9-3-2013

Modeling and control of a dynamic information flow tracking system

Maria Khater

Maria Khater
Candidate

Electrical and Computer Engineering
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Professor Rafael Fierro , Chairperson

Professor Jedidah Crandall

Professor Chaouki Abdallah

Modeling and Control of A Dynamic Information Flow Tracking System

by

Maria Khater

B.E., Computer and Communication Engineering,
American University of Beirut, 2011

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science
Electrical Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2013

Dedication

To my parents, Elias and Hiam, for their continuous support, encouragement and patience.

To my brother, Jad, for his motivation, laughter and love he surrounded me with, from miles away.

To the memory of Khalo Bassam and Jeddo Toufik, for all the precious moments I couldn't spend with you.

Acknowledgments

I would like to thank my advisor, Professor Rafael Fierro, for giving me a great opportunity to work on this topic. I can't be but extremely thankful for all his suggestions, patience and guidance he provided, in which this thesis would have never been accomplished without it. I would also like to thank my co-advisor, Professor Jedidiah Crandall, for his guidance and help throughout the two years I have spent working under his supervision.

My thanks goes to Professor Chaouki Abdallah for serving on my defense committee and reading my thesis, not to forget his help and guidance since 2010 up till now.

I would like to thank all members of MARHES Lab for their friendship and kindness. I also want to thank my colleague Antonio Espinoza, who is a PhD student with my co-advisor, for his invaluable help and support in working on the thesis.

I would like to extend my appreciation to my friends in Albuquerque who were a second family to me, for the great source of happiness in the past two years. I can never forget to thank all my friends back in Lebanon who provided support from miles away. I am truly appreciate your presence in my life, Firas Ayoub and Sarah Abdel Massih.

At last, words fall short of expressing my thanks to my parents and brother for being extremely supportive. I would have never been able to do this without you. Being away from you has never been easy. Thank you for being a great source of joy in my life.

This work was support by NSF grant #: 1017602.

Modeling and Control of A Dynamic Information Flow Tracking System

by

Maria Khater

B.E., Computer and Communication Engineering,
American University of Beirut, 2011

M.S., Electrical Engineering, University of New Mexico, 2013

Abstract

This thesis introduces and details the effort of modeling and control design of an information tracking system for computer security purposes. It is called Dynamic Information Flow Tracking (DIFT) system. The DIFT system is developed at the Computer Science Department at the University of New Mexico, works by tagging data and tracking it to measure the information flow throughout the system. DIFT can be used for several security applications such as securing sensor networks and honeypot – which is a trap set to detect, deflect, or counteract attempts at unauthorized use of information systems. Existing DIFT systems cannot track address and control dependencies, therefore, their applicability is currently very limited because important information flow dependencies are not tracked for stability reasons. A new approach is taken, aimed at stabilizing DIFT systems and enabling it to detect control dependencies at the assembly-level, through control theory.

Modern control has been used to model several cyber-physical, computing, networking, economical... systems. In an effort to model a computing system using

control theory, this thesis introduces a general hybrid systems framework to model the flow of information in DIFT when control dependencies are encountered. Information flow in DIFT is represented by a numeric vector called "taint vector". The model suggested benefits from the characteristics of hybrid systems and its ability to represent continuous variables and discrete events occurring. The system is stabilized by making sure that the taint vectors represent the true information flow in control dependencies. This problem is solved by designing a PID and model predictive controller which guarantee that system does not over taint, while allowing information to flow properly. The modeling framework is validated by comparing simulations of the hybrid models against. This research provides a new approach to solve the DIFT over-tainting problems through modeling it as a hybrid system and forcing the constraints to be obeyed by the taint values.

Contents

List of Figures

List of Tables

Glossary

$\mathbb{R}$	Taylor series coefficients, where $l, m = \{0..2\}$
$A_{\mathbf{p}}$	Complex-valued scalar denoting the amplitude and phase.
A^T	Transpose of some relativity matrix.

Chapter 1

Introduction

1.1 Overview

This document introduces a model for a computing system which tracks the information flow in a program at the assembly level. Dynamic Information Flow Tracking system (DIFT) that is being studied in this thesis for the purpose of modeling through control theory. Several control theorists are attempting to model complex systems, computing systems, queuing systems, smart grids and more... Modern control is now applied to several physical systems for instance, power grids, internet grid and cloud computing systems in which control theory addresses the problems in such systems, models it and provides solutions.

The authors, in [11], provide a starting point to model and control design problems of computing systems via control theory. The authors in [11] provided several examples of modeling computing systems such as HTTP Apache server, queuing systems, load balancing and random early detection of router overloads. After defining the problem in these systems, the authors suggest a model for each of the above systems which represents the dynamics of the continuous and/or discrete variables

in the systems.

Several other computing systems and networking protocols are modeled through control theory, for instance the TCP congestion model. In [15], the authors have used the hybrid systems in order to model the traffic flows of a communication network. The window size, TCP queue for congestion control, timeouts and packet drops that have discrete events and continuous dynamics, are all modeled as hybrid system. The hybrid model starts by representing the path taken by the TCP packets by a set of nodes representing the routers and links connecting them. The model switches from one state (timeout, drop, full queue...) to another upon any discrete event occurs. The hybrid model can reproduce the packet-level simulation accurately. Such models are beneficial for analyzing and designing protocols, more details are provided in Chapter 2.

This document discusses the modeling of a DIFT system which tracks the flow of information through representing the transfer of the data between the input to the different registers of a computer architecture and its memory [2, 5, 19, 28]. The input data to a specific program propagates through the registers and the memory of the computer uniquely that the DIFT system tracks it and stores it. The information flow in any program is also affected if it was provided with different input. The DIFT system can be used for security purposes in detecting an attack through the input by monitoring the information flow of the program. As any other system, the DIFT system has several problems which may cause a mis-representation of the flow of information.

The data transfer is tracked through a taint decimal vector of length N. It represents the data transfer from the input through registers and the memory of the computer. The flow of data is recorded through these taint vectors. A more detailed description of the DIFT system is provided in Chapter 3. In some cases the taint can overflow and saturate, causing the system to give an overestimation of the data

transfer. This phenomenon is called over-tainting, resulting with taint vectors which are not representative of the true information flow in the DIFT system.

The taint vectors in a DIFT system vary dynamically according to the program that is being executed. Each program has a different information flow in the computer architecture, resulting with different taint vectors *i.e.*, different information flow. Modeling a DIFT system in terms of mathematical models –which are translated to differential equations and state-space descriptions, is crucial for analyzing the behavior of the taint in different programs and monitoring the over-taint incidents that happens. This document proposes two models for the DIFT system that reproduces the behavior of the taint dynamics given by a specific program.

The taint dynamics are governed by the instructions of the program thus the different dependencies of the instructions of the program causes the taint to vary. The dependencies between all of the instructions are hard to identify and isolate, after executing a large program the taint vector be the result of all of these dependencies. The dynamics of the taint is analyzed when over-tainting occurs the behavior of the DIFT system is similar to existing systems which have been addressed in control theory.

1.2 Motivation

Dynamic Information Flow Tracking (DIFT) is a technique where tags are assigned to data upon input to a system, then the tags follow the data as the system computes to learn something about the flow of information within the system. The problems DIFT systems face have been the subject of much classical and recent research in the field of computer security. However DIFT systems' problems are similar to many real world problems that control theory is ideal for solving. For example, in aerial vehicle designs, a technique called "relaxed static stability" (RSS) allows the

design of aircraft with advanced maneuverability and stealth capabilities by relaxing the requirement of designing the aircraft to be asymptotically stable during flight [11]. RSS is used for more maneuverability during the flight, it provides a tradeoff between controllability and maneuverability. RSS favors one feature over the other when designing the controllers of fighter jets and those of commercial airplanes. The extra maneuverability is used in fighter jets to achieve a high angle of attack and more aggressive motion, whereas commercial airplanes use the extra controllability to achieve stability. The control theory involved in our DIFT system is inspired by this concept of maneuverability and controllability in jets.

Control theory has been previously used to model several computing systems such as the traffic on a server and the resource allocation for each user [17, 18, 27]. Such models and control techniques may also apply for other applications and in other communities. For instance, Relaxed Stability has been used in aircraft designs to take advantage of the open loop characteristics of the plant in order to perform extreme moves during flight. The concept of relaxed stability can be applied to DIFT. For instance, the information flow in a program is captured in the taint vectors but in several cases where overtainting occurs, it might be possible that the taint vectors would saturate. Consequently, the information can not be tracked through the program. The system needs to be controlled but if the DIFT system is extensively controlled then the taint vectors will not demonstrate the information flow. In conclusion, relaxing some constraints results in a better tracking of information in a program.

The model of the system will be given as a hybrid model, this will be justified in this document. Hybrid systems are one of the most examined topics in control theory and such systems have been proven to form a union of discrete and continuous systems [8]. Moreover, hybrid systems are becoming one of the most researched topics dealing with state flow systems [3]. In [25], the incident of Ariane 5 in June of

1996, which crashed 37 seconds after launching, was due to a failure of coordination between continuous and discrete systems. However, hybrid modeling provides a better representation for real-time systems, which never tend to exhibit a purely discrete nor a purely continuous model.

DIFT systems are designed to track information as it flows through the computer by tainting inputs of interest and propagating the taint through the system as the program is executed. DIFT has many applications, such as malware analysis, intrusion detection, and security models concerned with confidentiality, integrity, and availability. However, currently there are no working DIFT systems that correctly deal with certain dependencies resulting in over-tainting of the system. Because of this, current DIFT systems are used for a limited range of real world applications.

The DIFT system has been studied by computer scientist, for several purposes. It can be used for significant security and information analysis in critical systems. Securing sensor networks before they are deployed is critical. Today, these systems are being deployed in military and medical applications, yet are still built on top of architectures with simple memory models using the C language, or variants of C [17]. Memory corruption vulnerabilities can lead to worm attacks [18] and other remote intrusions that allow adversaries to completely take control of the network. At the same time, compared to general purpose computers, sensor devices have design constraints that place even more restrictions on security mechanisms designed to thwart these attacks.

1.3 Contribution

This thesis introduces a novel approach in solving the over-tainting in a DIFT system from the computer science community perspective. The problem of over-tainting is approached in as a control problem. The DIFT system is modeled from the point of

view of control theory.

This approach is quite challenging since there almost no literature review and previous work in modeling a computer based dynamic system. The DIFT system is built by the Computer Science Department at University of New Mexico. The model provided in this thesis specifically represent the DIFT system which was developed by my co-advisor Professor Jedidihal Crandall. The DIFT system is introduced in Chapter 3 for the purpose of modeling and designing a controller.

Moreover, the work done in terms of modeling computing systems through control theory is limited to couple of computing systems such as servers, scheduling of tasks in a CPU, queues and graph models of networked systems. All of those systems are dealt with on a high level and generally was assumed in most models. It was beneficial to study the models which are designed for computing systems, since this thesis deals with one. However, there are few material which consider DIFT as a control problem.

Several challenges where encountered in the modeling process of the DIFT systems. This document presents the preliminary models and the incremental changes done to come up with the final one. Knowing the modeled computing systems in control theory, the models was refined in order to be more reflective of the DIFT system's dynamics. The main contribution of this thesis is introducing a new model of a computing system, via control theory.

1.4 Organization of the Thesis

The remainder of the document is organized as follows. After the introduction, the Literature Review of the useful research regarding computing system and control theory are presented in Chapter 2 to illustrate the previous computing systems which

was modeled in control theory. It also introduces the model which was presented to a different DIFT system. Chapter 3 defines the DIFT system which being studied. Chapter 4 contains the preliminary mathematical models with the incremental changes done and Chapter 5 discusses the final model and presents the results of the simulations and the verification of the model. Chapter 6 provides an MPC controller for the modeled DIFT system. In the last chapter, conclusions and future work are stated.

Chapter 2

Literature Review

Modeling computing systems through control theory has been addressed earlier by researchers. This thesis discusses an information flow tracking system. Its significant is to provide a brief overview of the problems addressed before since it helped in modeling the DIFT system. Some system models are built for the purpose of replicating the output of the original system since they may require computational resources to model the data of the real system. Other system models are used for addressing the problems in the system and provide a solution through feedback. The basic overview on this topic is given in the Feedback in Computing systems book [11].

The book provided guidelines and examples of basic modeling of computing systems. For instance, they have modeled the CPU utilization in a server, by constructing an empirical model for the plant by collecting real time data. The administrator has a certain constraints on the percentage of utilization of the CPU related to the maximum number of clients on the server. The plant is represented in a first order Linear Time Invariant (LTI) discrete time transfer function, where the overall objective is to regulate the utilization to a predefined reference utilization with a

short, accurate and stable convergence. The feedback loop is closed and an integral controller is added to achieve the desired results.

The book also demonstrates how linear programming can be used by applying Linear Quadratic Regulator(LQR) to more complex systems, such as the workload management in data centers and tandem queues. The dynamic controller is designed based on a set of constraints which are defined by the system being studied resulting with different Q and R matrices. The book defines the main principles of control theory and provides a good start to get introduced to the area of using feedback control in computing systems. There are more efforts to model and control several computing systems such as the Transmission Control Protocol (TCP) network as mentioned in Chapter 1.

The model for the data transfer in the networking system is modeled through graphs in which the dynamics of the routes, queues and discrete events - timeouts, drops or congestion, are modeled through a set of difference equations. The difference equations model the packet transmission along a predefined path in the network. There are several scenarios in which the packet can be transmitted from the sender to the receiver. Modeling the system from the perspective of control theory as a hybrid system, yielded with more realistic representation of the system. One of the improvements of this model is that the Round Trip Time (RTT) is dynamic, it is updated with the state of the system, older models have assumed a constant RTT by averaging the value in a fluid-like model for the same system.

In this paper [15], the authors modeled the end-to-end TCP connection flow by graph of nodes and unidirectional links, where weights are set to define the bandwidth in the links and the delays for the nodes. The network is modeled in a dynamic way where different paths can be taken from the end-to-end level. On the routers level or in other words on the node level, the behavior of the packet has four possible states which are related to the state of the queues on the nodes or routers. The queues can

be empty so all the packets coming in are transmitted, not empty but not full, full with additional incoming packet or timeouts can occur where packets are lost.

The paper introduced a reliable model for a network system using hybrid systems benefiting from the continuous and discrete in the system. The variables of the model are updated based on the system's state for instance, the general form of the queue's dynamics is

$$\dot{q}_f^l = s_f^l - d_f^l - r_f^l, \tag{2.1}$$

where l and f denote the link and the flow, q_f^l the number of bytes in the queue, s_f^l rate of arrival of the packets to the queue, d_f^l rate of packet drop, r_f^l rate of transmission of packets. The above values are updated accordingly based on the state of the system. After simulating the end-to-end model, it is shown that the results of the model is similar to the actual data grabbed from the network system. This paper provides insight about the significance of using hybrid models in modeling networked and computing systems. In addition, the hybrid model 2.1 for the packet flow in the queues in the routers was beneficial for the model presented later for the DIFT system.

For developing the hybrid model for the DIFT system, it was essential to look at the different approaches to develop and control thermostat model since the DIFT system share common aspects with the ON/OFF modes in the thermostat hybrid model. The paper [10] has examined using the LQR approach in order to track the reference input of the cooling thermostat. The model developed is for the aggregated power response of a homogeneous population of thermostatically controlled loads then the results have been analyzed under the influence of disturbance and noise. Similar papers was read in order to be more familiar with applying LQR on hybrid systems.

