Lecture 12

- Common mistakes to avoid.
- Reconciling intractability.

Following are some common mistakes I have encountered in the context of reductions and the class **NP**.

(1) Mistake: equating NP with "hard"

Simply because a problem is in **NP**, it does not necessarily mean that the problem is computationally hard. Folks sometimes mistakenly say, "the problem is in **NP**, therefore it is hard." Another way to say this is: it is ok to assume that **NP** $\neq$ **NP**-complete; it is weird to assume **NP** = **NP**-complete.

Recall that "$q \in$ **NP**" establishes an *upper-bound* computational hardness for the problem q only. The problem q may be really easy, and be $\in$ **NP**. For example, there is a linear-time algorithm for ShortPath, and it is indeed the case that ShortPath $\in$ **NP**. Also consider the following problem: given as input a sorted array $A[1, \ldots, n]$ and an item i, is i in $A[1, \ldots, n]$? Both of the following are true: (i) there is an algorithm that runs in time, and therefore space, $O(\lg n)$ for this problem, and, (ii) this problem $\in$ **NP**. However, under the customary assumption that **L** $\neq$ **NP**, this problem $\notin$ **NP**-complete.

To reiterate, "$\in \mathcal{C}$" expresses an upper-bound hardness for a problem only.

(2) Mistake: NP is "non-polynomial time"

NP stands for "Non-deterministic Polynomial time." Not "non-polynomial time." Whether $\mathbf{P} \neq \mathbf{NP}$, i.e., whether there exist problems for which there are efficient non-deterministic polynomial-time algorithms, but not deterministic polynomial-time algorithms, is of course a famous open problem.

(3) Mistake: a problem is NP-hard implies that no polynomial-time algorithm exists for it

A problem q may be **NP**-hard, and in **NP**. We do not know whether $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$. In the (admittedly unlikely) event that $\mathbf{P} = \mathbf{NP}$, there exists a polynomial-time algorithm for such a q. It is best to not be lazy, and say something like the following fully:

> *The problem is **NP**-hard, and under the customary assumption that $\boldsymbol{P \neq NP}$, no polynomial-time algorithm exists for it.*

(4) Mistake: assuming NP = P ∪ NP-complete

It is a mistake, or at least, weird, to assume that the only two subclasses of **NP** are **P** and **NP**-complete. Rather, it is conjectured widely, under the customary assumption $\mathbf{P} \neq \mathbf{NP}$, that there are problems "between" **P** and **NP**-complete. A well-known example is ISO: given as input a pair of graphs, $\langle G_1, G_2 \rangle$, is G_1 isomorphic to G_2? Recall that we considered SUBGRAPHISO before: whether a subgraph of a given graph is isomorphic to another graph. ISO asks whether the two are isomorphic to one another. ISO is in **NP**, but widely conjectured to not be **NP**-hard. It is conjectured also that ISO is not in **P**.

Another example is PRODTWOPRIMES, which we have discussed earlier: whether, given an input natural number, it is the product of two primes. We know that PRODTWOPRIMES $\in \mathbf{NP} \cap \mathbf{co\text{-}NP}$, and therefore unlikely to be **NP**-complete. It is conjectured that PRODTWOPRIMES is not in **P**.

Indeed, we can think of each of ISO and PRODTWOPRIMES as inducing their own complexity classes which are "between" **P** and **NP**-complete. For example, we could consider ISO-completeness under $\leq_c$. Consider AUTO: given a graph G, does there exist an isomorphism between G and itself that is not the identity mapping? It is possible to show that AUTO $\leq_c$ ISO and ISO $\leq_c$ AUTO. Thus, AUTO is ISO-complete under $\leq_c$.

In summary: assuming that **NP** comprises only **P** and **NP**-complete would be unconventional, and would induce considerable skepticism.

(5) Mistake: assuming that an instance of a problem in NP has only a few certificates, or even one certificate

Consider the following claim.

Claim 83. *Suppose q is a problem in **NP**. Then, an instance of q of size n that is* true *may have $\Omega(2^n)$ certificates.*

Recall that a certificate is some efficiently-sized evidence for the true-ness of the instance. E.g., for an instance of SAT, a certificate is a satsfying assignment. What is the worst-case number of satisfying assignments that an instance of SAT may have? The answer is $\Theta(2^n)$ and therefore $\Omega(2^n)$. Given any n, consider the formula $x_1 \vee x_2 \vee \ldots \vee x_n$. Then, all but one of the assignments to $\langle x_1, \ldots, x_n \rangle$ is satisfying. Thus, we have $2^n - 1 = \Theta(2^n)$ satisfying assumptions.

