [4], a broker system for marketing and using computing power on idle workstations, the cypher breaking meta computer of distributed.net [2] and Damp [19], Jet [11], Javelin [7] and the Web computer white paper [10] – interfaces to meta computing facilities based on Java. They all share common features as infra-structural overhead, customer orientation and volunteering.

Infra-structural overhead

Current web-based distributed computing systems try to add or circumvent features lacking or featuring the Java virtual machine in order to deliver a full-grown parallel computing environment. This considerably adds overhead to the already prevailing drawbacks of wide area distributed computing as poor network latency and bandwidth and slow Java byte code interpretation.

Customer orientation

All related projects are clearly more oriented towards the sparse customers than to the thousands or hundred thousands of potential providers of idle computing time. A parallel programming environment is provided from the programmer’s or user’s point of view neglecting the fact it takes several magnitudes more providers than customers if an idle-time brokerage framework shall work out and propagate in practice.

Volunteering - Lack of Incentives

All systems lack an effective incentive to propagate the acceptance and usage of their systems. Most environments completely lack any motivation at all for providers to donate idle CPU time. They set their hopes in volunteer computing and address their target group in the scientific and computer science field.
ReGTime brings together supply and demand in an idle-time spot market without any pre-information about a possible price – the negotiation is left to provider and customer. So before contacting a ReGTime broker a potential provider or customer has no hint how much idle-time he is going to get or supply and at which price. This situation is far from suited to successfully establish an electronic idle-time market.

A few others as distributed.net feature a kind of abstract motivation as feelings of community or feelings of doing something ‘good’ by joining i.e. the proof of the inefficiency of the DES algorithm for ensuring privacy ([21]). In [16] this feeling is referred to as bayanihan, a "community spirit of unity and cooperation that makes seemingly impossible tasks possible through the concerted effort of many people". These incentives, whose value is hard to quantify, make the success of a idle-time facility subject to good will or luck and – depending on the situation – may work or may not.

In what follows we will give a short overview of the cited related projects by giving more details about the points deferring from above main concepts. ReGTime as the only real idle-time brokerage system focuses the formal and legal aspects of selling and buying idle computing time. In order to enable controlling and accounting of idle-time additional software overhead is granted as necessary i.e. maintainance of separate logins and such. Thus the (architecture dependent) installation of ReGTime is more a matter of hours than minutes.

The Bovine distributed.net effort is a very effective and low overhead meta-computing scenario because it is highly customized and optimized for the decryption of assorted cryptographic algorithms e.g. by use of native binary cypher testing clients and a hierarchy of low latency key servers. It is questionable though if Bovine is able to adopt to and support 'real world' applications in the future without a complete rewrite of their software infrastructure. Jet, coming from Portugal, and Javelin and SuperWeb, both coming from Santa Barbara, California, are more versatile approaches in trying to enhance the limited capabilities the Internet is offering for parallel computing. These enhancements include but are not limited to applet-to-applet communication, fault-tolerance, load balancing by eager scheduling and work stealing, consistent check-pointing, and encryption. Goal of these extensions to the Java virtual machine is broadening the list of possible applications for the distributed computing infrastructure. Dampp and the 'Web computer', the latter being a vague compilation of buzzwords and flashy ideas, are mainly white-paper proposals and exist only as proof-of-concept implementations [19, 8]. The Dampp proposal though features the most basic and promising concept of all Java based distributed computing environments. Therefore we used the Dampp Jobmaster class as a prototypical cornerstone of our own idle-time brokerage scenario.

2.2 New Paradigms

All of the Java based distributed computing environment described above share some common conceptual deficiencies leading to a limited acceptance among potential users and inhibiting the propagation of idle-time brokerage. In the following outline we propose new paradigms – provider orientation, low overhead, working incentives and new embarrassingly modest applications – which heal most of those deficiencies and could give way to establishing an electronic idle-time market. By making joining a distributed computing brokerage system and providing CPU cycles as easy as surfing a Web page with embedded banner ads, idle computing power has the potential to become a marketable Internet good similar to other goods as web space, banner placements and ad clicks further boosting acceptance and pervasion.
Provider oriented  As already mentioned above, current brokerage systems are more oriented towards the sparse customers than to the thousands or hundred thousands of potential providers of idle computing time. All systems require the provider to visit a certain web page – say the meta computer’s home page – in order to donate idle-time, a procedure to be repeated every time the user’s browser is restarted to permanently join the system. Compared to traditional computing environments this admission procedure is refreshingly easy. We think though, it is far from being easy enough.