One of the interesting papers which inspired the design of the controller of the

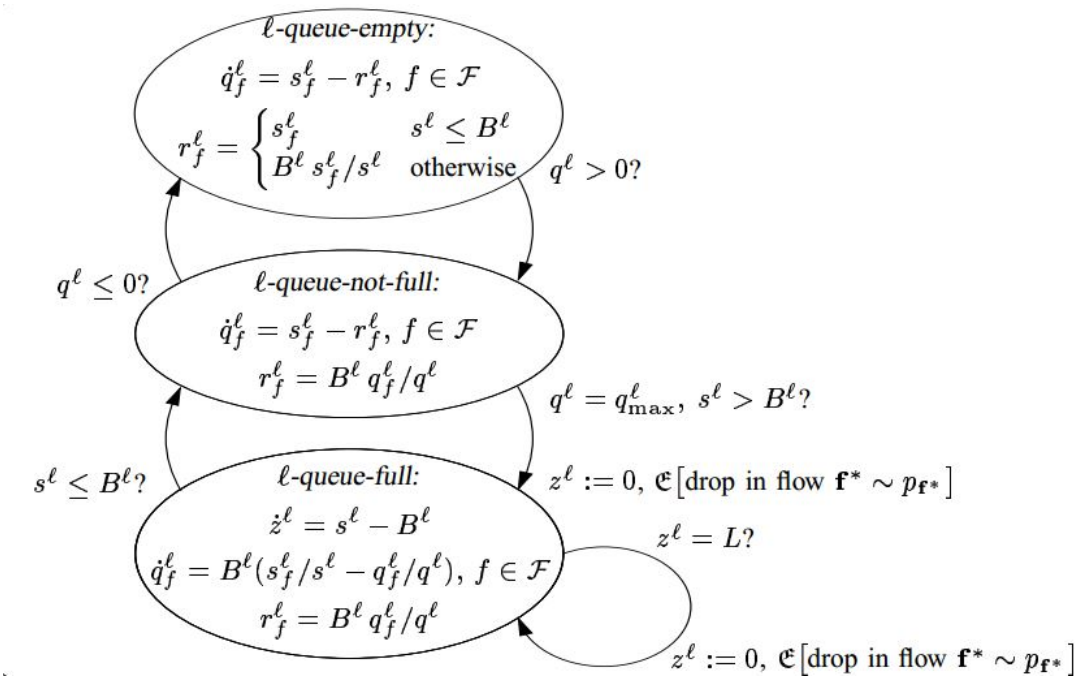


Figure 2.1: Hybrid model of queue in TCP.

DIFT system is [8, 30]. A hybrid controller is developed for doing an aggressive flip in a quadrotor through controlling the velocities of the motors [8]. Quadrotors are inherently unstable systems, however, there are numerous number of controllers which are implemented on board and off board. Quadrotors can't hover at a point in space without a tracking system, on board, off board controllers and height. All these controllers are programmed to maintain the position of the quadrotor, without them the quadrotor would be hard to control manually. Similarly, the DIFT system is inherently unstable, it tracks and monitors the ow of information through the CPU, the system will saturate with tagged data in an finite time. The DIFT system is compared to the quadrotor's flips done in [8]. It uses a hybrid controller which limits the speed of the motors to ensure a safe recovery of the maneuver as seen in Figure 2.2.

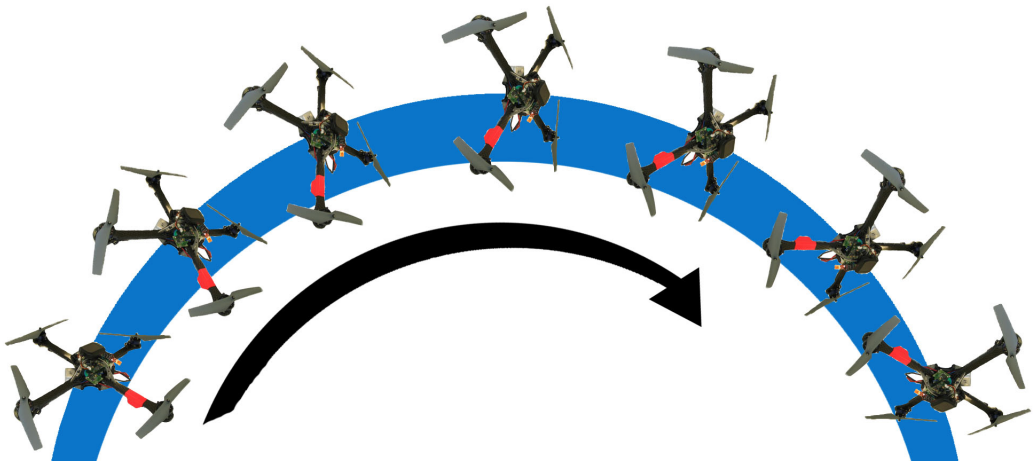


Figure 2.2: The three modes of recovery in a back flip from right to left.

2.1 Previous Work

In [1], the dynamic information flow tracking (DIFT) scheme that was presented in this paper marks all data that is read from the network (typically from a radio device) as untrusted using a tag bit, sometimes also called a taint bit (throughout this paper used the terms untrusted and tainted interchangeably). Tags are propagated throughout the system by the architecture, with no need for modification to the program binaries. No untrusted data can be used as the target address for a control flow transfer such as a jump, call, or return. This restriction makes attacks that overwrite control data, such as buffer overflows and related attacks, impossible. Control data includes return addresses, function pointers, the bases and offsets of linked library functions, and more. Thus the architecture presented stops any attack that overwrites control data and hijacks control flow to take control of a network sensor.

The PhD dissertation [1] discussed how information can flow, using Suh *et al.*

[28] categorization of information flow dependencies into five types:

- **copy dependency** – when data is copied from register to register, memory to register, or register to memory its taint tag is also copied from source to destination,
- **computation dependency** – when an operation is performed on one or more source operands and stored in a destination, the maximum taint of the source operands is stored in the destination taint tag,
- **load and store address dependency** – if the address of a load or store operation has a taint value that is greater than the source then the taint value of the address is copied to the destination's taint tag and
- **control dependency** – when tainted data is used for conditional control flow decisions the program counter is tainted with the maximum taint of the values compared.

The dissertation addressed the problems of the DIFT system especially the control dependency. The key difference between the DIFT systems considered is that the one addressed in [1] taints data with one tag bit, however, the DIFT system studied in this thesis tracks tainted data with vectors.

2.1.1 Security Applications using DIFT

A control dependency occurs when the value of one piece of data affects the execution path of the program, which in-turn affects the value of another piece of data. Consider the C code example below:

if ($x \leq 0$)

y = 0
else
y = 1

The code above has an implicit dependency between the variables x and y, the program counter (PC) is tainted implicitly on conditional control flows.

When tainted data is used for conditional control flow decisions (e.g., x or y in the C code "if (x == y)" compiled into a compare instruction followed a conditional jump in assembly). The program counter is tainted with the maximum taint of the values compared, where control dependencies become a security problem for DIFT is when it is possible for the attacker to launder the taint bit, so that a value under their control is no longer tainted.

The attacker can compromise the system based on his knowledge in control dependencies of the code. The DIFT system starts with trusted data in the registers and the input which is inputted from the network is considered untrusted. The taint propagation scheme correctly handles copy, computation, load-address and store-address dependencies [1].

2.1.2 Model of the Control Dependency in DIFT

In [1], the open-loop control system that throttles how sensitive the DIFT tracking is to these dependencies by a pre-defined constant called *throttling factor*. For each set of dependency a specific throttling factor is set, which remains constant and independent of the input and the previous results of the DIFT system, refer to 2.1.2.

Dependency Type	Throttle Constant	Summary
Copy Dependency	0.99	the throttle factor is multiplied by the loaded taint value and then stored in both the destination and source.
Computational Dependency	1	the maximum taint of the source operands is stored in the destination taint tag.
Load/Store Dependency	1	these dependencies are not throttles currently.
Control Flow Dependency	0.5	the program counter is tainted with the maximum taint of the values compared. The throttle factor for the program counter is currently set to the constant 0.5 and the throttle factor of the flag is multiplied by 0.99.

Table 2.1: Table with dependency constants in [1]

The amount of taintedness in the CPU control path is calculated for each instruction as a moving average y of the program counter's taintedness e, using the equation

$$y' = cy + (1 - c)e. \tag{2.2}$$

2.2 Modeling Computing Systems via Graphs

This Section provides a small overview on complex computational systems modeled via graphs. These papers provided an insight on the methodology which computing systems are approached and modeled via graphs. The DIFT is a system which tracks the dependencies among the instructions and the resisters after executing a snippet of code. The DIFT system can be viewed as a graph which is altered based on the code being executed. More details are provided in Chapter 3. Two papers [9, 13] are considered below which considered graph models for security purposed in computing systems.

2.2.1 A Graph Model for Fault-Tolerant Computing Systems

The paper [9] deals with modeling a system S which can execute an algorithm A. It examines the removal of k nodes from the system, then test if the algorithm will execute without the removed nodes. The system will have a k-fault tolerant (k-FT) realization of S. The paper provides several techniques for designing such realizations for single loop systems. The approach given by the paper evaluates the ability of programs to run with hardware defects in the computer architecture. The faults which occur in a computing system should be first classified and identified in order to determine if the algorithm can be executed on a k-FT system. However, there are some faults, especially physical, which can't be identified. Redundancy and reliability modeling techniques are employed in order to improve the performance of such fault tolerant systems.

Faults in this paper can be hardware (control units, arithmetic processors) and software(executable files, compilers) failures. A system is defined by a graph where

the nodes of the graph are the facilities - in other words, the software and the hardware are components of the system S. The algorithm A will be also represented by a facility graph which includes the facilities that the algorithm uses for execution and the edges are defined by the links of the facilities. k- fault F in the system S is defined by the removal of k nodes, in addition to the connecting edges. The design of a k-FT system is said to be optimal if there is no system that has less number of nodes with the same node fault tolerance value. One more definition is required, the single loop system is a graph consisting 3 or more nodes each node has a degree 2. To construct the k-FT optimal realization, the single loop system when k is even, is as follows:

1. Form the single loop system with $n + k$ nodes.
2. Join the nodes to all the nodes of distance j, where j is greater than 2 and less than $k/2 + 1$.

2.2.2 Directed Graph Epidemiological Model of Computer Viruses

The previous paper has dealt with computing systems which had malfunctions in either hardware or software faults, however this paper [13] deals with the malfunctions of the caused by viruses. The propagation of software viruses is similar to biological viruses. The mathematical epidemiological study of the spread of biological viruses can be extended to software viruses. The epidemiological model considered in this paper is called SIS which stands for susceptible - infected - susceptible. The computer virus is modeled through a random graph of N nodes and $N(N-1)$ edges. The probability adding an edge (i.e a new infection) is p, therefore, the total number of possible edges in the graph modeling the system is $pN(N-1)$. Each node is associated with a birth rate (infection) and a death rate (cured). The epidemic can

be analyzed through a dynamic approximation. The connections of the graph are unimportant for the dynamics of the graph in this approach. The dynamics of the graph is captured by a differential equation which models the infection and cure rate of the graph. Assuming that the node which is infected will only infect i neighbors with a probability b. Another approach is suggested in this paper, which is a probabilistic approximation. It considers the stochastic nature of the epidemic unlike the deterministic model. This model associates the infected nodes with a probability in function of time t and the current infected nodes at that particular time t. Moreover, a transition rate is assigned from each state to another, where the states represent the amount of infected nodes in the system. It can indicate an increase or a decrease. The paper provides more details about the distribution of the states, given certain infection rates taken at different time. More models are given in the paper: hierarchical and spatial model.

Chapter 3

Dynamic Information Flow Tracking System

Before describing the DIFT system, some definitions are needed. The Program Counter (PC) is the register which stores the address of the next instruction to be executed in the Central Processing Unit (CPU). A taint mark is a binary flag used to indicate whether or not a bit, or set of bits, is tainted. In the DIFT system considered we use a vector of floating point numbers. Inputs are assigned random vectors. The length of the vector represents the amount of taint (specifically, the mutual information the tainted data shares with the original tainted input), and the directions of the vectors have no meaning except that correlated pieces of information (with a similar provenance) will have correlated vectors.

A branch is an instruction that can alter the flow of instructions during execution. A branch predictor tries to guess the outcome of the branch based on the success of past predictions. It is mainly used to save time during execution. The actual result will be compared with the prediction made, if the predictor picked the right decision about branching then it is a hit. If the predictor picked the wrong decision then it

is a miss [22].

3.1 DIFT Overview

This document considers a system which tracks information flow at the assembly level. There are several types of instructions such as arithmetic, logic, memory and branch or jump instructions. The system tracks the information at the registers and memory locations of the system by assigning to each a taint vector. This vector associated with the program variables is modified according to the operations which manipulate their values. For instance copying from one word in the system to another causes the taint to also be copied. This is called a copy dependency. Figure 3.1 illustrates when the taint is copied as it is from the input to the *var*.

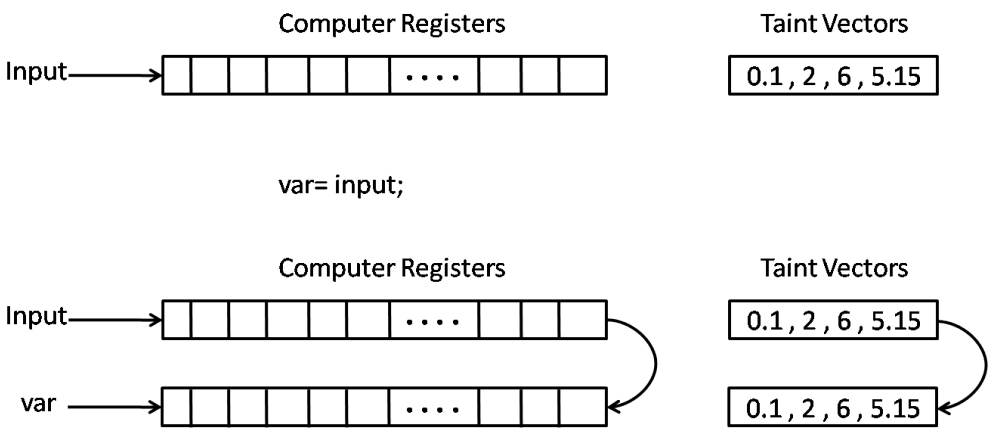


Figure 3.1: Taint propagation upon copying.

Considering another example of adding the values of two variables which each have their own taint vector, the resulting taint will be a Euclidean sum of two vectors. The taint vector of $var2$ will have a representation of the information in both variables a and b. This example uses the addition operation. However, it also applies to logic and other arithmetic operations. Let $\vec{T_{var}}$ denote the taint vector

of the corresponding variable *var*. The below addition operation is an example of a *computation dependency*.

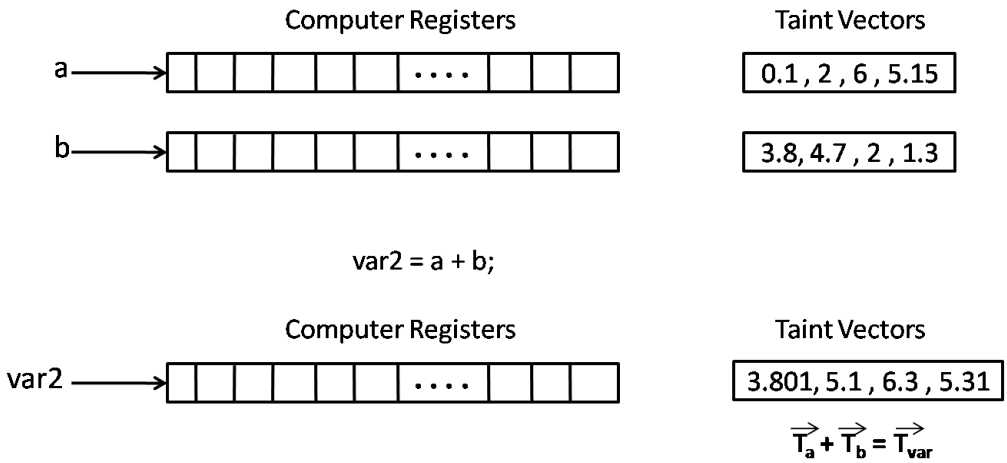


Figure 3.2: Taint propagation upon upon arithmetic or logic operations.

Copy and computation dependencies will not cause the system to saturate. Over-tainting should be avoided since the information flow cannot be tracked effectively when there is too much taint in the system. If everything is always tainted at the end of a computation we can learn nothing about the flow of information for that computation.