Thus, there is nothing surprising that there may be exponentially many certificates for an instance of a decision problem, even one that is **NP**-complete.

Following is an example of a piece of work that suggests that we should be surprised by this.

On the Hardness of Topology Inference

[redacted][1] and [redacted][2]

[1] The University of Texas at Austin, USA

[2] The National Science Foundation, USA [redacted]

Abstract. Many systems require information about the topology of networks on the Internet, for purposes like management, efficiency, testing of new protocols and so on. However, ISPs usually do not share the actual topology maps with outsiders; thus, in order to obtain the topology of a network on the Internet, a system must reconstruct it from publicly observable data. The standard method employs traceroute to obtain paths between nodes; next, a topology is generated such that the observed paths occur in the graph. However, traceroute has the problem that some routers refuse to reveal their addresses, and appear as anonymous nodes in traces. Previous research on the problem of topology inference with anonymous nodes has demonstrated that it is at best NP-complete. In this paper, we improve upon this result. In our previous research, we showed that in the special case where nodes may be anonymous in some traces but not in all traces (so all node identifiers are known), there exist trace sets that are generable from multiple topologies. This paper extends our theory of network tracing to the general case (with strictly anonymous nodes), and shows that the problem of computing the network that generated a trace set, given the trace set, has no general solution. The weak version of the problem, which allows an algorithm to output a "small" set of networks- any one of which is the correct one- is also not solvable. Any algorithm guaranteed to output the correct topology outputs at least an exponential number of networks. Our results are surprisingly robust: they hold even when the network is known to have exactly two anonymous nodes, and every node as well as every edge in the network is guaranteed to occur in some trace. On the basis of this result, we suggest that exact reconstruction of network topology requires more powerful tools than traceroute.

1 Introduction

Knowledge of the topology of a network is important for many design decisions. For example, the architecture of an overlay network - how it allocates addresses etc.- may be significantly optimized by knowledge of the distribution and connectivity of the nodes on the underlay network that actually carries the traffic. Several important systems, such as P4P [9] and RMTP [7], utilize information about the topology of the underlay network for optimization as well as management. Furthermore, knowledge of network topology is useful in research; for

M.K. Aguilera et al. (Eds.): ICDCN 2011, LNCS 6522, pp. 251–262, 2011.
© Springer-Verlag Berlin Heidelberg 2011

An excerpt:

> [The problem]... is **NP**-complete. ... We show... [the problem] is in fact much harder than **NP**-complete; it is not just intractable, but unsolvable. ... [possesses] a number of [certificates] which is at least exponential...

Again: there is nothing surprising with the number of certificates being exponential in the size of the instance.

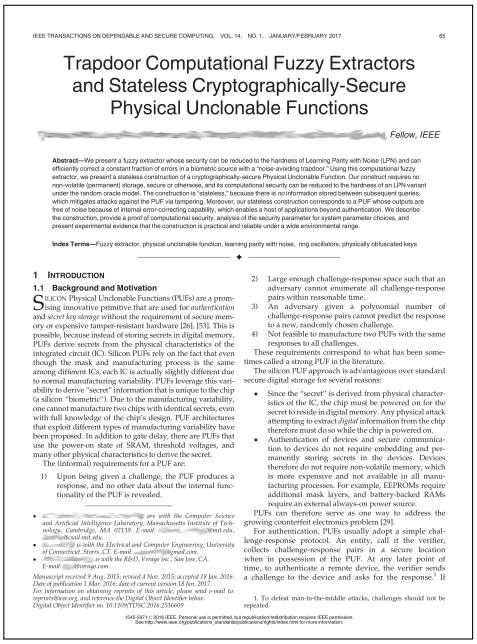
Aside (and not part of the course): instead of a decision problem, which is a function whose co-domain is $\{0, 1\}$, we could instead generalize to a *counting* problem, which maps a set of problem-instances to a non-negative integer which is the number of certificates. E.g., we could map every instance of

SAT or VertexCover to the number of satisfying assignments, or number of different vertex cover sets of size k. It should be easy to observe that if the decision version is **NP**-hard, then such a counting version of the problem is **NP**-hard. There is a complexity class that is associated with these kinds of counting problems. It is written "$\#P$," pronounced "sharp-P."

