Volume 13 • Issue 6 • December 2019

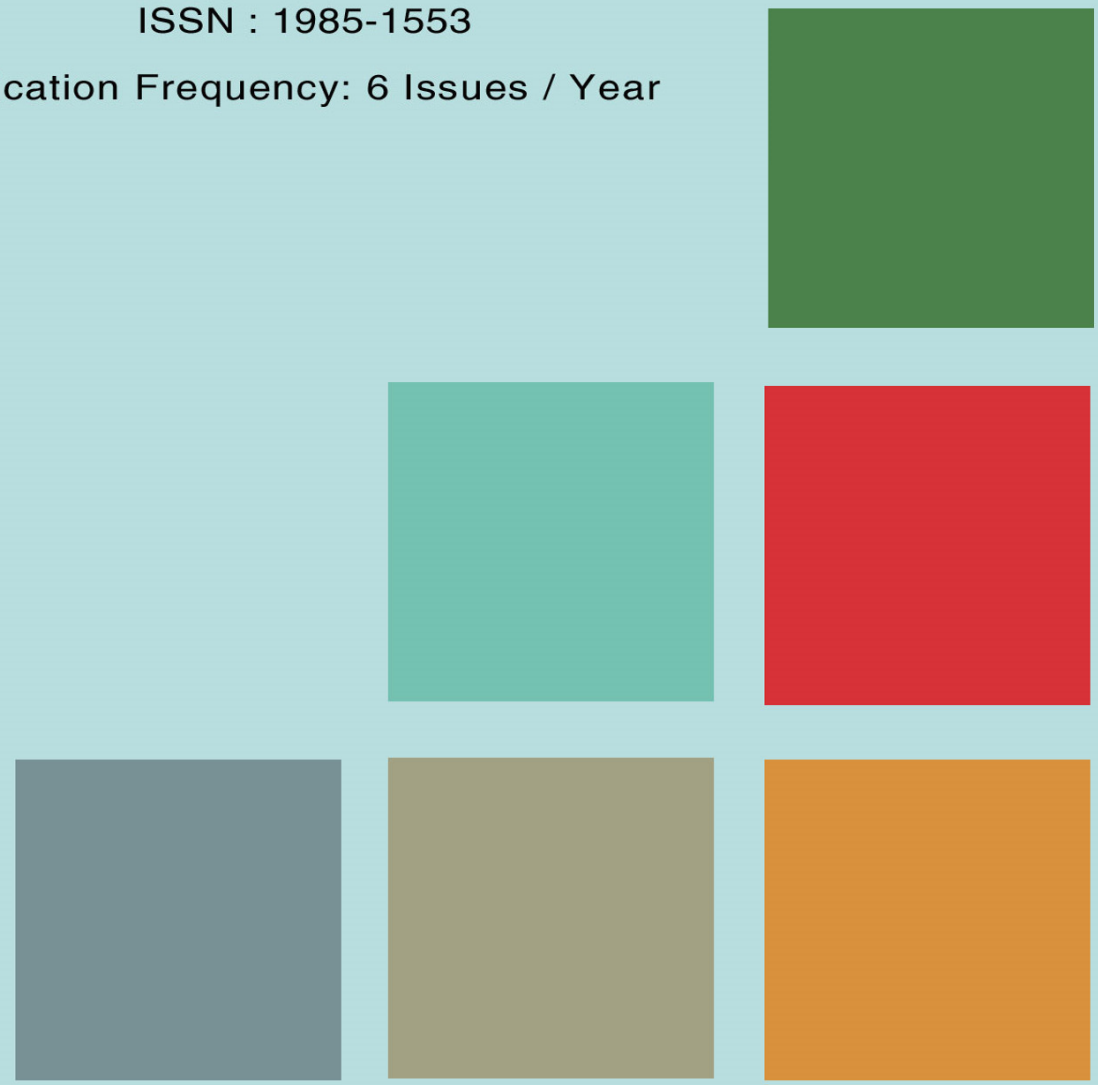
Editor-in-Chief
Dr. Chen-Chi Shing

INTERNATIONAL JOURNAL OF

COMPUTER SCIENCE AND SECURITY (IJCSS)

ISSN : 1985-1553

Publication Frequency: 6 Issues / Year



CSC PUBLISHERS
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND SECURITY (IJCSS)

VOLUME 13, ISSUE 6, 2019

EDITED BY
DR. NABEEL TAHIR

ISSN (Online): 1985-1553

International Journal of Computer Science and Security is published both in traditional paper form and in Internet. This journal is published at the website http://www.cscjournals.org, maintained by Computer Science Journals (CSC Journals), Malaysia.