When each high-level program is converted to assembly language for execution, the computer will keep track of the next instruction to be executed by storing its address in a designated register called PC. In some cases of the DIFT system, the PC might get tainted in branch operations, therefore the next instructions executed will have their variables tainted as well. This is a *control dependency*. In addition, all the instructions which will have their addresses stored in the PC register will get tainted resulting in a positive feedback loop and over-tainting of the system.

Since 1986 commercial computers have included branch predictors in their computing units [21, 22]. When a branch is encountered the program will pick the next

instruction to be executed based on the past branch results. This is commonly used to make the CPU execute faster since branch results take a long time to be computed. For instance, in Figure 3.3 assume that predictor would pick $a = 1$; as the next instruction to be executed. These predictors are useful since later they are used to reduce the taint in the system and as an avoidance mechanism to reduce the taint. In Figure 3.3 below, the system will take $a = 1$; as the next instruction but this branch result was taken based on the value of the input which is marked as untrusted by its taint. Consequently, all the instructions which are executed next are tainted, as well as the PC register since the address of the next instruction is stored in it.

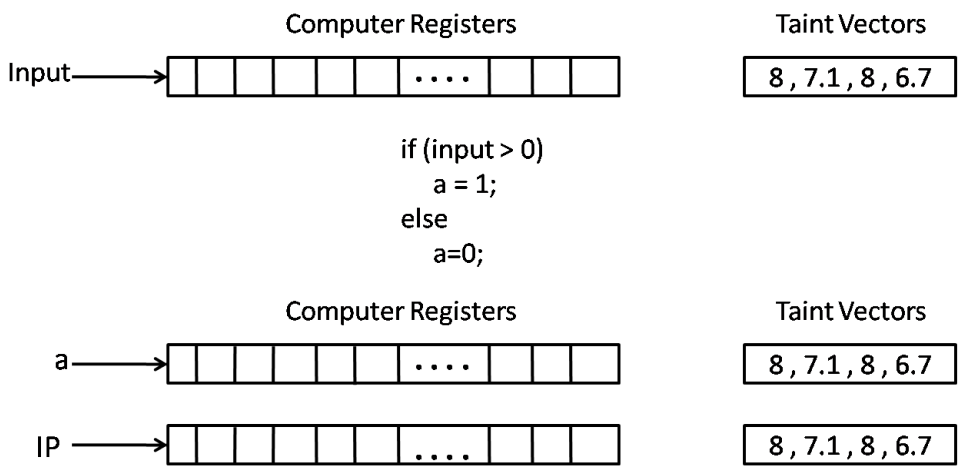


Figure 3.3: Taint propagation upon branch and jump operations.

Aside from the branch operations, another problem is encountered when the program is loading a value from memory. When the program requests the data from a memory location information flows from the address used for the lookup into the value that is loaded from memory. This is called an *address dependency*.

3.2 Taint of PC Vector

After examining the DIFT system and researching the existing models for computing systems. The taint for the PC is a vector of N decimal values, the vector is modified based on the instructions in the program. The taint vector variations are random since each program has different instructions. The input to the taint vector is the additional taint variations due to the new instructions.

The series of instructions cause the taint associated with the taint vector to get modified. In order to measure this variation, the angle between the previous vector and the updated vector is calculated. The size of the PC taint vector is monitored to check if it is constant. In addition, to measure the variation of the taint with respect to the input, the cosine similarity is designed to indicate the difference between two programs after performing the cosine similarity.

3.3 Graph Representation of Instruction Dependencies

3.3.1 DIFT Graph Model

A graph representation of the DIFT system is needed for defining the dependency of the tainted registers/memory locations. The instructions are the nodes and the edges represent the dependency of the instruction's variables. Consider $\mathcal{N}$ set of instructions – nodes, connected by a set of links $\mathcal{L}$. The graph is directed and the edges are weighted based of the vectors modeling the dependency of the different instructions. In order to construct the graph of the dependencies in a program, some instructions should be given. Thus, we have generated a random set of assembly

instructions to analyze the dependencies among them. The links are denoted as l_{ij} from node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$ and every link is weighted by a finite taint value.

For a given set of instructions $\mathcal{I}$, it includes a number M of nodes $\in \mathcal{N}$. The nodes are connected with links which represent the dependency based on the operands and the memory locations involved in I_i. The links are denoted as $l_{i,j}$ indicating the dependency between nodes i and j, the link is associated with a register, memory location or a flag. Weight of each link can be defined as ${d_l}^\tau$ where τ is the taint value associated with a specific operand. In addition, the link also represents the type of dependency which the two nodes are related to each other, refer to the example below to see the dependency representation via colors. There is a function f where the value of the taint, register or memory location which is tainted by τ is mapped to the link $l_{(i,i+m)} \in \mathcal{L}$ where $m \in \mathbb{Z}_+^*$ is a positive integer indicating offset of the dependent instruction from instruction i. Let f be a function where $f(l_{(i,i+m)}) = {d_l}^\tau$ $f : \mathcal{L} \to \mathbb{R}^N$, where τ is the taint value associated with a specific operand (memory location or register). A general representation of the graph given in Figure 3.4 to represent a set of instructions executed in the DIFT system. ${d_l}^\tau$ is defined to be a vector of size N – the size of the taint vector defined in the DIFT system, where ${d_l}^\tau \in \mathbb{R}^N$. N is a preset value of the taint size in the DIFT system, the bigger the value of N the more the DIFT system is representative of the flow of information.

3.3.2 Graph Example

A random set of assembly instructions is generated in MATLAB at run time. The instructions have arithmetic or branch dependency. The initial computing system should have 32 or 64 registers in its representation however, for the sake of illustration, the number of registers is assumed to be 8. The analysis in this section is done based on the assumption that there is a probability that the number of control

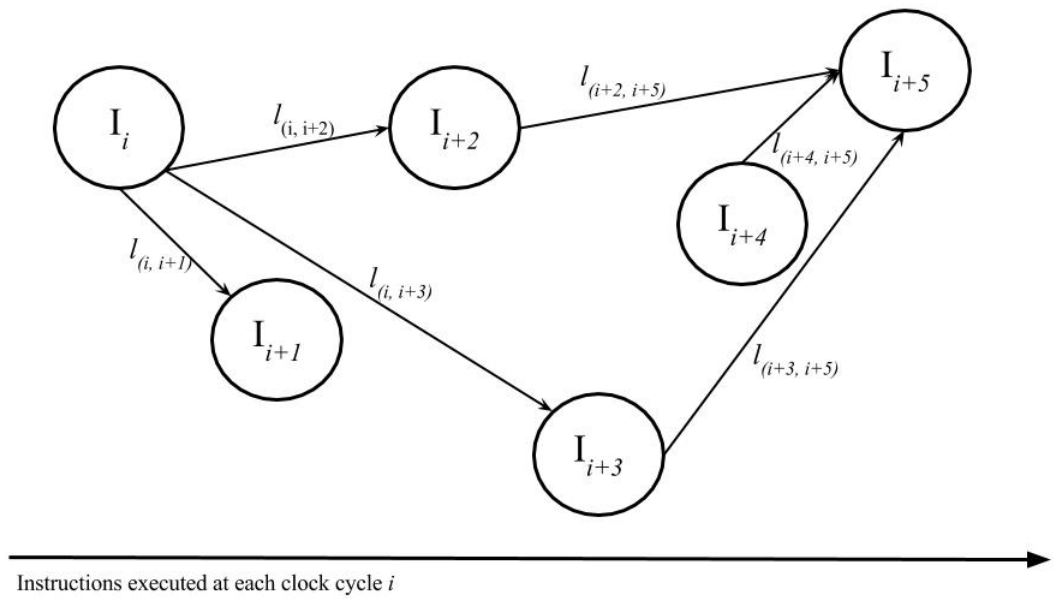


Figure 3.4: A general graph representation of a set of instructions and dependencies as nodes and links.

dependency instructions is less than that for the arithmetic dependency. In addition, the instructions are narrowed down to two formats as given in the example below:

- **add** $\$r1$**,** $\$r2$**,** $\$r3$ – add the content of registers
- **beq** $\$r1$**, address** – command checking if the content of $\$r1$ is equal to zero if this condition is met then the PC register is modified to the new address.

After generating the set of assembly instructions, dependency is defined when an instruction uses the data of a register which has been previously modified or saved by an executed instruction. The type of the dependency is determined by checking the type of the instruction using the register. For instance, consider the code sample below:

add $\$r1,\ \$r2,\ \$r3\ \mathrm{I}_1$

sub $\$r4,\ \$r1,\ \$r2\ \mathrm{I}_2$

The use of $\$r1$ as an argument for the second arithmetic instruction I_2 will result in an arithmetic dependency. It should be noted that when an instruction updates the value of register $\$r1$ given that the previous instructions have been executed, then there can not be a dependency with $\$r1$ on I_1. It now depends of the most recent instruction which updated its value.

All the dependencies are directional; however, they have one direction only; since all the dependencies are from the instruction which generate a value towards the instruction which used that same value, thus, it is insignificant to indicate that. Consider the example given above; all the directions of the edges will be similar to (I_1, I_2) I_1 to I_2.

To get a simpler idea on a small scale consider this quick example below of the following instructions:

fadd $\$r1,\ \$r2,\ \mathrm{I}_1$

inc $\$r1\ \mathrm{I}_2$

sub $\$r3,\ \$r1,\ \mathrm{I}_3$

The dependency representation of the the above small code is shown in Figure 3.5 below:

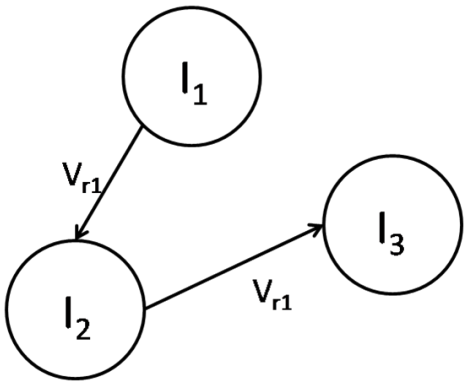


Figure 3.5: Graph representation of the three instructions.

The Figure 3.6 below, is the simulated result of having 100 random instructions in the program and having a 0.75 probability that the instructions would be arithmetic instead of control ones. Random Program dependency graph: blue edges for arithmetic dependencies and red for branch dependencies. Nodes are represented in circles.

It is measured by identifying the important nodes in a graph. The definition of an important node in this paper is the number of edges which is similar to the friendship network. The degree of each node is important since one can tell if the variable or register contains valuable information for other instructions. It can be used in analyzing the information flow in the system. After several runs of the MATLAB code provided in the appendix the degree centrality vary from 0 to 20.

Along the lines of modeling two different programs which do the same thing however their translation to assembly is different, merge sort and quicksort are considered to illustrate the difference between these two programs. Merge sort has a worst case scenario comparisons $O(nlog(n))$ [14] while quick sort has $O(n^2)$ [7] where n is the number of elements in the list being compared. Branch instructions are used when a piece of code requires comparison. Therefore, merge sort has less comparisons than quick sort in Figures 3.7 and 3.8.

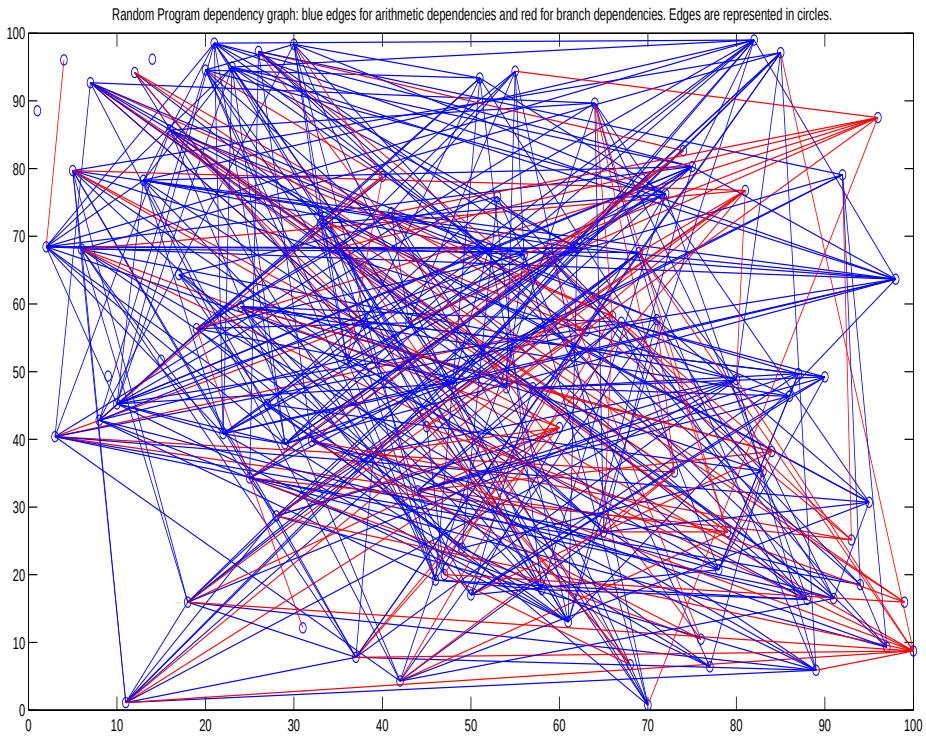


Figure 3.6: Graph representation of the instruction dependencies on MATLAB.

Graph Theory Analysis

Small world effect is not possible due to the fact that there are no loops in the system and there is no way we can construct a certain number of variables which are dependent on each other in a loop manner since it is a one directional dependency.

The graph is built to identify the dependencies of the graph and count the degree of each node. Such graphs can be used for information flow analysis which is crucial in security to identify the significant nodes and enforce more protection on them.

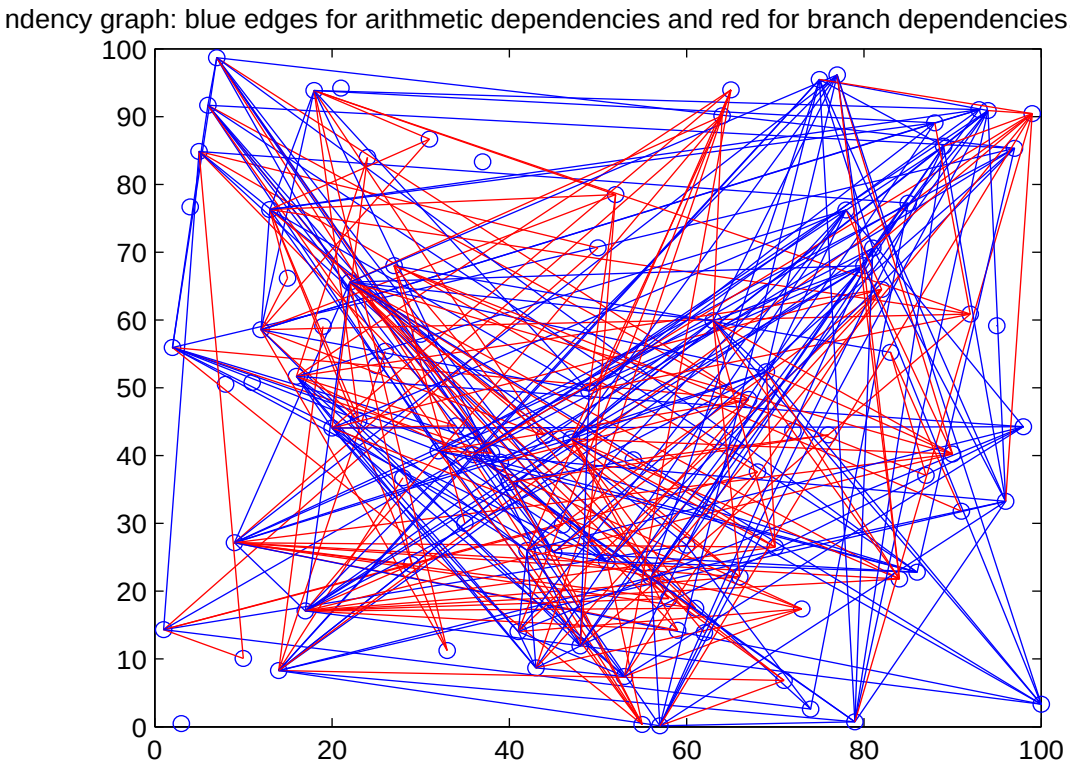


Figure 3.7: Graph representation of the instruction dependencies on MATLAB of Merge Sort.