(6) Mistake: when we reduce problem A to problem B, where B is NP-complete, that implies that A is hard

This mistake is similar to Mistake (1) above, in which we think that $A \in \mathbf{NP}$ means that A is hard. Here, we are considering a reduction $A \leq B$. This is the kind of mistake with which the use of "$\leq$" for a reduction comes in handy to clarify what a reduction actually achieves. A reduction from A to B is written $A \leq B$. And as that indicates, we have shown only that B is an upper-bound for A. That is, we have said nothing about how hard A is. We have said something about how easy A is only.

An example of such a "reduction in the wrong direction" mistake:



IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 14, NO. 1, JANUARY/FEBRUARY 2017 65

Trapdoor Computational Fuzzy Extractors and Stateless Cryptographically-Secure Physical Unclonable Functions

[redacted] Fellow, IEEE

Abstract—We present a fuzzy extractor whose security can be reduced to the hardness of Learning Parity with Noise (LPN) and can efficiently correct a constant fraction of errors in a biometric source with a "noise-avoiding trapdoor." Using this computational fuzzy extractor, we present a stateless construction of a cryptographically-secure Physical Unclonable Function. Our construct requires no non-volatile (permanent) storage, secure or otherwise, and its computational security can be reduced to the hardness of an LPN variant under the random oracle model. The construction is "stateless," because there is no information stored between subsequent queries, which mitigates attacks against the PUF via tampering. Moreover, our stateless construction corresponds to a PUF whose outputs are free of noise because of internal error-correcting capability, which enables a host of applications beyond authentication. We describe the construction, provide a proof of computational security, analysis of the security parameter for system parameter choices, and present experimental evidence that the construction is practical and reliable under a wide environmental range.

Index Terms—Fuzzy extractor, physical unclonable function, learning parity with noise, ring oscillators, physically obfuscated keys

1 Introduction

1.1 Background and Motivation

Silicon Physical Unclonable Functions (PUFs) are a promising innovative primitive that are used for *authentication* and *secret key storage* without the requirement of secure memory or expensive tamper-resistant hardware [26], [53]. This is possible, because instead of storing secrets in digital memory, PUFs derive secrets from the physical characteristics of the integrated circuit (IC). Silicon PUFs rely on the fact that even though the mask and manufacturing process is the same among different ICs, each IC is actually slightly different due to normal manufacturing variability. PUFs leverage this variability to derive "secret" information that is unique to the chip (a silicon "biometric"). Due to the manufacturing variability, one cannot manufacture two chips with identical secrets, even with full knowledge of the chip's design. PUF architectures that exploit different types of manufacturing variability have been proposed. In addition to gate delay, there are PUFs that use the power-on state of SRAM, threshold voltages, and many other physical characteristics to derive the secret.

The (informal) requirements for a PUF are:

1) Upon being given a challenge, the PUF produces a response, and no other data about the internal functionality of the PUF is revealed.
2) Large enough challenge-response space such that an adversary cannot enumerate all challenge-response pairs within reasonable time.
3) An adversary given a polynomial number of challenge-response pairs cannot predict the response to a new, randomly chosen challenge.
4) Not feasible to manufacture two PUFs with the same responses to all challenges.

These requirements correspond to what has been sometimes called a strong PUF in the literature.

The silicon PUF approach is advantageous over standard secure digital storage for several reasons:

- Since the "secret" is derived from physical characteristics of the IC, the chip must be powered on for the secret to reside in digital memory. Any physical attack attempting to extract *digital* information from the chip therefore must do so while the chip is powered on.
- Authentication of devices and secure communication to devices do not require embedding and permanently storing secrets in the devices. Devices therefore do not require non-volatile memory, which is more expensive and not available in all manufacturing processes. For example, EEPROMs require additional mask layers, and battery-backed RAMs require an external always-on power source.

PUFs can therefore serve as one way to address the growing counterfeit electronics problem [29].