As already outlined above it takes several magnitudes more providers than customers if an idle-time computing framework shall work out and propagate in practice. In our opinion an initiative addressing hundred thousand or millions of anonymous Internet users – the Internet mainstream – should be as simple and intuitive as possible. It is possible to choose the meta-computing home page as the startup page but which mainstream user is willing to dive into the depths of his browser’s configuration menus? Besides, a computational applet for consumption of idle CPU time will, in the mainstream user's point of view, do nothing. No flashing messages, no moving animations, nothing. But if a surfer isn’t rewarded directly we cannot expect him to click a link. This means we cannot rely on any action involved by the providers to donate idle-time, instead we must make sure the applet does its work even without interaction by making it running automatically. In short: If we cannot make the provider come to the worker applet, we have to bring the applet to the mainstream provider. We will shed more light on this point in paragraph Incentives.

Low overhead - keep it simple  Most of the related work in this field, ambitious and elaborate projects with powerful features in order to deliver a full-grown parallel computing environment, considerably add overhead to the already pending penalties of wide area distributed computing as slow Java byte code interpretation and poor network latency and bandwidth. In our opinion an initiative addressing the Internet mainstream should be embarrassingly simple. The overwhelming success of the Internet, or rather the popular World Wide Web part of it, propagating like a bush fire despite – or rather because – the pitiful simplicity and notorious weakness of its means of implementation, i.e. the Hyper Text Transport Protocol, the Common Gateway Interface and the Hyper Text Mark-up Language, is an example of embarrassing simplicity. It seems the end user, and it is him we are targeting with our approach, only accepts and uses what he is able to grasp mentally. It is questionable if he is going to use a distributed meta-computing environment he does not understand and that sends him encrypted worker applets of unknown origin to do encrypted computations of unknown good and kind.

Openness  Security issues take up a substantial part in the efforts of distributed computing infrastructures to deliver a safe parallel computing environment. These include correctness, privacy and safety of the involved code, data and results.

Correctness regards the integrity of the returned results. Attractive incentives not only move users to provide idle-time but also propel the drive to gain these incentives by fraud. Among the proposed means to verify results returned by untrusted idle-time providers potentially forging results are cross and statistical checking, checksum methods and redundant computations.

Privacy concerns protection of proprietary data against theft and espionage enabling businesses and professionals with sensitive data to use an distributed computing system. Possible solutions include splitting tasks into segments too small to reveal useful information for prying eyes and encryption of data and code.
Safety deals with the integrity of the involved hard and software infrastructure. Malignant code can hamper or crash customer, provider or even the broker applications or machines. Most of these concerns can be cured by use of the Java sand-box model for applets though.

We discourage from any method for enforcement of security issues going beyond the features of the Java Virtual Machine. This strong engagement has a number of reasons.

First distributed computing on the Internet is not (yet) for businesses with sensitive proprietary data. The reason why Internet idle-time computing projects exist is not the Internet’s fundamental qualification for distributed computing but mainly because it is out there and it is very cheap. So idle-time computing, once established, will be a low-cost niche market with a number of inconveniences such as latency, bandwidth, security and time line constraints that will be made up by an unbeatable price-value-ratio. Professionals and businesses seeking for secure parallel job processing should stock up their budget for computing time in an leased or own computing center.

Second special security measures object a number of points imminent to the Internet and its community. The success of the Internet is based on the use of open source (mostly under the GNU Public Licence GPL) and open protocols. Nearly all current Browsers — stemming from NCSA’s open source Mosaic Navigator — are free with Netscape’s source code even available under the GPL. The transportation protocols as HTTP, FTP on top as well as TCP/IP below, openly discussed and passed by means of Requests For Comment (RFC), have even found their way into mainstream desktop operating systems. As there is a clear tendency towards more and more of computing and embedded devices enriching our everyday lives, reacting to our individual needs and wishes but on the other hand also collecting large amounts of sensitive personal data, we think it is crucial not to obfuscate their function by intellectual encryption – possibly hiding extra or “Trojan horse” functionality under the stated purpose – to protect intellectual property of designers and manufacturers. In [17] intellectual encryption and obfuscation, including stripped executables (from object link names), encrypted executables (running in an un-encrypted sliding window) or invisible executables, are denoted as counterproductive to the evolution of (computer) science which is based on natural curiosity.