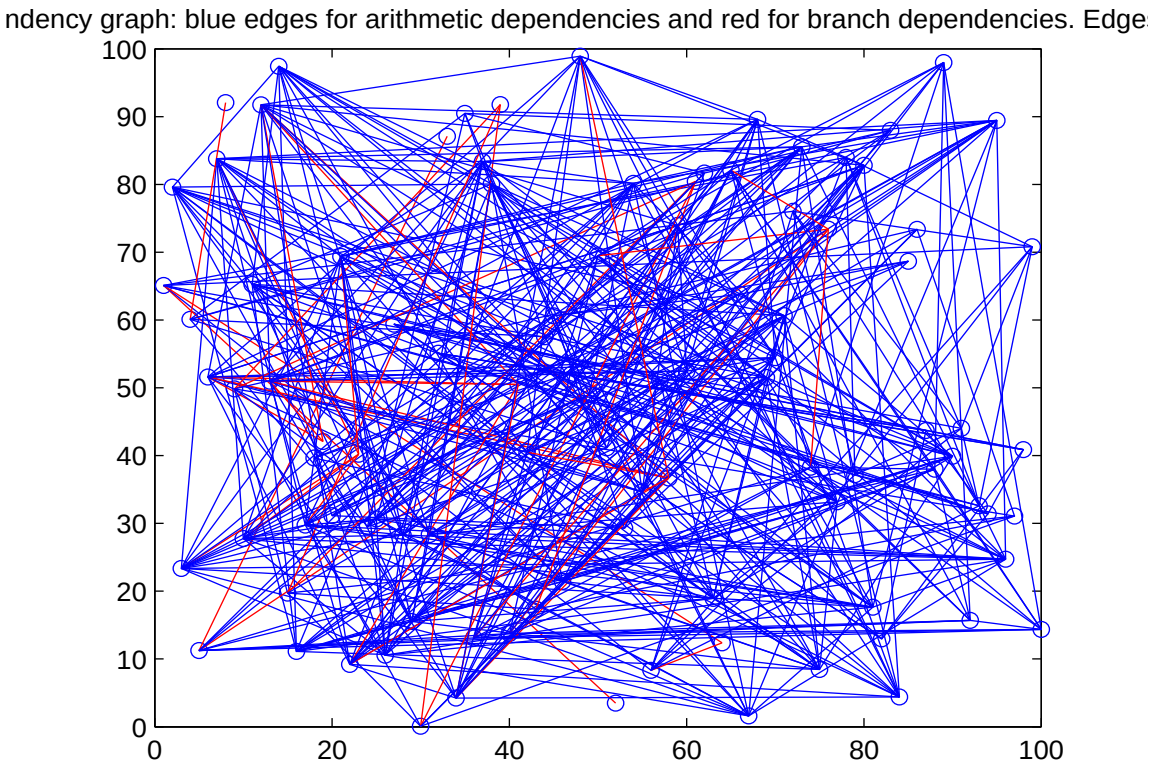


Figure 3.8: Graph representation of the instruction dependencies on MATLAB of Quicksort.

Chapter 4

Preliminary Mathematical Model

This Chapter introduces the model of the PC taint behavior in the DIFT system. The taint is a value quantifying the modifications done to the bits of the registers in a computer architectural system. The taint is a vector of length 65. Each value represents the data operations performed on the data in the registers. The control dependencies have a direct effect on the PC register- which controls the instruction flow of the program. The taint of the PC register should be well monitored since it contains the address of the next instruction to be executed, in other words determining the execution flow of the program.

4.1 Problem Formulation

After giving an overview about the DIFT system, this Section focuses specifically on defining the problem of PC over-tainting and the information destruction in the DIFT system. Over-tainting occurs when the taint associated to the PC register is high, causing all the taint vectors representing the data flow in the registers used in the next instructions to be highly tainted - since the PC register specifies the instruction

to be executed next in the pipeline. This phenomena is called over-tainting. The consequence of over-tainting is the inaccurate representation of the information flow in the program, this usually happens in loop and in conditional instructions. The trace of the taint in the DIFT system is over estimated, thus, the misrepresentation of the information flow of a set of instructions will be inconclusive.

For illustrating the effect of over-tainting in the PC register, consider the following conditional branch x86-instruction **JNZ** jump if not equal. The branch instruction jumps if the zero flag **ZF** is equal 0 – meaning that the jump is associated with a condition that checks if two variables are equal. Therefore, if the condition is false – meaning the two variables are not equal, then the flag is set to 0 and the execution of the instruction should jump to a pre-defined address by updating the PC register content.

When the DIFT system is updating the taint vectors when the program is being executed, it encounters several jump instructions. The taint transfers from the variables of the condition statement to the **ZF** and if the branch was taken, the taint transfers from the **ZF** to the PC register. Since the PC register holds the address of the next instruction to be executed, the taint associated with the PC register is transferred to the rest of the instructions following the jump instruction in the pipeline. An example is provided below in order to demonstrate the taint which may transfer from the PC register to the

$if(i == j)$	jnz $L1$
$i++;$	addi $\$r1, \$r1, 1$
$j--;$	$L1$: addi $\$r2, \$r2, -1$

Figure 4.1 below illustrates the propagation of the taint while the execution of the above assembly code. The left part of the figure shows the execution of the instructions, the arrows indicate how the execution is happening. The right part

demonstrates the propagation of the taint while the execution is happening. For clarity the taint in this example is assumed to be colors, however, in the DIFT system the taint is a vector with a length N. The taint in Figure 4.1 is denoted as T($\$r1$) where $\$r1$ is a register in the pipeline. Figure 4.1 below illustrates the propagation of the taint in a set of instructions of branch instruction flowed by arithmetic instructions depending on the result of the branch. The registers $\$r1$ and $\$r2$ are initially tainted with green and yellow respectively. Assume that the **ZF** depends on the values of $\$r1$ and $\$r2$, therefore the value of the taint in the two registers is combined and transferred to **ZF** resulting with a blue taint. The branch is conditioned on the **ZF** thus the taint is transferred to the PC register, since branch instructions update the value of PC and changing the flow of the program. After the jump occurs, indicated in red in the figure, the taint is transferred to the $\$r2$ register resulting with over-tainting. If the T(PC) is high and it can't be cleared it will taint all the instructions to be executed causing a misrepresentation in the taint at the end of the program. The end result of the DIFT system will be inaccurate in this case and the vectors will reflect the flow of the program.

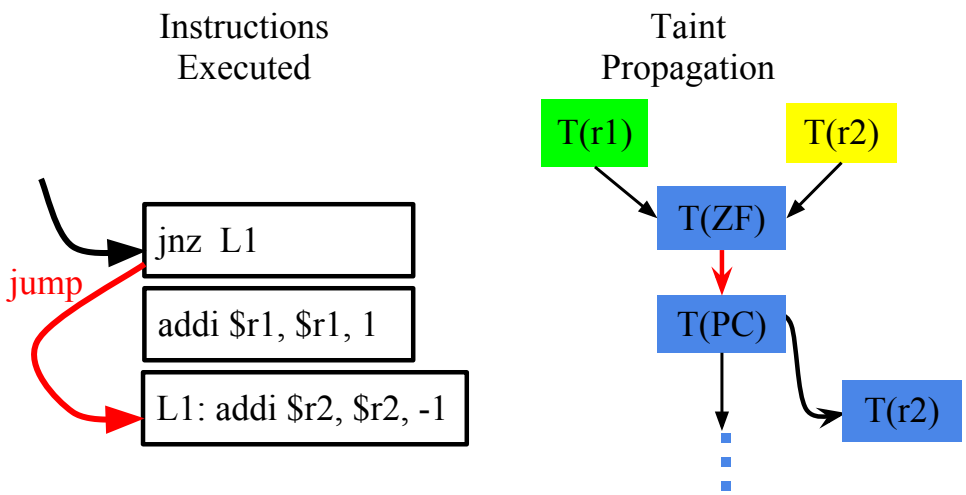


Figure 4.1: Example 1: Propagation of the Taint during conditional jumps.

Once the PC is over-tainted, the DIFT system does not give a correct information

flow in a set of instructions. Another problem is the destruction of information, where the information held in the taint vector is getting destroyed because of the new taint added. When the DIFT system encounters an arithmetic operation or branch instruction (most of these instructions use two registers in their operations), it combines the vectors by adding the taint vectors. A two dimensional illustration is given in Figure 4.1, the taint vector in the DIFT system is of length N, where N is usually a large number by design. Information destruction occurs when the taint is accumulated in the vector in such a way that the newest taint added is the most significant. The taint is most affected by the most recent instructions executed. The older information is being destroyed and replaced by newer information. In order to reduce this effect control theory is used to reduce information destruction especially in the branch instructions. The following Section provides a model for the taint associated with the PC.

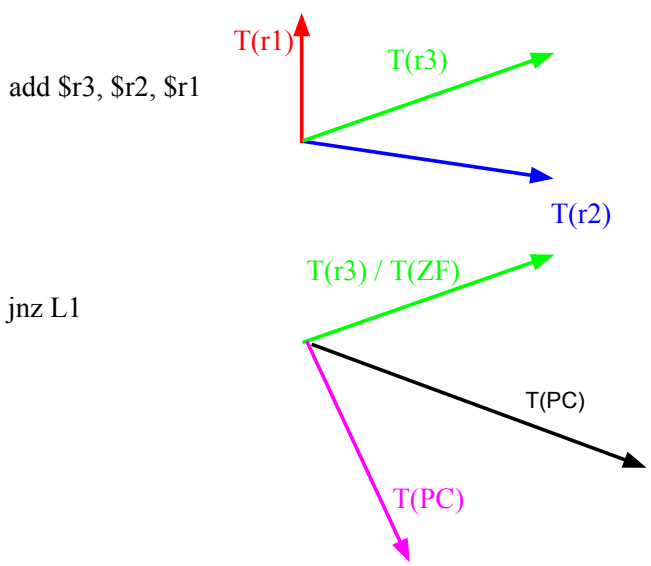


Figure 4.2: Example 2: Information destruction upon taint propagation.

Figure 4.2 , provides an example of the information destruction. First, assume that the jump instruction depends on the value stored in $\$r3$ hence the taint asso-

ciated with $\$r3$ is transferred to the **ZF**. The first instruction transfers the taint to T($\$r3$). It is transferred later after the execution of the following instruction to the taint of the PC. The resulting taint in the PC is a combination of the taint of $\$r3$ and PC. It can be easily seen that the effect of the taint associated with $\$r1$ is now gone.

4.2 Hybrid Control Methodology

As a solution for the taint problem, the system should have some limits on the taint value of the overall system so that if the taint value is high then certain actions should be taken to avoid over-tainting. The instructions should stop execution on the CPU and wait for the result of the branch predictor. Based on the predictor's result the DIFT system will modify the taint vectors. This can be viewed as an analogy with a thermostat system. The system is modeled in a hybrid manner since it has a switching mode whether the PC register gets tainted or if the DIFT system would reach the threshold of over-tainting.

Why do branch predictors decrease the amount of the taint? Branch predictors pick a branch and execute the corresponding instruction according to the past history of branches and misses. When something is predicted correctly then it is most likely has no significant information, because the result was expected. But, when something cannot be predicted correctly then it is unknown and we learn more information from the result. By this argument, we can say that if the branch predictor chooses the correct branch then less taint should be propagated than if the branch predictor misses.

The constraints on the taint vector of the PC are limiting its 2-norm within the range of 32 to 16. In addition to this constraint the throttle factor is determined through a feedback loop in order to avoid information destruction. The throttle

factor is responsible for determining what percentage of each vector is being added to the new taint vector after branch operations. Therefore, it preserves important data by computing a throttle factor scaling the old information up or down. This decision is made through monitoring the branch misses and hits which are recorded in the branch predictor.

The DIFT system is modeled through control theory as a hybrid system. The state of the DIFT system changes when branch misses occur, *i.e.* the taint should be modified. The 2-norm of the taint increases as the program progress, the DIFT system cuts it down back to 8 in all cases. The Figure 4.3 below gives a block diagram of the DIFT system after adding controller for correcting the throttle factor for the DIFT system. The plant is the taint vector of the PC which accumulates the additional taint from the new instructions executed. The branch predictor is an important part of the DIFT system where it determines the state of the controller in the cases of hits and misses in branch conditions. The branch predictors vary in each processor some examples of predictors are: saturating counter, two-level adaptive predictor, local branch prediction, global branch prediction, agree predictor, hybrid predictor and more. They are all based on the history of the executed branches and how many times the predictor was right in branch predictions. The throttle factor is used to reduce the 2-norm of the PC taint vector. In addition, based on the previous discussion, the taint value which calculated through combining two vectors should be throttled in order avoid information destruction.

One of the main reasons to use the hybrid system in representing the dynamics of the taint vector associated with the PC register, is achieving maneuverability and controllability in the system. Since these two notions are complementary, the system is viewed in a hybrid model where one state maneuverability is high and the second state the system is controlled in order to restore the correct amount of taint in the PC vector.

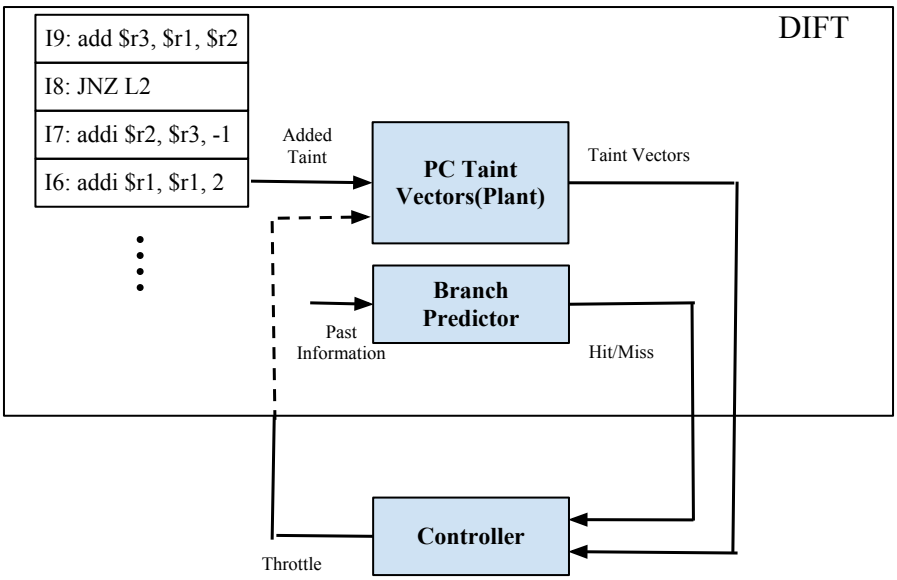


Figure 4.3: The DIFT system block diagram with the controller.

The output of the plant is the taint vector $\{y|y \in \mathbb{R}^N\}$ of the PC of size N . The DIFT system as represented above has a continuous state, $\{x|x \in \mathbb{R}_{>0}\}$ which is the norm of the taint vector in the PC. The norm of the taint's dynamics in the PC vector is monitored and has a first order representation in the hybrid model. The Figure below 4.4 introduces the hybrid model of the system.

The discrete states are $Q = \{q_0, q_1\}$.

- q_0 := branch predicted correctly: hit
- q_1 := branch predicted incorrectly: miss

Let T_L and T_H be the low-level and high-level taint norm respectively, where T_L and $T_H \in \mathbb{Z}^+$.

Vector Field $f(.,.) : Q \times X \to \mathbb{R}$

$Dom(q_0) = \{x \in \mathbb{R}|x < T_L\}$ and $Dom(q_1) = \{x \in \mathbb{R}|x > T_H\}$.