For authentication, PUFs usually adopt a simple challenge-response protocol. An entity, call it the verifier, collects challenge-response pairs in a secure location when in possession of the PUF. At any later point of time, to authenticate a remote device, the verifier sends a challenge to the device and asks for the response.[1] If

1. To defeat man-in-the-middle attacks, challenges should not be repeated.

[redacted] are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139. E-mail: [redacted]@mit.edu, [redacted]@csail.mit.edu.
[redacted] is with the Electrical and Computer Engineering, University of Connecticut, Storrs, CT. E-mail: [redacted]@gmail.com.
[redacted] is with the RAD, Verrayt Inc., San Jose, CA. E-mail: [redacted]@verrayt.com.

Manuscript received 8 Aug. 2015; revised 4 Nov. 2015; accepted 18 Jan. 2016. Date of publication 1 Mar. 2016; date of current version 18 Jan. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TDSC.2016.2536609

1545-5971 © 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

The first sentence of the abstract is:

> We present [our approach] whose security can be reduced to the hardness of [some problem that is presumed to be hard]...

Such a reduction would establish nothing about the security of their approach. What they need is a reduction in the other direction: from the hard problem to their approach. Because, presumably what they want is to say that their security is lower-bounded by the difficulty of solving that other problem. Which is a good segue to the next common mistake...

(7) Mistake: if a problem is NP-hard, then every instance of that problem is hard

A notion such as **NP**-hard only captures the *worst-case* hardness of a problem. That is, under the customary assumption $\mathbf{P} \neq \mathbf{NP}$, if a problem is **NP**-hard, then this means that there exists *some instances* of the problem that are hard. Not every instance is necessarily hard. Indeed, most instances of a problem that is **NP**-complete, for example, may be easy, and only very few may be hard. This is an observation that is leveraged by real-world tools such as SAT solvers. It is easy to conceive of entire classes of SAT, all of whose instances are easy. And hardness is not necessarily correlated with the size of an instance. E.g., for SAT, for every instance-size n, one can easily come up with instances of SAT that are easy to decide.

Even for a problem such as Halt which is undecidable, we certainly have classes of instances for which the problem is certainly decidable. Thus, $\mathcal{C}$-hard captures only the worst-case, not every case, and not even the average, or expected, case.

We can indeed conceive, and folks have proposed and used an "average-case" kind of reduction. In such a reduction, the idea is roughly to establish that if we were to pick an instance randomly, then it is hard. But the customary $\leq_k$ which is used to show that a problem is **NP**-hard does not establish anything like that.

(8) Mistake: not addressing the "if" part of the "if and only if" property of a Karp-reduction

Recall that $A \leq_k B$ requires proof of the following property for the underlying mapping, m from instances of A to instances of B: an instance i of A is true if and only if an instance of B is true. A common mistake is to present a mapping that satisfies the "only if" property only, and neglect the "if."

Consider, for example, the following mapping from ISO and SUBGRAPHISO. We know that ISO $\leq_k$ SUBGRAPHISO because ISO $\in$ **NP** and SUBGRAPHISO $\in$ **NP**-hard. Our proposed mapping is: given an instance $\langle G_1, G_2 \rangle$ of ISO, simply output $\langle G_1, G_2 \rangle$ as our instance of SUBGRAPHISO. So we are claiming that the identity mapping works as a Karp-reduction from ISO to SUBGRAPHISO. This is false.

This mapping satisfies the "only if" property. That is, if $\langle G_1, G_2 \rangle \in$ ISO, then $\langle G_1, G_2 \rangle \in$ SUBGRAPHISO. Why? Because G_2 is a subgraph of itself. However, the "if" property is not satisfied. That is, it is possible that G_1 is subgraph isomorphic to G_2, but not isomorphic to G_2. It is easy to come up with an example of this. Thus, the identity mapping does not work as a Karp-reduction from ISO to SUBGRAPHISO.

To reiterate: one must consider both "if" and "only if." Not the latter only.

(9) Mistake: adopting a function that is not computable in polynomial-time as a Karp-reduction

The other important property for the mapping that underlies $A \leq_k B$ is that it must be computable in polynomial-time. A common mistake is to propose a mapping that is not computable in polynomial-time in the worst-case, and then conclude that $A \leq_k B$.