IJCSS Journal is a part of CSC Publishers
Computer Science Journals
http://www.cscjournals.org

INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND SECURITY (IJCSS)

Book: Volume 13, Issue 6, December 2019
Publishing Date: 31-12-2019
ISSN (Online): 1985 -1553

IJCSS Journal is a part of CSC Publishers
http://www.cscjournals.org

Published in Malaysia

Typesetting: Camera-ready by author, data conversation by CSC Publishing Services – CSC Journals, Malaysia

CSC Publishers, 2019

EDITORIAL PREFACE

This is *Sixth* Issue of Volume *Thirteen* of the International Journal of Computer Science and Security (IJCSS). IJCSS is an International refereed journal for publication of current research in computer science and computer security technologies. IJCSS publishes research papers dealing primarily with the technological aspects of computer science in general and computer security in particular. Publications of IJCSS are beneficial for researchers, academics, scholars, advanced students, practitioners, and those seeking an update on current experience, state of the art research theories and future prospects in relation to computer science in general but specific to computer security studies. Some important topics cover by IJCSS are databases, electronic commerce, multimedia, bioinformatics, signal processing, image processing, access control, computer security, cryptography, communications and data security, etc.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 14, 2020, IJCSS appears with more focused issues. Besides normal publications, IJCSS intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

This journal publishes new dissertations and state of the art research to target its readership that not only includes researchers, industrialists and scientist but also advanced students and practitioners. The aim of IJCSS is to publish research which is not only technically proficient, but contains innovation or information for our international readers. In order to position IJCSS as one of the top International journal in computer science and security, a group of highly valuable and senior International scholars are serving its Editorial Board who ensures that each issue must publish qualitative research articles from International research communities relevant to Computer science and security fields.

IJCSS editors understand that how much it is important for authors and researchers to have their work published with a minimum delay after submission of their papers. They also strongly believe that the direct communication between the editors and authors are important for the welfare, quality and wellbeing of the Journal and its readers. Therefore, all activities from paper submission to paper publication are controlled through electronic systems that include electronic submission, editorial panel and review system that ensures rapid decision with least delays in the publication processes.

To build its international reputation, we are disseminating the publication information through Google Books, Google Scholar, Directory of Open Access Journals (DOAJ), Open J Gate, ScientificCommons, Docstoc and many more. Our International Editors are working on establishing ISI listing and a good impact factor for IJCSS. We would like to remind you that the success of our journal depends directly on the number of quality articles submitted for review. Accordingly, we would like to request your participation by submitting quality manuscripts for review and encouraging your colleagues to submit quality manuscripts for review. One of the great benefits we can provide to our prospective authors is the mentoring nature of our review process. IJCSS provides authors with high quality, helpful reviews that are shaped to assist authors in improving their manuscripts.

Editorial Board Members
International Journal of Computer Science and Security (IJCSS)

EDITORIAL BOARD

TABLE OF CONTENTS

Volume 13, Issue 6, December 2019

Pages

IOT Power Management For Reducing The Dependency On Batteries

Ahmed Abdulmanea *ahmed_abdulmanea@adeneng-faculty.edu.ye*
Faculty of Engineering / University of Aden / Information Technology
Aden, Yemen

Lutfi Khanbari *llkhanbari@gmail.com*
Faculty of Engineering / University of Aden / Computer science and Engineering
Aden, Yemen

Abstract

Various reports and studies projects that, by 2020, about to 50 billion devices will be connect to internet of things and the global market value will reach $7.1 trillion, which will give the engineers the opportunity to design solutions to several problems such as healthcare, industry, transportation, agriculture, smart homes, etc.

The Internet of Things (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.

Sensors are the core of the IoT as its collect the data from the environment and then exchange it with a web cloud server through the network (internet) and then send a response to the things (devices) to take actions.

Most of the devices will be connect wirelessly due to the inconvenience, expense or infeasibility of wiring it, and many of them have size constrains with limited battery space and no power cord, so powering these devices (to achieve several months of functioning) become serious challenge. This paper highlights focusing in this challenge and addressing some solutions by using environmental energy to make IoT self-powered such as solar energy, these will decrease and, in some cases, eliminating their dependence on batteries.

Keywords: Internet of Things, Wireless Power Management, Energy Harvesting, Low Power, Solar Energy.