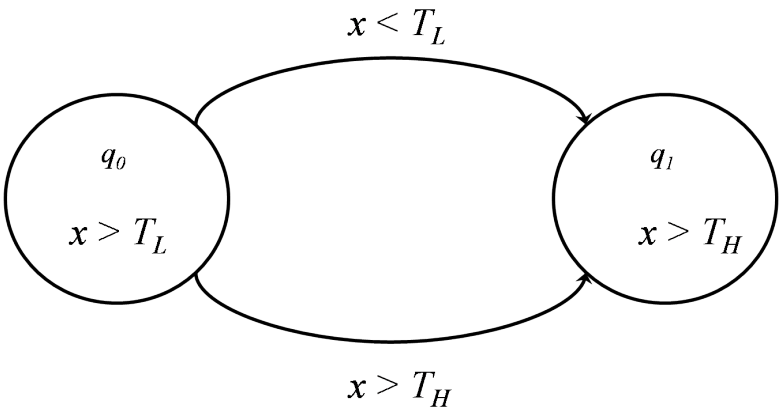


Figure 4.4: Two modes of the system.

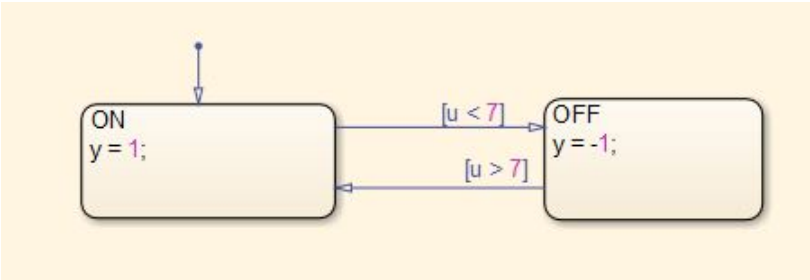


Figure 4.5: Two modes of the system in simulink.

$E = \{(q_0, q_1), (q_1, q_0)\}$ the possible switching in the system.

$G(q_0, q_1) = \{x \in \mathbb{R} | x < T_L\}$ and $G(q_1, q_0) = \{x \in \mathbb{R} | x < T_H\}$,

Figure 4.5 shows the state chart in the system where the T_H was set to 7 and the output of the system is positive or negative integer which will pick the state variables of the system.

The above Figures 4.6, 4.7 and 4.8 describe the taint in the PC vector and its variations in the system after running a program where it modifies the taint vectors associated with the registers and the memory locations in the computer. The taint in the PC register is decreased when the branch predictor hits *i.e.*, the prediction

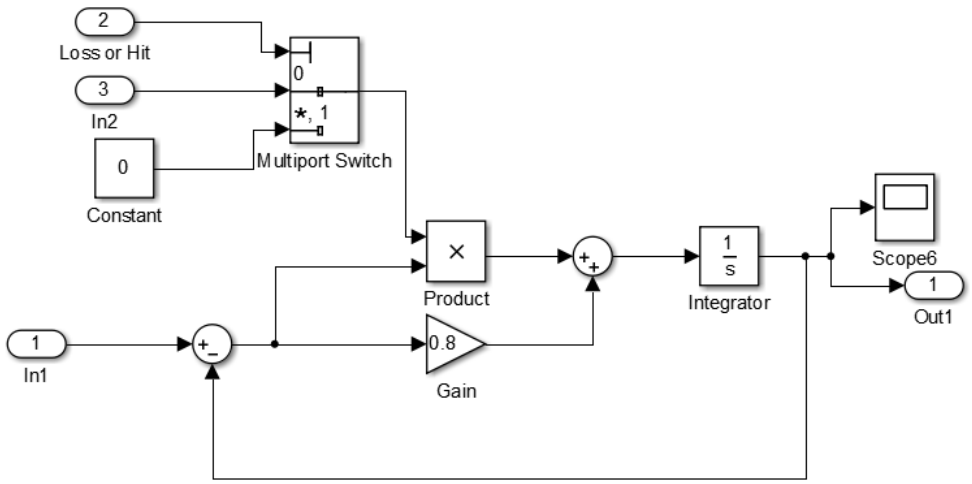


Figure 4.6: Taint dynamics as modeled in Simulink.

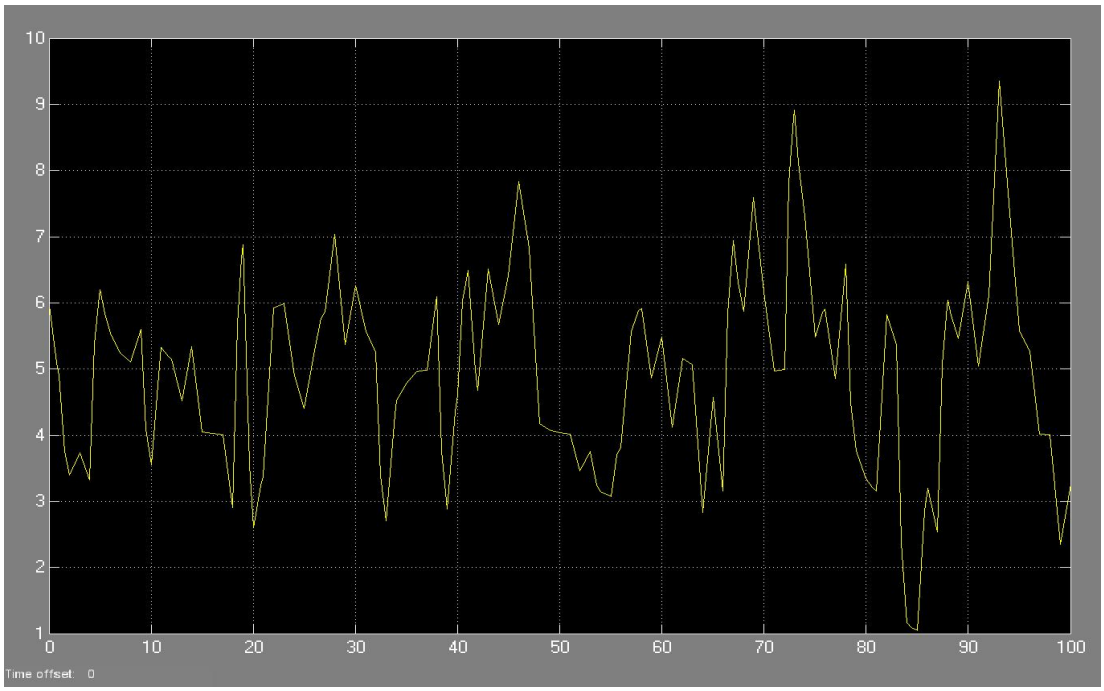


Figure 4.7: The 2-norm of the taint in the system.

was correct. The taint length should be reduced based on how much the predictor

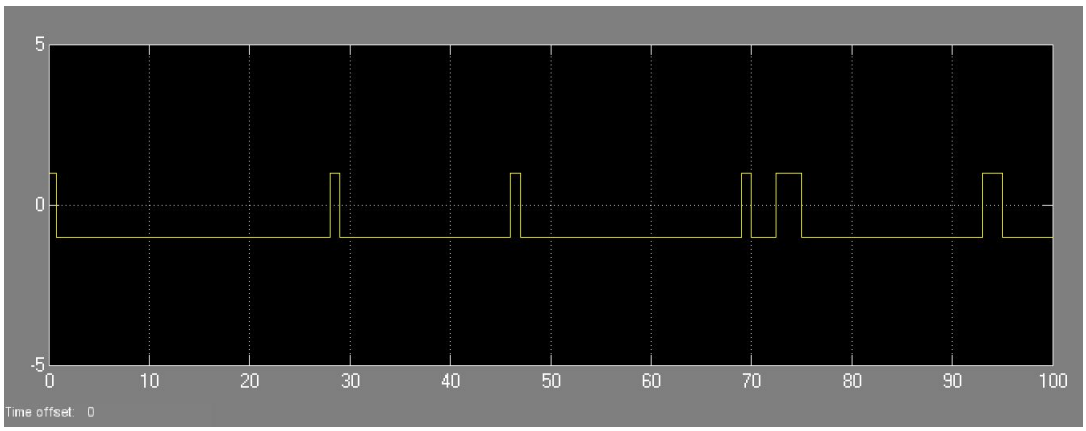


Figure 4.8: Instances when the controller is active or inactive $Q = \{q_0, q_1\}$.

is guessing the correct instruction to be executed next. When the taint bounds are violated the system goes into the conservative state q_1 where the taint is reduced according to the hits of the branch predictor and the taint overage in the DIFT system. The controller in this Section is a PID controller where the values have been tuned in order to achieve the smoothness in the taint reduction.

Figures 4.9, 4.10 and 4.11 are screen caps of the Simulink files used to design the DIFT system. The controller used for the taint system is a PID controller with the following constants. The taint norm doesn't exceed the bounds which was set to 8 in this design and DIFT model. The PID compensator has the following characteristics in 4.2. The purpose of using compensation that can be employed to help fix certain system metrics that are outside of a proper operating range, which is the case in this model.

However in this paper the solution for address dependencies are left for future work. We discuss the first two states. The system switches between two modes where each will affect the taint differently. The DIFT system in general has two modes of operation: q_0 is tainting regularly at the program's pace and, (ii) q_1 is when the system should wait for the predictors result to decrease the taint in the system.

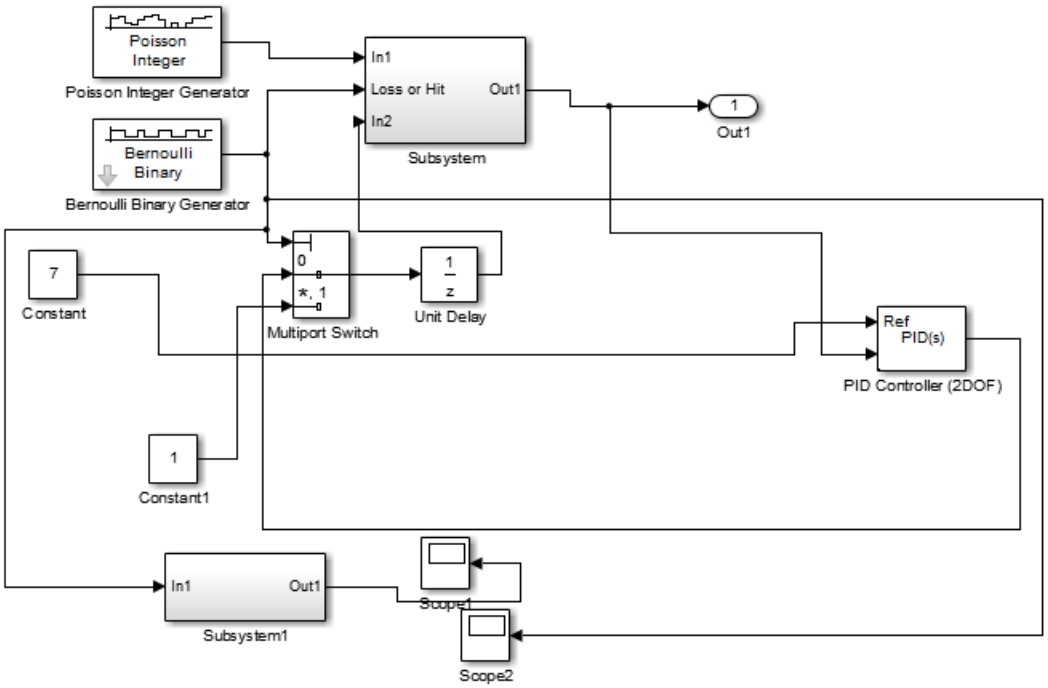


Figure 4.9: The overall system with the tuned PID taint norm controller.

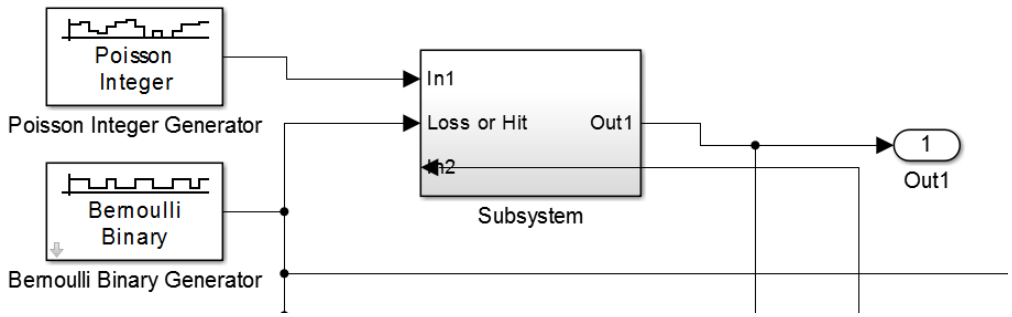


Figure 4.10: The input of the system when real time data is not available.

The hybrid model of the DIFT system has two modes of operation. The first is when the system is executing at normal pace but while monitoring the amount of taint and checking if the PC register is tainted. The second mode is when the system is waiting for the branch predictors and the memory location predictors to reduce the amount of taint in the system, or, more specifically, the PC register. It is ensured

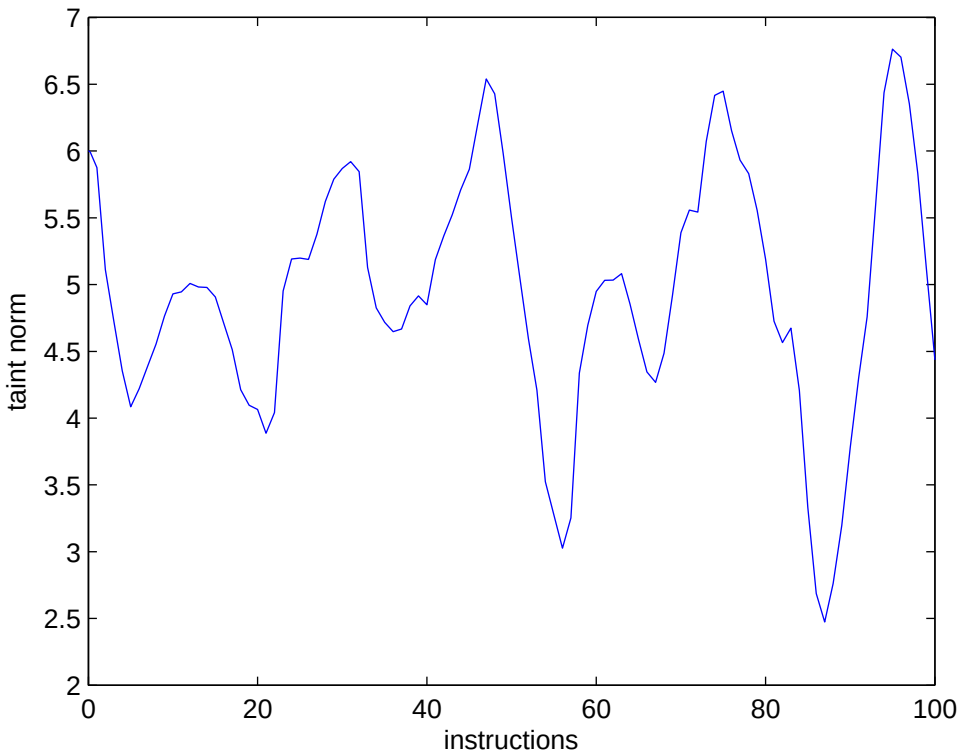


Figure 4.11: The input of the system when real time data is not available.

by having guards on each mode. By definition, an invariant is a necessary condition for switching between the two modes of the DIFT system. The switch between the

PID Constant	Value
P	0.3456
I	0.0001
D	-0.648
N	0.5333

Table 4.1: Table listing the constants of the PID controller where N is the filter coefficient which leads to a smoother controlled output [4]

.

two modes of the system happens when the guard is satisfied. Whenever the location invariants fail the new mode (state) will be determined by the guards.

Figure 4.8 describes the two different modes of the system. The first will keep the system running when the taint level is below 50% while the second will force the system to wait for the result of the predictors. The model of the system ensures that the system will keep the taint bounded between the limits by the set guards.