For example, consider reducing VERTEXCOVER to SETCOVER. An instance of VERTEXCOVER is $\langle \langle V, E \rangle, k \rangle$. An instance of SETCOVER is $\langle G, \mathcal{S}, k' \rangle$. Consider the following function, call it m, from the set of instances of VERTEXCOVER to SETCOVER:

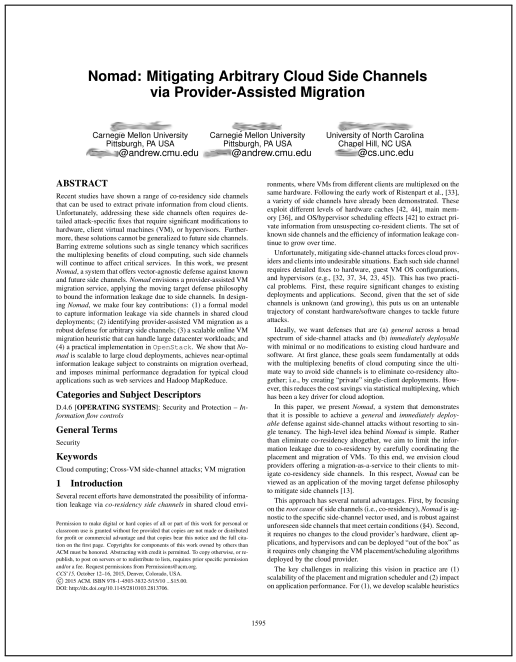
– Corresponding to each edge, $e \in E$, an item $i_e \in G$.

– Corresponding to each subset of vertices of size k, a set $S \in \mathcal{S}$ that is the set of all edges incident on any of those vertices.

– $k' = 1$.

The mindset behind the above function is that we are asking if there there is a set of size k of vertices that touch all the edges. So what we do is simply enumerate each of the sets of that size. The above function m indeed satisfies the "if and only if" property. However, it is not computable in polynomial-time in the worst-case. There exist values of k for which the number of subsets of that size is super-polynomial in the size of the instance.

A couple of negative examples from research: the work to the left (which won a prestigious award) has an exponential-time reduction to Integer Linear Programming (ILP), and the work to the right has an exponential-time reduction to SAT.



Nomad: Mitigating Arbitrary Cloud Side Channels via Provider-Assisted Migration

Carnegie Mellon University
Pittsburgh, PA, USA
[redacted]@andrew.cmu.edu

Carnegie Mellon University
Pittsburgh, PA USA
[redacted]@andrew.cmu.edu

University of North Carolina
Chapel Hill, NC USA
[redacted]@cs.unc.edu

ABSTRACT

Recent studies have shown a range of co-residency side channels that can be used to extract private information from cloud clients. Unfortunately, addressing these side channels often requires detailed attack-specific fixes that require significant modifications to hardware, client virtual machines (VM), or hypervisors. Furthermore, these solutions cannot be generalized to future side channels. Barring extreme solutions such as single tenancy which sacrifices the multiplexing benefits of cloud computing, such side channels will continue to affect critical services. In this work, we present *Nomad*, a system that offers vector-agnostic defense against known and future side channels. *Nomad* envisions a provider-assisted VM migration service, applying the moving target defense philosophy to bound the information leakage due to side channels. In designing *Nomad*, we make four key contributions: (1) a formal model to capture information leakage via side channels in shared cloud deployments; (2) identifying provider-assisted VM migration as a robust defense for arbitrary side channels; (3) a scalable online VM migration heuristic that can handle large datacenter workloads; and (4) a practical implementation in `OpenStack`. We show that *Nomad* is scalable to large cloud deployments, achieves near-optimal information leakage subject to constraints on migration overhead, and imposes minimal performance degradation for typical cloud applications such as web services and Hadoop MapReduce.

Categories and Subject Descriptors

D.4.6 [**OPERATING SYSTEMS**]: Security and Protection – *Information flow controls*

General Terms

Security

Keywords

Cloud computing; Cross-VM side-channel attacks; VM migration

1 Introduction

Several recent efforts have demonstrated the possibility of information leakage via *co-residency side channels* in shared cloud environments, where VMs from different clients are multiplexed on the same hardware. Following the early work of Ristenpart et al., [33], a variety of side channels have already been demonstrated. These exploit different levels of hardware caches [42, 44], main memory [36], and OS/hypervisor scheduling effects [42] to extract private information from unsuspecting co-resident clients. The set of known side channels and the efficiency of information leakage continue to grow over time.