1. INTRODUCTION

By 2020, there will be around 50 billion smart objects connected to the Internet of Things (more than six times the world's projected population at the time), making the IoT one of the fastest growing technology across all of computing [24]. These smart devices will change all aspects of our daily lives and fundamentally change the way we interact with our physical environment, thereby revolutionizing a number of application domains such as telemetry, healthcare, home automation, energy conservation, security, wearable computing, asset tracking, maintenance of public infrastructure, etc., as shown in Figure 1.

One of the biggest challenges to realizing this IoT vision is the problem of powering these tens of millions of IoT devices. Most of these devices will be battery-powered for reasons of cost, convenience, or the need for untethered operation. Despite tight constraints on size and, hence, battery capacity, many IoT devices will be required to have long operational lifetimes (from a few

months to possibly several years) without the need for battery replacement, because frequent battery replacement at scale is not only expensive, but often not even feasible in addition to the Alkali effect of the battery on the environment. For example, the Environment Protection Agency reports that more than 3 billion batteries are discarded in the USA every year and that, placed end to end, discarded AA batteries would circle the earth six times. The rapid proliferation of IoT devices will only exacerbate this problem, making the need to address it an urgent priority.

This paper highlights some promising directions for addressing this challenge and makes a case for focusing on two main building blocks: (a) the development of intelligent system-level power management techniques that allow an IoT device to adjust its power consumption in a context-aware manner, and (b) the use of environmental energy harvesting to make IoT devices self-powered, thus decreasing in some cases, even eliminating their dependence on batteries. These building blocks are illustrated using examples of IoT devices, including the QUBE wireless platform, which exploits the characteristics of emerging non-volatile memory technologies to seamlessly and efficiently enable long-running computations in systems that have an intermittent and unreliable power supply.

It is important to recognize that IoT devices have very diverse power requirements and longevity requirements, which have a profound influence on how they are designed. One group of devices, henceforth referred to as Type I devices, are wearable devices.

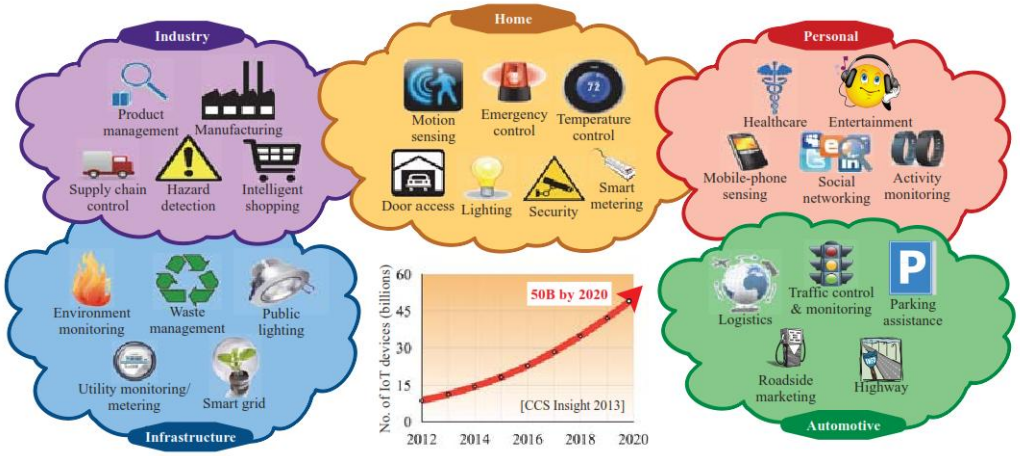


FIGURE 1: A summary of the envisioned applications and growth application for the Internet of Things.

(e.g., smart watches, fitness monitors, connected glasses), which have a longevity requirement of several days because a user is likely to own only a few such devices and can recharge them regularly, particularly with the advent of wireless charging technologies. A second group of devices, henceforth referred to as Type II devices, are set-and-forget devices (e.g., home security and automation sensors, water leak sensors) that a user wants to deploy and then not tinker with for several (2 to 5) years. A user is likely to own dozens of such devices, therefore frequent battery replacement would be very inconvenient and hamper the user experience. A third group of devices, henceforth referred to as Type III devices, are semi-permanent devices (e.g., wireless sensors that monitor public infrastructure such as bridges, highways, and parking structures), where the device is installed and needs to operate for more than a decade. The scale of these devices makes frequent battery replacement simply infeasible. A fourth group of devices, henceforth referred to as Type IV devices, are battery less and passively powered (e.g., RFID tags, smartcards), drawing their power from an external source such as a tag reader. Finally, a fifth group of devices, henceforth referred to as Type V devices, are powered appliances (e.g., smart refrigerators, microwaves, smart TVs) that will always be plugged into a power outlet, eliminating the need for a battery.