4.3 Tandem Queue Methodology

The taint associated with the PC register has a similar behavior as a queue. The taint is used to model the data operations done to the content of the PC - the instruction's address to be executed next .The model which will be presented will be similar to a queue model but it includes some modifications in order to reflect the variations of the taint vector associated with the PC.

After examining the DIFT system and researching the existing models for computing systems. The taint for the PC is a vector of N decimal values, the vector is modified based on the instructions in the program. The taint vector variations are random since each program has different instructions. The input to the taint vector is the additional taint variations due to the new instructions.

In correlation with several models of computing systems, a model has been used to represent the taint in each element in the vector. After studying several models in the computing systems, it is significant to introduce the model for the tandem queue. A tandem queue consists of several queues connected in series, it is used to model complex systems such as multi-tiered e-commerce applications with interconnected components. These applications are modeled a networked queuing systems, in other words, a tandem queue.

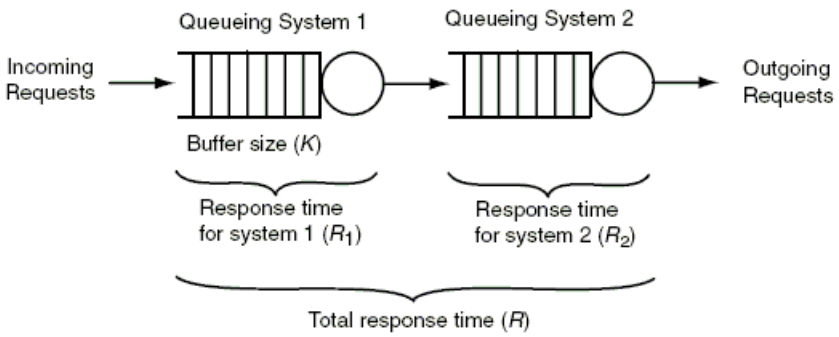


Figure 4.12: Tandem queue in a networking system.

In the DIFT system, each element in the taint vector will be modeled as a queuing system. Each queue will represent the taint value in each element. The taint value can be negative, thus, the queue will be associated with the absolute value of the taint. However, it will be noted that the queue is associated with a negative taint. For a better understanding of the tandem queue, the Figure 4.12 below represents two queuing systems in a simple networking concept.

The tandem queue is modeled by monitoring the average response time of each queuing system. The overall system will measure the response time of all of the individual queuing systems in the tandem queue. The state variables are $r1$, $r2$ representing the response time in each queue. The input to the tandem queue is the offset value in the buffer at time k, and the output is the average response time of all the queues.

Comparing the DIFT system to the tandem queue, the input to the networked system is the available requests in the queues while in the DIFT system is the additional taint added at time k to the taint vector associated with the PC register. The output of the DIFT system is the angle between the successive taint vectors associated with the PC register measuring the taintedness of the system, while the output for the queuing system measures the response time of the overall system.

The queuing system is modeled through using the ARX (Auto-Regressive with external input estimator) model in MATLAB using the system identification toolbox. It will return the model of the system as a state-space representation. The same modeling scheme can be applied to the DIFT system, since the two systems exhibit similar characteristics, as explained above. The only difference between the two systems is the output.

The data is recorded from the DIFT system by saving the taint vector associated with the PC register. To assess the data and the degree of difficulty in identifying a model, you first estimate the simplest, discrete-time model to get a relationship between $u(t)$ and $y(t)$– the ARX model. This black-box approach does not require you to model the physics of your system. The ARX model is a linear difference equation that relates the input to the output. System Identification Toolbox uses these parameters to compute delayed inputs and outputs in the difference equation [29].

Discrete-time ARX model: $A(z)y(t) = B(z)u(t) + e(t)$

Chapter 5

Mathematical Model, Validation and Results

5.1 Mathematical Model of the DIFT system

The DIFT system has a hybrid model which is built based on the similarity between the taint accumulation and the queue model. There are three states which the system can be operating. The system is a non-linear time variant system. The system could be modeled with a difference equation and discrete variables however the choice of having it modeled as a continuous system represented by differential equations due to the delays encountered in branch instructions especially when misses are encountered and PC register's taint is not updated at each instruction, although the value of the PC register is. The discrete model is left for the future work the sampling time should be the clock cycle of the computer, since at each clock cycle an instruction is being executed.

The data transfer is studied on the four bytes of the register, where the taint is added and subtracted from a byte of the register. The dynamics of the taint is viewed

in the same manner as the packet transfer in the TCP connection model which was presented in the Chapter 2. The taint is modeled as follows

$$\dot{x} = -(x - \tau_{I_n} y_{I_n}), \tag{5.1}$$

where $\dot{x}$ is the taint dynamics in the byte of the PC register, x is the rate of increase of the taint in the specific byte, y_{I_n} is the taint decrease rate in the byte due to the instruction I_n and τ_{I_n} is the tainting factor for the corresponding instruction I_n. The variables are updated when the taint of the PC register is updated. The value of τ_{I_n} varies depending on the state of the system and the instructions executed in the pipeline. y_{I_n} is the input to the model representing the amount of taint being added or removed to the taint vector associated with the PC register. The dynamics of the taint in the PC register is modeled as a hybrid system and it is illustrated in the diagram below.

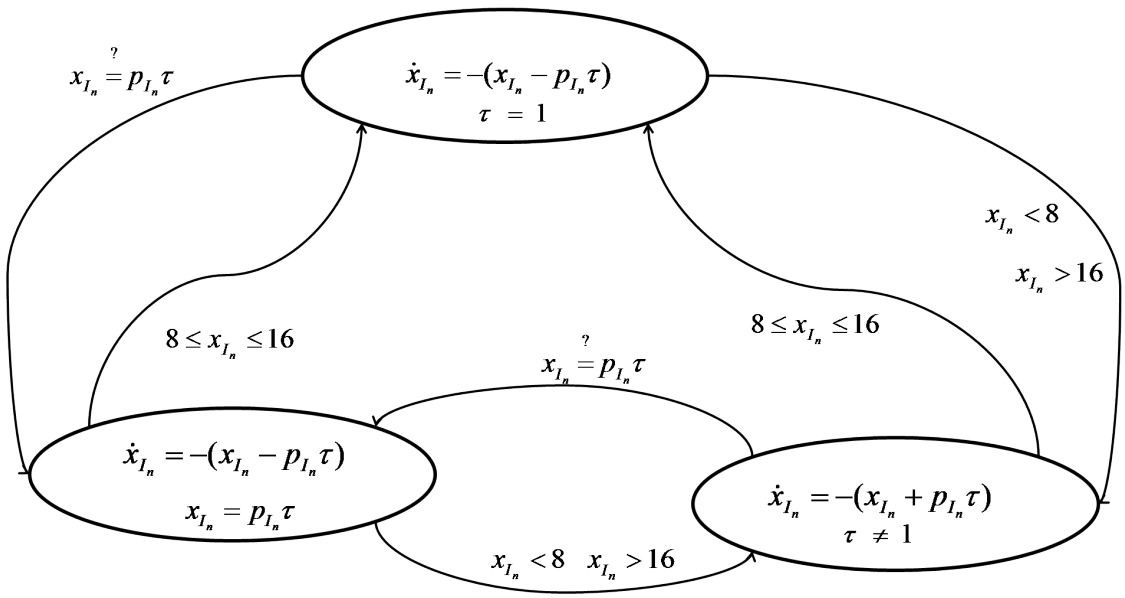


Figure 5.1: The hybrid system representing the taint dynamics in a program.

The DIFT system hybrid model has three states which represent the dynamics of

the taint in the PC register as a queue. There are three possible states in the model presented above where the taint is quantified by the rate of variation of taint. The states are described as follows:

- **Constant Taint $\dot{x_{I_n}} = 0$:** The taint in the PC vector shows no variation, it happens in the case where the taint in the PC vector is not altered in the DIFT system. For instance, some instructions are executed and no changes should be done to the taint associated with the PC register, as in arithmetic instructions $(addi, subi)$. Such instructions will only change the taint in the registers storing the values involved in the operation thus there is no need to update the PC taint. A state is needed when the taint is constant, x_n stays constant or in other words $\dot{x_n} = 0$. In this case x_n and p_{I_n} are equal to zero, there is no change in the PC taint $x_n \neq p_{I_n}$.

  The above discussion can be summarized by the below equations, the state has no changes in the variables of the differential equation 5.1.

- **Increasing Taint $\dot{x_{I_n}} > 0$:** The taint in the PC vector increases in this case since the τ is negative. In this case the taint associated with the PC register is increasing since the instruction which was executed has tainted data and should be transferred to the PC register since it affects the next instruction to be executed – for example, branch instructions. A state is needed in order to indicate the increase in the taint based on the instructional dependency.

- **Decreasing Taint $\dot{x_{I_n}} < 0$:** The taint in the PC vector decreases based on the dependency of the previously executed instructions. The decrease in the taint associated with the PC register is the desirable since DIFT systems have a misrepresentation with continuous increase in the taint. Some cases in the code being studied in a DIFT system, the next instruction to be executed is dependent on a non-tainted variable or a variable that has a low taint value,

thus it causes the taint to decrease. A state is needed to represent the decrease in the taint dynamics.

The hybrid model provided gives a partition between the continuous states x and the discrete states q. The hybrid automata for the DIFT system given by:

- the modes of the system $Q = \{1, 2, 3\}$
- the domain map Domain: $Q \to \mathbb{R}$ for each $q \in Q$ in which x_{I_n} changes

$$\text{Domain}(1) = \mathbb{R}, \text{Domain}(2) = \mathbb{R}, \text{Domain}(3) = \mathbb{R} \tag{5.2}$$

- a flow map $f : Q \times \mathbb{R} \to \mathbb{R}$ describing the variation of the state

$$\begin{aligned} f(1, x_{I_n}) &= -(x_{I_n} - \tau p_{I_n}) \\ f(2, x_{I_n}) &= 0 \\ f(3, x_{I_n}) &= -(x_{I_n} + \tau p_{I_n}) \end{aligned} \tag{5.3}$$

- a set of edges $Edges \subset Q \times Q$, which identifies the pairs (q_i, q_j) such that a transition from the mode q_i to the mode q_j is possible where $q_i, q_j \in Q$ such that $Q = \{(1,2), (2,1), (3,2), (2,3), (3,1), (1,3)\}$.
- a guard map $Guard : Edges \to \mathbb{R}$, which identifies, for each edge $(q_i, q_j) \in Edges$, such that $i \neq j$ the set Guard

$$\begin{aligned} &Guard(1,2) : \dot{x_{I_n}} = 0, && Guard(2,1) : 8 \geq x_{I_n} \geq 16 \\ &Guard(3,2) : \dot{x_{I_n}} = 0, && Guard(2,3) : \{x_{I_n} < 8\} \cup \{x_{I_n} > 16\} \\ &Guard(3,1) : \{x_{I_n} < 8\} \cup \{x_{I_n} > 16\}, && Guard(1,3) : 8 \geq x_{I_n} \geq 16 \end{aligned} \tag{5.4}$$

- a reset map $Reset : Edges \times \mathbb{R} \to \mathbb{R}$, which describes, for each edge $(q_i, q_j) \in Edges$, where $i \neq j$ the value of the continuous state during the transition.

$$Reset(q_i, q_j, x) = x \tag{5.5}$$

The three-state hybrid system represent the possible taint dynamics of the taint in the PC register and it can be used to represent the taint in the memory locations based on probabilistic predictions on where would the code access the memory location. More details are provided in the future work in Chapter 7.

The variable τ_{I_n} in the case of modeling the taint dynamics in the PC register only is equal to 1. This is because the system is not controlled or regulated by the predictability of the branch predictor. The same differential equation 5.1 is still used to reflect the changes in the taint levels when the controller is added. It can obey an adhoc control law with static values or the value of τ can be changed based on a closed control loop which is computed based on a certain control law that is chosen to provide the most desirable output.

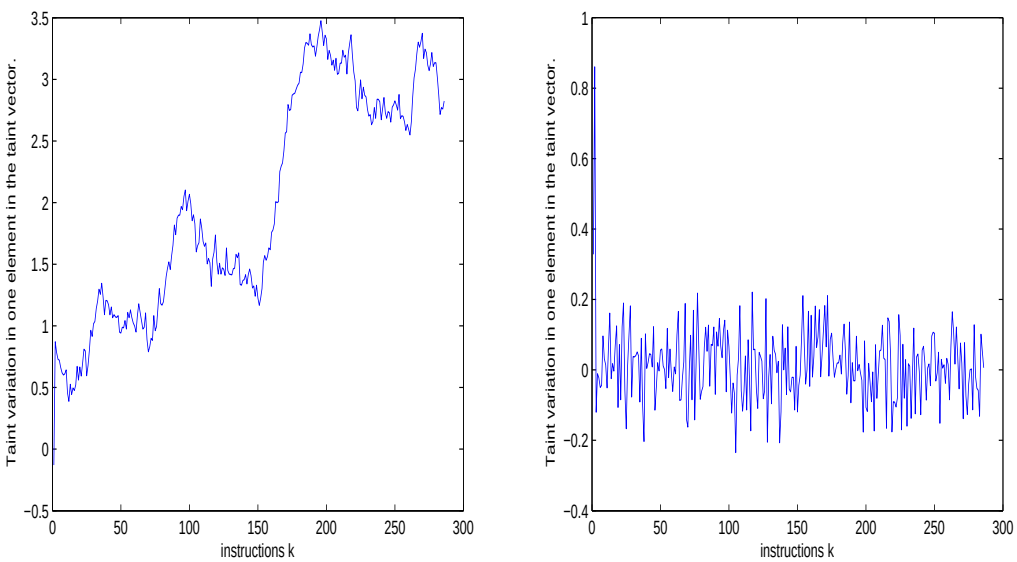
5.2 Validation and Results

The hybrid model is compared with data from the DIFT system which is provided by UNM's Computer Science Department where a code sample is executed and the trace of the taint is saved. The taint is recorded at each instruction including the branch if it is taken or not.

The model is simulated on MATLAB using the ode45 function for solving differential equations and is fed input from the taint dynamics on the DIFT system. The hybrid system discrete events are modeled through events interrupting the each function corresponding to each state. The input was chosen to be the variation of the taint rate in the PC vector given from the real DIFT system. In some experiments, the input of the model was given to be a stochastic random variable with different characteristics in terms of the branch hits and misses, the taint variation of the PC vector, it was used to validate the model and the its limitations.

The simulation of the differential equation 5.1 representing the state of the system depending on the variations in the taint rate update. The hybrid model is highly dependent on random variables which require update after each ode45 function evaluation especially τ_{I_n} and y_{I_n}.

After observing the taint variations in the simulated results from the DIFT system developed at the CS department, the hybrid model is tested after imputing several types of random distributions, after adding the appropriate offset. The parameters of the equation representing the plant is highly random. In contrast to the TCP hybrid system model provided in Chapter 2, the TCP system has more predefined rules where packets are heading and rules in which the flow is regulated. It is easier to model the input and the expected behavior of the flows at the router level.



The Figure below 5.2 is the performance of the hybrid model represented in the equation 5.1 and τ is changed based on a random uniform distribution. The model considers a random number of instructions and takes into consideration that every program is written differently and is translated differently into set of assembly instructions.