Unfortunately, mitigating side-channel attacks forces cloud providers and clients into undesirable situations. Each such side channel requires detailed fixes to hardware, guest VM OS configurations, and hypervisors (e.g., [32, 37, 34, 23, 45]). This has two practical problems. First, these require significant changes to existing deployments and applications. Second, given that the set of side channels is unknown (and growing), this puts us on an untenable trajectory of constant hardware/software changes to tackle future attacks.

Ideally, we want defenses that are (a) *general* across a broad spectrum of side-channel attacks and (b) *immediately deployable* with minimal or no modifications to existing cloud hardware and software. At first glance, these goals seem fundamentally at odds with the multiplexing benefits of cloud computing since the ultimate way to avoid side channels is to eliminate co-residency altogether; i.e., by creating "private" single-client deployments. However, this reduces the cost savings via statistical multiplexing, which has been a key driver for cloud adoption.

In this paper, we present *Nomad*, a system that demonstrates that it is possible to achieve a *general* and *immediately deployable* defense against side-channel attacks without resorting to single tenancy. The high-level idea behind *Nomad* is simple. Rather than eliminate co-residency altogether, we aim to limit the information leakage due to co-residency by carefully coordinating the placement and migration of VMs. To this end, we envision cloud providers offering a migration-as-a-service to their clients to mitigate co-residency side channels. In this respect, *Nomad* can be viewed as an application of the moving target defense philosophy to mitigate side channels [13].

This approach has several natural advantages. First, by focusing on the *root cause* of side channels (i.e., co-residency), *Nomad* is agnostic to the specific side-channel vector used, and is robust against unforeseen side channels that meet certain conditions (§4). Second, it requires no changes to the cloud provider's hardware, client applications, and hypervisors and can be deployed "out of the box" as it requires only changing the VM placement/scheduling algorithms deployed by the cloud provider.

The key challenges in realizing this vision in practice are (1) scalability of the placement and migration scheduler and (2) impact on application performance. For (1), we develop scalable heuristics

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
CCS'15, October 12–16, 2015, Denver, Colorado, USA.
© 2015 ACM. ISBN 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813706

2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications

Time-triggered Program Self-monitoring

School of Computer Science
University of Waterloo
200 University Avenue West
Waterloo N2L 3G1, Canada
Email: borzoo@cs.uwaterloo.ca

Dept. of Elec. and Comp. Eng.
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, Canada
Email: j22thoma@uwaterloo.ca

Dept. of Elec. and Comp. Eng.
University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, Canada
Email: sfischme@uwaterloo.ca

***Abstract*—*Runtime monitoring* aims at analyzing the well-being of a system at run time in order to detect errors and steer the system towards a healthy behavior. Such monitoring is a complementary technique to other approaches for ensuring correctness, such as formal verification and testing. In *time-triggered* runtime monitoring, a monitor runs as a separate process in parallel with an application program under scrutiny and samples the program's state periodically to evaluate a set of properties. Applying this technique in a computing system results in obtaining bounded and predictable overhead. Gaining such characteristics for overhead is highly desirable for designing and engineering time-critical applications, such as safety-critical embedded systems. However, a time-triggered monitor requires certain synchronization features at operating system level and may suffer from various concurrency and synchronization dependencies and overheads as well as possible unreliability of synchronization primitives in a real-time setting.**

In this paper, we propose a new method, where the program under inspection is instrumented, so that it *self-samples* its state in a periodic fashion without requiring assistance from an external monitor or internal timer. We call this technique time-triggered *self-monitoring*. First, we formulate an optimization problem for minimizing the number of points in a program, where self-sampling instrumentation instructions must be inserted. We show that this problem is NP-complete. Consequently, we propose a SAT-based solution and a heuristic to cope with the exponential complexity. Our experimental results show that a time-triggered self-monitored program performs significantly better than the same program monitored by an external time-triggered monitor.

***Keywords*–Runtime verification; monitoring; time-triggered; instrumentation; predictability; self-monitoring; real-time; embedded systems**

I. Introduction

In computing systems, *correctness* refers to the assertion that a system satisfies its specification. *Verification* is a technique for checking such an assertion. *Runtime verification* [1]–[6] refers to a lightweight technique, where a *monitor* checks at run time whether or not the execution of a system under inspection satisfies a given correctness property. Runtime verification complements exhaustive verification methods, such as model checking and theorem proving, as well as incomplete solutions, such as testing and debugging. This is because exhaustive verification often requires developing a rigorous abstract model of the system and suffers from the state-explosion problem. Testing and debugging, on the other hand, provide us with under-approximated confidence about the correctness of a system, as these methods only check for the presence of defects under specific scenarios.