Incentives - Finding the right price  Most computing environment cited in this paper have no incentive at all to move potential providers to supply idle-time. They depend solely on the commitment or bayanihan of the Internet community to a project with ’good’ purposes i.e. the proof of the inefficiency of the DES algorithm for ensuring privacy ([2]). Others are trying to introduce economical incentives as commodities or micro-currencies. Commodities as proposed by [7] are the means of payment a certain customer of a distributed computing environment chooses to offer his individual providers. A digital library may offer information access, a digital effects company may offer movie tickets and a public computing center may even offer computing time or logins in exchange for providing idle-time. The commodity incentive though, bypasses the market regulating effect of the broker which acts solely as a distribution center of worker applets. Furthermore such commodities may be subjectively experienced as incentives by one and deterrent by the other. Micro-currencies impose another – economical – standard to the quick propagation and acceptance of idle-time brokerage. Before an idle-time market can win recognition a comprehensive global micro-currency market has to be established which is even more difficult barring exchange ratios of constant change.

We propose an two-way approach to the problem of an effective incentive. First we think that the most natural means of payment for an idle-time market is idle-time itself. This may sound absurd, but has lots of advantages on deeper thought. There is no need to determine and establish
an (ever changing) exchange rate between idle-time and money, a rate which would be a constant source of irritation as it may or may not represent the appropriate value for some or most of the users or may lead to speculative buying and selling of idle-time. So when there is no conversion from and to money, providers may be able to spare idle computing time during the night or weekends and consume these accumulated CPU cycles — less a certain amount for the commission — when having increased demand for it.

Second, we propagate idle computing time as a replacement of money or micro-currencies, a tendency that can already be observed in some parts of the commercial Internet. In online advertisement business, banner placement, page impressions and ad clicks are sold by web masters (to finance expensive bandwidth), bought by the industry (to reach their target groups) and — essentially but literally — paid by the average Internet user for the access of an attractive online content, be it access to an online database or up-to-date news (by accumulating page impressions or by buying an advertised good). This phenomenon was recently backed up by an open letter of Clearway, Inc. accompanying the withdrawal of their ad blocking Adscreen software after only ten hours [14]. In this letter, Mark Kriegsman, President of Clearway, recommends to **not** block any ads lest starving out **free** web sites either being forced to shut down or charge memberships for access.

Online advertisement is thus already used as a micro-currency in order to pay for hardly quantifiable online services as visiting a web page, down-loading brand new stock charts or accessing a search engine. None of these services are **free** as it may appear to the surfer. All of them are guarded by a small animated or flashing customs stations invisibly demanding a small toll for accessing the site. We are sure that the mainstream surfer will be accepting worker applets just like he has accepted banner ads on his favorite web sites, worker applets literally doing **nothing** but skimming his idle CPU time. If we make joining a distributed computing brokerage system and providing CPU cycles as easy as surfing a Web page with embedded banner ads, then (micro-) paying for online content with CPU cycles will become common practice.

**Suited parallel Applications** Instead of trying to compensate and/or hide the deficiencies of global distributed computing for conventional massively parallel applications at any price, we should rather think about parallel applications that can live with high latency, small bandwidth, node and network failures and potentially unbound computing time. Well, where is the bargain? The price-value-ratio of idle-time will be unrivaled. If a customer is willing to accept some of the above, well say annoyances, he can have any embarrassingly parallel or brute force job solved at a sales price — provided he has no need for a proof of correctness, no objections against spreading his data all over the Internet and a lot of time. In short, to say it with a computer science proverb: Turn the bug (lack of performance, latency, bandwidth) into a feature. Brute force and highly parallel tasks as raytracing are good examples of suited applications. More of them will show up if cheap idle-time will be available.

3 CONCLUSION

We have used a distributed RC5 testing scenario built around worker applets placed on a highly frequented web site to investigate merits and deficiencies of current Java-based distributed computing and idle-time brokerage. We realized that related approaches suffer from common misconceptions leading to poor acceptance and usage of idle computing time. Our new concepts — provider orientation, low overhead, working incentives and new embarrassingly modest applications — solves those deficiencies and gives way to a fuller propagation of an electronic idle-time market.
We disagree to related works centering around the extension of the Java sand-box model in order to supply a full-grown, user-oriented distributed computing environment for traditional parallel applications. Instead we advocate Locust (L0w cost Computing Utilizing Skimmed idle Time), a embarrassingly simple and light-weight approach focused on the mainstream Internet user as providers of idle computing time. We don’t think there is need for massively-parallel, real-time and high-availability extensions to the notoriously slow and congested Internet. All we need are new embarrassingly parallel – what concerns data dependencies – programming models, embarrassingly modest – what concerns their overall demands on latency and bandwidth – jobs and embarrassingly indulgent – what concerns the user’s time line – customers.

REFERENCES