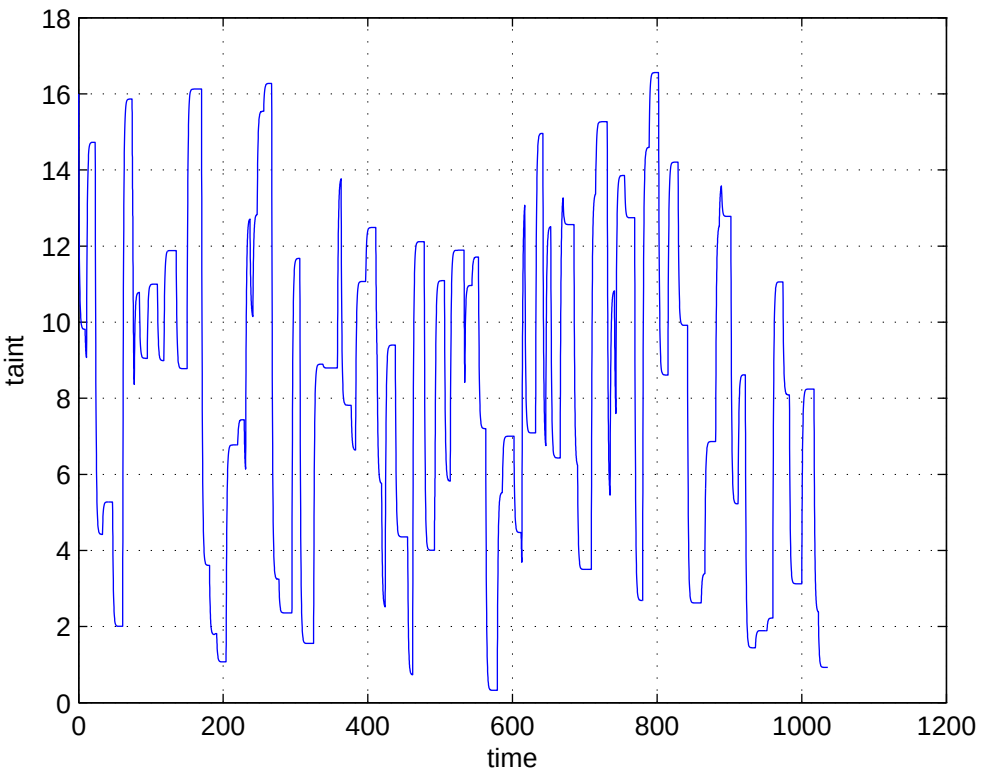


Figure 5.2: MATLAB result of the hybrid system representing the taint rate in a program.

After simulating the model on a strictly random input, the model is tested using data from the original DIFT system. The input was the actual variation of taint from both bubble and selection sort.

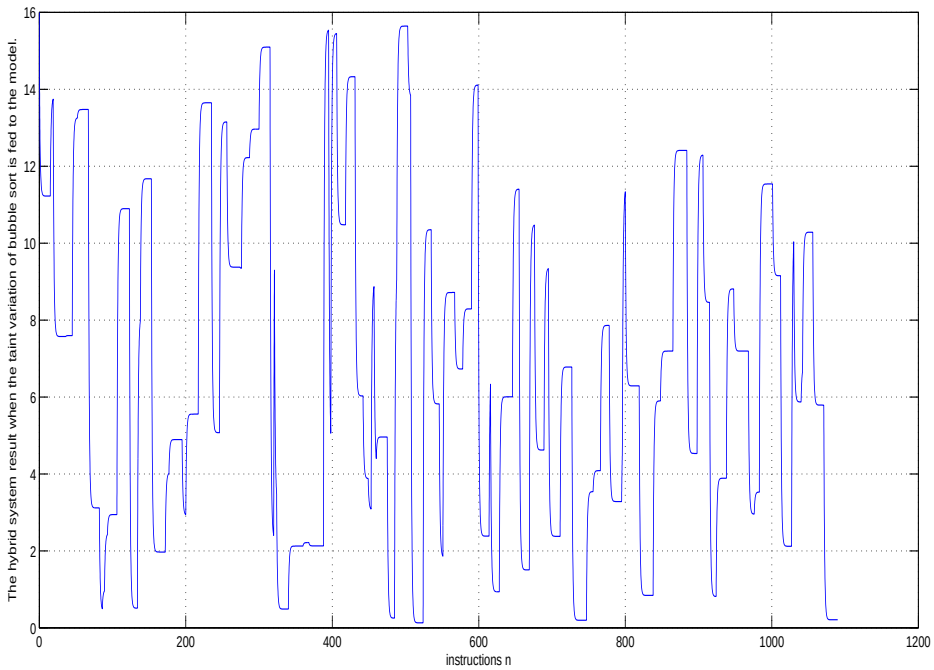


Figure 5.3: MATLAB result of the hybrid system representing the taint rate in a program.

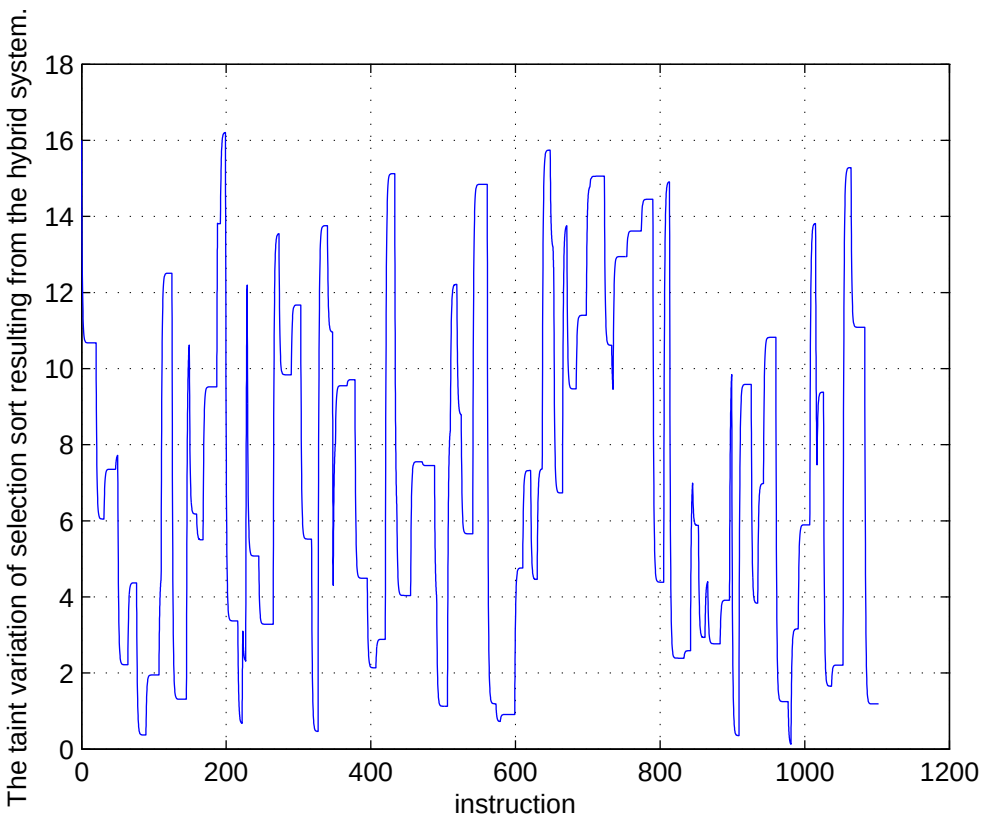


Figure 5.4: MATLAB result of the hybrid system representing the taint rate in a program.

Chapter 6

Model Predictive Control and DIFT

6.1 Model Predictive Control Overview

The model provided above is a time varying hybrid model where its parameters are stochastic and the input of the hybrid system follows few to no rules. In order to optimize the model, we have decided to use model predictor control (MPC) for the purpose of regulating a highly inter-related DIFT system.

MPC is used to tackle multivariable problems which exhibit nonlinearities and difficult dynamic behavior. Model predictive control (MPC) provides an appropriate control approach for taking into account, system dynamics and forecast uncertainties. In addition, it is widely used whenever there are a lot of constraints on the input and the output [16, 20, 26]. The overall objectives of MPC are:

- Prevent violations of input and output constraints.
- Drive some output variables to their optimal set points, while maintaining other

outputs within specified ranges.

- Prevent excessive movement of the input variables.

The behavior of dynamic systems is generally represented as

$$x(t+1) = Ax(t) + Bu(t) \tag{6.1}$$

where t is an instant of time, x is the state of the system and A and B are constants.

There are three components in the MPC approcach: Prediction model, Objective function and Obtaining the control law. The objective of the MPC is to compute a control signal which will satisfy the cost function with the constraints on the input and the output taken into consideration. The MPC algorithm is set to compute an optimal control signal over some period of time. The predicted future output is a function of the current state, feedback, feedfarward and future input adjustment of the controlled system. The control calculations are based on both future predictions and current measurements.

MPC is based on iterative, a finite horizon optimization of a plant model. At a specific point in time t, the system (6.1) is sampled and a cost minimizing control algorithm is applied for a short time T. After a cost minimizing strategy is found for the next T seconds. The plant is sampled again to update the current state the computation of the possible trajectories is done thus the control law is updated based on the predictions done.

The main advantage of MPC is the fact that it allows the current timeslot to be optimized, while keeping future timeslots in account. This is achieved by optimizing a finite time-horizon, but only implementing the current timeslot.

The cost function is given

$$J = \sum_{i=1}^{N} w_{x_i}(r_i - x_i)^2 + \sum w_i \Delta u_i^2 \text{where,} \tag{6.2}$$

x_i = i-th controlled variable,

r_i = i-th reference variable,

u_i = i-th manipulated variable,

w_{x_i} = weighting coefficient reflecting the relative importance of,

w_{u_i} = weighting coefficient penalizing relative big changes in.

6.2 Related Work for MPC

This Section provides a small overview on how the MPC was developed and what are the significant MPC applications and research papers. MPC is not new, Kalman [12] added the algorithm to solve the Riccati Equation in a Linear Quadratic Regulator for finite horizon and extended his finding to cover the infinite-horizon. In the late 1970's, Richalet *et al.* [24] and Cutler and Ramaker [6] emulated the infinite-horizon LQR for constrained processes, marking the beginning of the industrial implementation of what comes to be known as model predictive control (MPC) [23]. Shell Oil started using MPC and it became the most applied control technique in the industry. In the 1980's, the theoretical development of MPC with constraints ran into serious difficulties, and it became increasingly apparent that a return to an infinite-horizon formulation is required to produce stabilizing control law [26].

As mentioned before, the MPC was common in industrial processes, since predicting the future output based on the current state and input of the industrial process is valuable. Qin *et al.* provided a great survey of the MPC progress, from 1970 till

2002. According to the authors in [23], MPC was first developed to meet the specialized control needs of power plants and petroleum refineries, now can be found in a wide variety of application areas including chemicals, food processing, automotive, and aerospace applications. The authors proceed by stating that successful industrial controller for the process industries must therefore maintain the system as close as possible to constraints without violating them. The application oriented industries have used LQR and LQG since it was more feasible for them to obtain accurate models. In addition, although the industry benefited from the MPC and contributed to its variations, researchers also were interested in MPC and they have developed LQR algorithms in which they would take into consideration the errors done in the modeling process of the plant.

Qin *et al.* provided a comparison between the conventional structure of a controlled system and one using MPC. Figure 6.1 provides a hierarchical block diagram of the control flow. The top part is the optimization for the whole plant which computes the optimal steady-state for each unit in the plant. The data can be optimized in local optimizers dedicated to each unit which can run more iterations than the plant-wide optimizer. The optimization unit computes the optimal steady state and then passes it to the dynamic constraint for implementation. The control unit monitors the transitions from one steady state to another while, satisfying the constraints. In the conventional structure, the transitions are accomplished by using a combination of PID algorithms, lead-lag (L/L) blocks and high/low select logic. It is often difficult to translate the control requirements at this level into an appropriate conventional control structure.

In [26], the authors Scokaert *et al.* has provided a huge contribution to LQR to extend it to MPC notion with finite-horizon linear program subjected to inequality constraints in the input and the output, in which it is useful in controlling the hybrid model in the DIFT system, more details in the following Section.

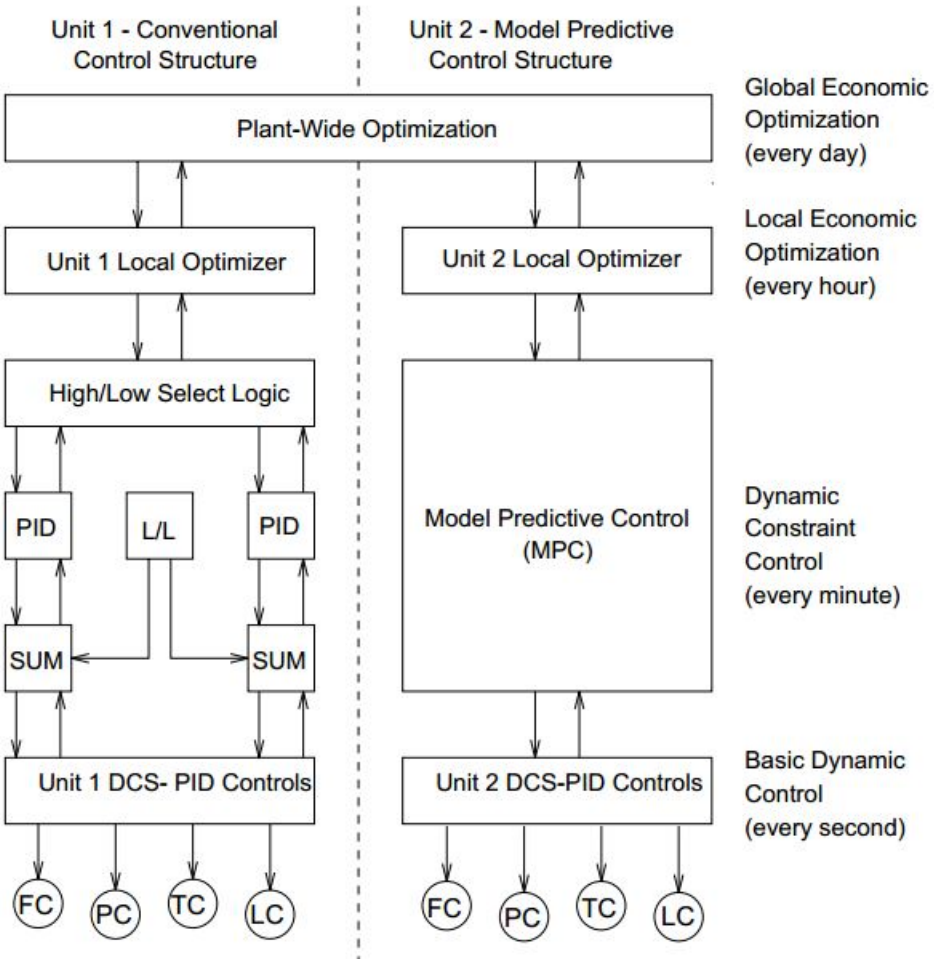


Figure 6.1: Hierarchy of control system functions in a typical processing plant [23].

6.3 Applying MPC for DIFT

The DIFT system has a hybrid model, as seen in the previous chapter, the states are in continuous time and discrete events are occurring and causing the state to alter. MPC is applied to the system after the discretization of the states based on a sampling time Δt.

In [26], consider a discrete time system represented as:

$$x_{t+1} = Ax_t + Bu_t \tag{6.3}$$

where $x_t \in \mathcal{R}^n$ and $u_t \in \mathcal{R}^m$ are the states and input vectors at time t, A and B are the state transition and input distribution matrices. Assuming that (A, B) are stabilizable.

The objective function is defined as:

$$\phi(x_t, \pi) = \sum_{j=0}^{\infty} x'_{j|t} Q x_{j|t} + u'_{j|t} R u_{j|t} \tag{6.4}$$

where $Q \geq 0$ and $R > 0$ are symmetric weighting matrices, such that $(Q^{1/2}; A)$ is detectable, and

$$\pi = \{u_{t|t}, u_{t+1|t}, ...\} \tag{6.5}$$

$$x_{j+1|t} = Ax_{j|t} + Bu_{j|t}, where \qquad t \geq j \tag{6.6}$$

The constraints are defined as

$$Hx_{j+1|t} \geq h, where \qquad t \geq j \tag{6.7}$$

$$Du_{j|t} \geq d \tag{6.8}$$

where $h \in \mathcal{R}_+^{n_h}$ and $d \in \mathcal{R}_+^{n_d}$ dene the constraint levels, with n_h and n_d denoting the number of state and input constraints respectively, and H and D are the state and input constraint distribution matrices.

Let $\mathcal{X}_K \in \mathcal{R}^n$ denotes the set of states x_t for which the unconstrained LQR law, $u_{j|t} = -Kx_{j|t}$ $(t \geq j)$, satisfies 6.7 and 6.6 as defined in [26].