Most monitoring approaches in runtime verification are *event-triggered*. In these approaches, the occurrence of new events (e.g., change of value of a variable) triggers the monitor. This constant invocation of the monitor leads to *unpredictable overhead* and potentially *bursts* of monitoring intervention at run time. These defects can cause serious issues especially in real-time embedded safety/mission-critical systems. To tackle these drawbacks, in [7], [8], the authors propose *time-triggered* runtime verification and execution monitoring, where a time-triggered monitor runs in parallel with the program and samples the program state periodically to evaluate a set of properties. Employing such a monitor results in observing bounded and predictable overhead at runtime, which are critical design parameters for a designer of embedded time-sensitive systems.

Although time-triggered monitoring results in obtaining a monitor with predictable overhead and probe effects, it introduces certain complexities as well. For example, the monitor needs to run as a separate process or thread. The first drawback of such a structure is the high cost of context switching and synchronization. Moreover, synchronization data structures require the underlying operating system to provide kernel-level system call primitives and inter-process communication features. In fact, some of the widely used embedded environments (e.g., TinyOS) lack such multi-tasking features. Moreover, if the program under scrutiny is blocked (e.g., for I/O), the monitor continues trying to sample the program periodically. This will waste system resources. Furthermore, a monitor process coupled with a program creates a tight dependency between them at run time. For instance, if the monitor crashes while evaluating properties, it may never resume the program's normal operation. Finally, since the monitor cannot directly read the state of the program, it will keep taking samples from the program even if no new events have occured between two samples.

978-0-7695-4824-1/12 $26.00 © 2012 IEEE
DOI 10.1109/RTCSA.2012.16

IEEE computer society

(10) Mistake: equating NP-complete and NP-hard

NP-hard is a lower-bound on computational hardness only. **NP**-complete is an upper- and lower-bound. There are problems that are **NP**-hard that are not **NP**-complete. An example is HALT. Following is another example that is more nuanced.

Let VERTEXCOVEREXACT be the following decision problem: given as input an undirected graph G and an integer k, does G have a vertex cover of size k and no smaller?

VERTEXCOVEREXACT is indeed **NP**-hard. However, it is unlikely to be in **NP**, and is therefore unlikely to be **NP**-complete. Why? Think about whether a `true` instance of VERTEXCOVEREXACT possesses an efficiently-sized certificate that can be verified efficiently. The "no smaller" part is the tricky part. More broadly, there are problems for which algorithms exist, which are "harder" than **NP**.

Note, however, that VERTEXCOVEREXACT $\leq_c$ VERTEXCOVER. Because, given a polynomial-time algorithm for VERTEXCOVER, we can do a binary search on k to identify the smallest size of a vertex cover for G.

Aside (not part of the course): we can prove that VERTEXCOVEREXACT is complete for a particular class in the *polynomial hierarchy*. Also, we can prove that VERTEXCOVEREXACT $\in$ **PSPACE**.

(11) Mistake: equating NP and co-NP

co-NP comprises problems that are complements of problems in **NP**. It is unlikely that **NP** = **co-NP**, though it is indeed true that **NP** $\cap$ **co-NP** $\neq \emptyset$. As a practical matter, however, one may argue that we do not need to carefully distinguish **NP** and **co-NP**, because an oracle for a problem q is also an oracle for its complement, $\overline{q}$, except with the output negated.

Dealing with intractable problems in practice

Given the customary assumption $\mathbf{P} \neq \mathbf{NP}$, we know that no efficient algorithm exists for a problem that is **NP**-hard. Indeed, even if a problem is, for example, Iso-hard, it is unlikely that an efficient algorithm exists for it. However, such problems arise quite frequently in practice, in every imaginable context in computing. For example, it was recognized in the early days of cloud computing that placing Virtual Machines (VMs) in servers is a kind of multi-dimensional bin packing problem, which is **NP**-complete.

So what is one to do? In the following, we consider several possible approaches when confronted with a situation in which one is unable to conceive of an efficient algorithm.