The objective is,

$$\phi_N(x) = min_{\pi\ in\mathcal{P}_{\mathcal{N}}}\phi(x,\pi) \tag{6.9}$$

$$\phi(x,\pi) = \sum_{j=t}^{N-1}(x'_{j|t}Qx_{j|t} + u'_{j|t}Ru_{j|t}) + x'_{t+N|t}\tilde{Q}x_{t+N|t} \tag{6.10}$$

where $\tilde{Q} = Q + K'RK + (A - BK)'\tilde{Q}(A - BK)$

The algorithm of solving for the steady state solution using MPC by implementing infinite-horizon LQR by increasing the N.

1. Choose a finite horizon N_0, set $N = N_0$.

2. Solve 6.10

3. If $x^N_{t+N|t} \in \mathcal{X}_K$ skip to 5

4. increase N, if $N \geq N_max$ then go to 2, terminate and apply MPC.

5. terminate $\pi^*(x) = \pi_N(x)$

Model predictive control techniques include a number of design parameters are as follows: We are considering only finite horizon in the system, thus, $P = M$ the controlled horizon and the predicted horizon are equal.

N	model horizon
Δt	sampling period
P	prediction horizon (number of predictions)
M	control horizon (number of control moves)
Q	weighting matrix for predicted errors
R	weighting matrix for control moves

Table 6.1: Table listing the MPC design variables.

```
mpc1 = mpc(sysd,Ts,p,m);                    % MPC object
mpc1.OV = struct('Min',16,'Max',32);        % Input saturation
    constraints
mpc1.Weights.OV = [sqrt(Qy)];               % Output weights (only on
    original output)
mpc1.Weights.MV = sqrt(Qu);                 % Input weight
mpc1.Weights.MVRate = 1e-5;                 % Very small weight
    on command input increments
ry = 16;                                    % Output set point
mpc1.MV.Target = ry/dcg;                    % Set-point for
    manipulated variable
Y = struct('Weight',[1]);                   % Weight on y(t+p)
U = struct('Weight',sqrt(Qu));              % Weight on u(t+p-1)
setterminal(mpc1,Y,U);                      % Set terminal weight
    y'*y = x'*P*x
setoutdist(mpc1,'remove');                  % Remove additional
    disturbance integrators
mpcgain = dcgain(ss(mpc1));
fprintf('\n(unconstrained) MPC: u(k)=[%8.8g]*x(k)',mpcgain(1));
fprintf('\n                  LQR: u(k)=[%8.8g]*x(k)\n\n',-K(1));
```

The result of the above computation of finite horizon with all the constraints including the output to be restricted between 16 and 32 (check line 2 in the code above).

(unconstrained) MPC: u(k)=[0.74626162]*x(k)

LQR: u(k)=[6.2603555]*x(k)

The output of the system is displayed in Figure 6.2 below.

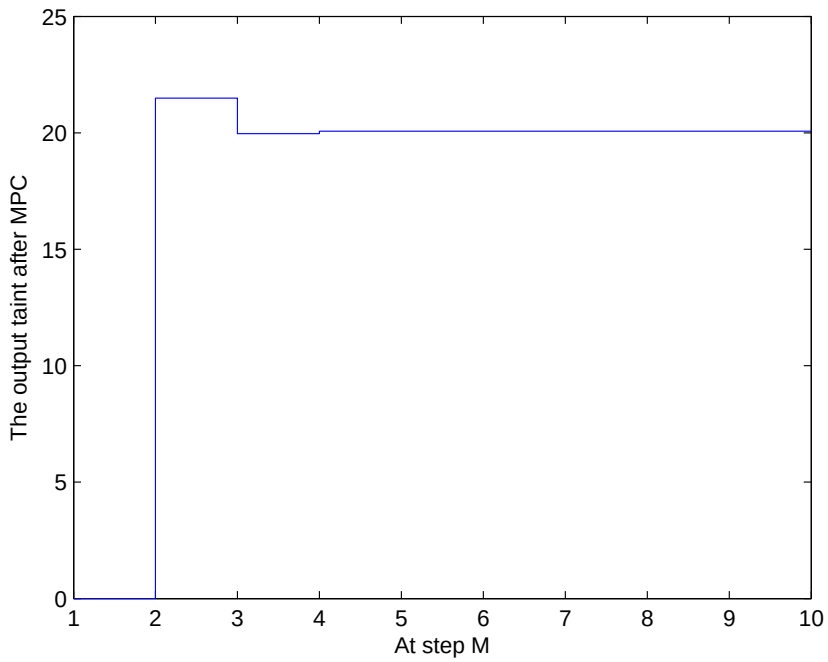


Figure 6.2: Output of the modeled DIFT system.

After simulating the resulting model with the MPC controller computed based on one instance in the DIFT system, in other words, the state us altered each time the system alters its parameters.

Chapter 7

Conclusions

In this thesis, a computing system – Dynamic Information Flow Tracking (DIFT), is studied in order to represent it as hybrid model. The model provided in Chapter 5, represents the taint in the PC register, the hybrid model is built as a combination between a thermostat model and a queue model. The incremental changes are provided before finalizing the model.

The hybrid model captures the dynamics of the taint of the PC register while guarding some rules which defines the state of the system. The guards and rules were defined in terms of the taint in the PC register and the predictability of the branch results. The main purpose of using the data of the branch results is to emphasize that when the information is predictable then less amount of information has propagated thus, the taint can be reduced. On the other hand, when the predictor is encountering a lot of misses in predicting the correct result of the branches, then the taint should be adjusted since the information should be altered thus, more taint is needed to represent the flow of information.

In Chapter 6, this thesis suggests a Model Predictive Controller (MPC) in order to limit the taint in the PC resister, the design figures and rules are set to meet the

constraints of the designed DIFT system. The weights assigned to the input and output were chosen in order to satisfy the constraints. The system is discretized and the MPC controller is added to the system.

7.1 Future Work

The thesis has examined a side of the over-tainting problem that can be applied to the DIFT system in which branch instructions. Another aspect that can be solved through the same technique applied in this thesis, is the over-tainting due to memory access. The memory instructions might over-taint while reading and writing from the memory. The same model can be applied to the memory instructions where the taint can't be cleared easily. The taint was reduced in branch instruction due to the predictor's efficiency in predicting the next instruction, the memory instructions need a similar predictor in order to surf as a taint reducer.

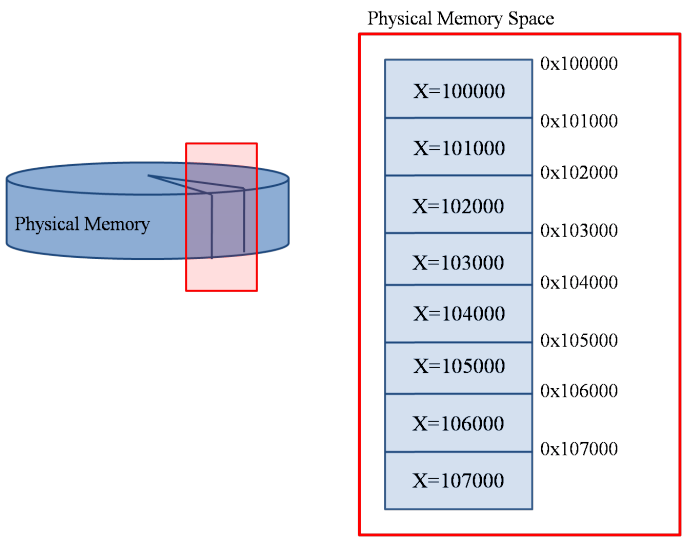


Figure 7.1: Memory address space is represented by a variable X, labeling each section of the physical memory space.

As a suggestion of a possible solution, the physical memory address space is divided into sections where a probability is assigned to it. It can be represented by a variable X $\in \mathcal{Z}^+$. At the beginning of the program X would be a uniform variable with f :X$\rightarrow \mathbb{R}$, where each section of the memory has equal probability to get accessed(read or write). Assuming that the memory is divided to K sections each of size M bytes (where $K, M \in \mathcal{Z}^+$). For instance in Figure 7.1, $K = \{100000, 101000, 102000, ...\}$ where $K = 101000$ represents the memory section from $101000 \rightarrow 101999$ where $M = 10000$.

$$P(X = x) = (1/K) \text{ where } x \in \mathcal{K} \tag{7.1}$$

The probability in 7.1, will be modified when memory is accessed. When a section is accessed it is more likely to be accessed again later. Therefore, when a section of the memory is accessed the taint is adjusted based on the probability assigned to the section. If the probability is high then the taint should be reduced.

The taint is ensured to be in between the bounds set by the DIFT system by using one of the previous controllers applied to the PC taint vector.

Appendices

Appendix A

Graph Model MATLAB Code

```
close all;
clear all;
% Generate a random code depending on 2 types of instruction
dependencies:
% control and arithematic
n = 100;
code = [];
for i = 1:n
var = rand;
number_of_registers = 8;
reg1 = num2str(round(rand(1)*number_of_registers));
reg2 = num2str(round(rand(1)*number_of_registers));
if(rand>0.8)
%branch instruction
new_instruction = ['branc', 'PC','r' reg2, 'ad'];
else
%arithematic instruction
new_instruction = ['arith', 'r' reg1,'r' reg1, 'r',reg2];
```

```
end
code = [code; new_instruction];
end
node_name = [];
A_graph = zeros(n,n);
link = 0;
var=rand(1,100)*100;
plot (var, 'o')
hold on;
count = 0;
for i = 1:n
for j = i:n
if (code(i,1:5)=='arith')
%check of the output is connected to something else and not
%updated
output_to_link = code(i,6:7);
if ((output_to_link == code(j,8:9)))
%|| (output_to_link == code(j,10:11))
A_graph(i,j) = rand; %establish link
if (code(j,1:5)=='branc')
plot([i j],[var(i) var(j)],'-r');
else
plot([i j],[var(i) var(j)],'-b');
end
%A_graph(i,j,2) = link + 1;
if (j == i)
else
if(output_to_link == code(j,6:7)) % the register has
been updated: no more dependency
%link = link + 1;
```

```
j = n;
end
end
end
end
end
10
end
title('Random Program dependency graph: blue edges for arithmetic
dependencies and red for branch dependencies. Edges are
   represented in
circles.')
% Plot the graph
nodes = [];
for i = 1:n
for j = 1:n
if (A_graph(i,j,1) ~= 0)
if (code(j,1:5)=='arith')
nodes= [nodes;1, i, -j ];
else
if (code(j,1:5)=='branc')
nodes= [nodes;0, i, -j ];
end
end
end
end
end
```

References

[1] M. I. Al-Saleh, *Fine-Grained Reasoning About the Security and Usability Trade-off in Modern Security Tools*, Doctoral Dissertation, Albuquerque, NM, USA, 2011.

[2] M. I. Al-Saleh and J. R. Crandall, *On Information Flow for Intrusion Detection: What if Accurate Full-System Dynamic Information Flow Tracking was Possible?* Proceedings of the Workshop on New Security Paradigms, 2010, pp. 17–32.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Springer-Verlag New York, Inc. 2006.

[4] D. J. Cooper, *Proven Methods and Best Practices for Automatic PID Control*, 2006. `http://www.controlguru.com/pages/table.html`, Accessed: June 23, 2013.

[5] J. R. Crandall and F. T. Chong, *MINOS: Control Data Attack Prevention Orthogonal to Memory Model*, Proceedings of the 37th International Symposium on Microarchitecture (MICRO). December, 2004.

[6] C. R. Cutler and B. L. Ramaker, *Dynamic Matrix Control A Computer Control Algorithm* (1980). Proceedings of Joint Automatic Control Conference. San Francisco, USA, January, 1980,

[7] B. C. Dean, *A Simple Expected Running Time Analysis for Randomized "Divide and Conquer" Algorithms*, Discrete Applied Mathematics Journal, 2006, **154**, no. 1, 1 –5.

[8] J. Ding, J. Gillula, H. Huang, M. P. Vitus, W. Zhang, and C. J. Tomlin, *Hybrid Systems in Robotics: Toward Reachability-Based Controller Design*, IEEE Robotics & Automation Magazine, September, 2011, **18**, no. 3, 33 –43.

[9] J.P. Hayes, *A Graph Model for Fault-Tolerant Computing Systems*, IEEE Transactions on Computers, 1976, **C-25**, no. 9, 875–884.

[10] J. He, Q. Wang, and T. Lee, *PI/PID Controller Tuning via LQR Approach*, Proceedings of the 37th IEEE Conference on Decision and Control, 1998, pp. 1177–1182.

[11] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*, John Wiley and Sons, New York, 2004.

[12] R.E. Kalman, *Contributions to the Theory of Optimal Control*, Vol. 5, Boletin De La Sociedad Matematica Mexicana, 1960.

[13] J.O. Kephart and S.R. White, *Directed-graph Epidemiological Models of Computer Viruses*, Proceedings of research in Security and Privacy, Symposium on IEEE Computer Society, 1991, pp. 343–359.

[14] Donald E. Knuth, *The Art of Computer Programming, Sorting and Searching*, 2nd ed. Vol. 3, Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.

[15] J. Lee, S. Bohacek, J. P. Hespanha, and K. Obraczka, *Modeling Communication Networks With Hybrid Systems*, IEEE/ACM Transactions on Networking, 2007, pp. 630 –643.

[16] J. H. Lee, *A Lecture on Model Predictive Control*, Georgia Institute of Technology, 2000. `http://cepac.cheme.cmu.edu/pasilectures/lee/LecturenoteonMPC-JHL.pdf`.

[17] C. Lu, T.F. Abdelzaber, J.A. Stankovic, and S.H. Son, *A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers*, Proceedings of the Symposium of on Real-Time Technology and Applications, 2001, pp. 51–62.

[18] I. Lu, T. F. Abdelzaher, and G. Tao, *Direct Adaptive Control of a Web Cache System*, Proceedings of the American Control Conference, 2003, pp. 1625 –1630.

[19] J. Newsome and D. Song, *Dynamic Taint Analysis for Aautomatic Detection, Analysis and Signature Generation of Exploits on Commodity Software*, Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS '05), 2005.

[20] P.E. Orukpe, *Basics of Model Predictive Control*, Imperial College London, 2005. `http://citeseer.uark.edu:8080/citeseerx`.

[21] D. A. Patterson, *Computer Organization and Design, Fourth Edition: The Hardware/-Software Interface*, 4th ed. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2008.

[22] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1990.

[23] S. Qin and T. Badgwell, *A Survey of Industrial Model Predictive Control Technology*, Control Engineering Practice, July, 2003 **11**, no. 7, 733–764.

[24] J. Richalet, A. Rault, J.L. Testud, and J. Papon, *Model Predictive Heuristic Control: Applications to Industrial Processes*, Automatica, 1978, **14**, no. 5, 413 –428.

[25] A. J. Schaft and J. M. Schumacher, *Introduction to Hybrid Dynamical Systems*, Springer-Verlag, London, UK, 1999.

[26] P. O. Scokaert and J.B. Rawlings, *Constrained Linear Quadratic Regulation*, IEEE Transactions on Automatic Control, 1998 **43**, no. 8, 1163–1169.

[27] V. Sharma, A. Thomas, T.F. Abdelzaher, and K. Skadron, *Power-aware QOS Management in Web Servers*, In Proceedings of the 24 th IEEE Real-Time Systems Symposium (RTSS'03), Cancun, 2003, pp. 63–72.

[28] G. E. Suh, J. Lee, and S. Devadas, *Secure Program Execution via Dynamic Information Flow Tracking*, Proceedings of ASPLOS-XI, 2004.

[29] A. Turevskiy, *System identification toolbox*, 2013. `http://www.mathworks.com/products/sysid/index.html`, Accessed: June 23, 2013.

[30] N. Xi, T. Tarn, and A.K. Bejczy, *Intelligent Planning and Control for Multirobot Coordination: An Event-based Approach*, IEEE Transactions on Robotics and Automation, 1996 **12**, no. 3, 439–452.