Efficient approximation

We have discussed some approximation algorithms that are efficient for problems whose decision versions are **NP**-complete; for example, those for TSP and Vertex Cover. There is a rich theory that underlies efficient approximation, and a reduction all of its own called a "gap-preserving reduction," which helps establish a lower-bound for the best approximation ratio we can achieve. For example, we can prove, via a gap-preserving reduction, that $\lg n$ is the best approximation ratio we can achieve with a polynomial-time algorithm for Set Cover under the assumption that $\mathbf{P} \neq \mathbf{NP}$. It is possible to sub-classify the class **NP** based on how approximable a problem is.

The monograph of Arora and Lund is a great starting point: `https://www.cs.princeton.edu/~arora/pubs/chapter.ps`. You should find that document accessible now that you have taken this course.

Randomized algorithms

Even though a problem is not in **P**, it is possible that it is not **NP**-hard. For example, it may be in the class **RP**: the class of decision problems for which there exists an *randomized* algorithm. For example, for a long time it was not known whether the problem of determining whether an integer is

prime is in **P**. It turns out that it is indeed in **P**, a discovery that was made in the mid-2000's. However, by then, we already knew that it is in **RP**, and the randomized Miller-Rabin primality test continues to be the most widely used algorithm, rather than the polynomial-time algorithm.

Thus, it may be worthwhile investigating whether a problem is in **RP**, if one is unable to devise a polynomial-time algorithm for it. However, it is conjectured that $\mathbf{RP} \subset \mathbf{NP}$, and therefore if you prove that your problem is indeed **NP**-hard, an efficient randomized algorithm is unlikely to exist.

Reduction to a problem such as SAT and model checking, and use of a constraint solver

If one is unable to conceive of an efficient, or really, any algorithm, for one's problem, then a possibility is to reduce a decision version to which the problem is polynomially related to the language of an off-the-shelf constraint solver. The right way to do this is to: (i) first identify a complexity class to which the problem belongs, and, (ii) carry out a Karp reduction to the problem the solver understands.

An example of such a solver is a SAT solver, which customarily takes as input an instance of CnfSAT, that is, a SAT formula in CNF. In my experience, reduction first to Circuit-SAT, and then employment of the "textbook" reduction to CnfSAT is usually the most conceptually simple way to go about this. For example, consider VertexCover, which is the following decision problem: given as inputs (i) a graph G, and, (ii) a natural number k, does there exist a subset of vertices of G of size k that is a vertex cover for G?

A reduction to Circuit-SAT could be the following. Construct a circuit as follows.

– Take as input one wire per vertex.

– For each edge, $\langle u, v \rangle$, join the input wire that represents u and v with an OR gate. This captures the constaint that at least one of those must be in the vertex cover.

– Add a sub-circuit that performs an integer-sum of the input wires to the circuit, and compare the output of that sub-circuit to a circuit encoding of the integer k.

– Do an AND of the wires that are the outputs of each 'edge' sub-circuit, and the comparison-with-k sub-circuit. This is the output wire of the circuit.

The size of the above circuit is $O(n \lg n)$, where n is the size of the input graph. From this instance of CIRCUIT-SAT, we can generate an instance of CNFSAT in linear-time.

If our problem is not in **NP**, it may be, for example, in **PSPACE**. In this case, we could perform a reduction to model-checking, and deploy an off-the-shelf model-checker such as nuSMV. The proof that a problem is in **PSPACE** can be constructive: we prove it by producing an algorithm that takes auxiliary space only polynomial in the size of the input.

Via assumptions about the input, and heuristics

A final way we mention is somewhat "hacky." And that is to simply make assumptions about the input so that we simply ignore hard instances. Recall that even if a problem is **NP**-hard, it is possible that only relatively few classes of input instances may be hard in practice. This is a feature that is exploited by solvers such as SAT solvers and model checkers. "Branch and bound" and "gradient descent" are algorithm design strategies that work as heuristics in this context. Thus, if it turns out that we can assume that hard instances do not arise in practice, even though the worst-case characterization of our problem may be that it is hard, we may have an efficient algorithm for large classes of inputs that are not hard.

A tricky part with this is that if a problem is indeed **NP**-hard, and we have an efficient algorithm driven, for example, by heuristics, then we may not get a strong sense for what classes of inputs our algorithm does not perform well. The reason is that if we do, then that would give us insight in to the separation of **P** from **NP**, which would be a major contribution to computing.