2. SELF-POWERED SYSTEMS USING ENERGY HARVESTING

Over the past, energy harvesting has emerged as an attractive and increasingly feasible option to address the power supply challenge in a variety low power systems. The use of energy harvesting significantly prolongs over all system lifetime and has the potential to result in self powered, perpetual system operation, particularly for Type II and Type III IoT devices. Figure 2 shows the power supply subsystem of an energy harvesting device. In this section, we discuss recent advances in the design of each constituent component, namely, the energy harvester (or transducer), the power conditioning unit, and the energy storage element.

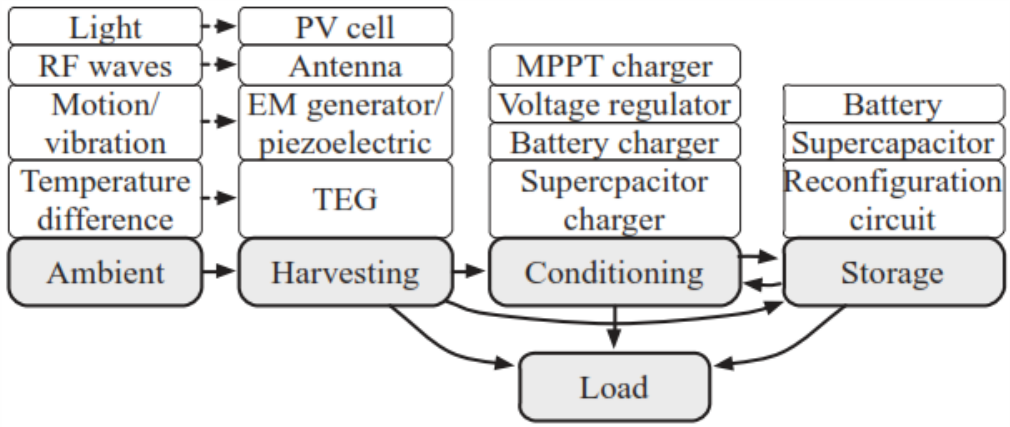


FIGURE 2: The power supply subsystem of an energy harvesting IoT device.

2.1 Harvesting Ambient Energy

An energy harvester, in our context, is a device that converts power from ambient sources, such as electromagnetic radiation (including light and RF waves), thermal gradients, mechanical motion, etc., into electrical power. Of these modalities, solar energy harvesting through photovoltaic conversion is the most mature and well-studied, in part because it has a higher power density (output power per unit area or volume) than other ambient power sources. Solar harvesting is well suited for IoT devices that have substantial exposure to light, such as the Flood Beacon [5], which is an outdoor environment monitor. Flexible photovoltaic cells [34] could possibly also be integrated into clothing and used to recharge wearable IoT devices.

Kinetic energy harvesting converts the mechanical energy of motion or vibration into electrical energy through electromagnetic induction [28] or the piezoelectric effect [39]. It is particularly attractive for wearable IoT devices that are powered by human motion and for devices attached to vibrating objects such as engines or motors. For example, the Pavegen [6] is an energy harvesting floor tile that can be installed on a sidewalk to gather energy from footsteps, which could be used for advertising, way finding solutions, etc. Intelligently scavenging energy from routine human activities could play a prominent role in improving the battery lifetime of IoT devices. RF energy harvesting uses the power received from incident RF waves for powering a device. This technique is commonly used in passive RFID systems. The source of the power can either be dedicated RF waves generated for wireless charging (e.g., the Qi wireless charging standard) [31], or ambient RF signals that are transmitted for wireless data transfer (e.g., WiFi or TV signals) [14]. Energy harvesting from ambient WiFi signals has been demonstrated [30], although the amount of harvested power that can be harvested is often minuscule.

Thermoelectric generators (TEGs) translate a thermal gradient between two surfaces into an electrical potential [51]. TEGs are suitable for powering IoT devices that are in contact with hot surfaces. Wearable IoT devices, such as smartwatches, can also use TEGs as a power source by exploiting the difference between the body's surface temperature and the ambient temperature.

In summary, the choice of harvesting modality for a particular IoT device is dependent on its operating environment, form factor constraints, as well as its power budget.

2.2 Power Conditioning

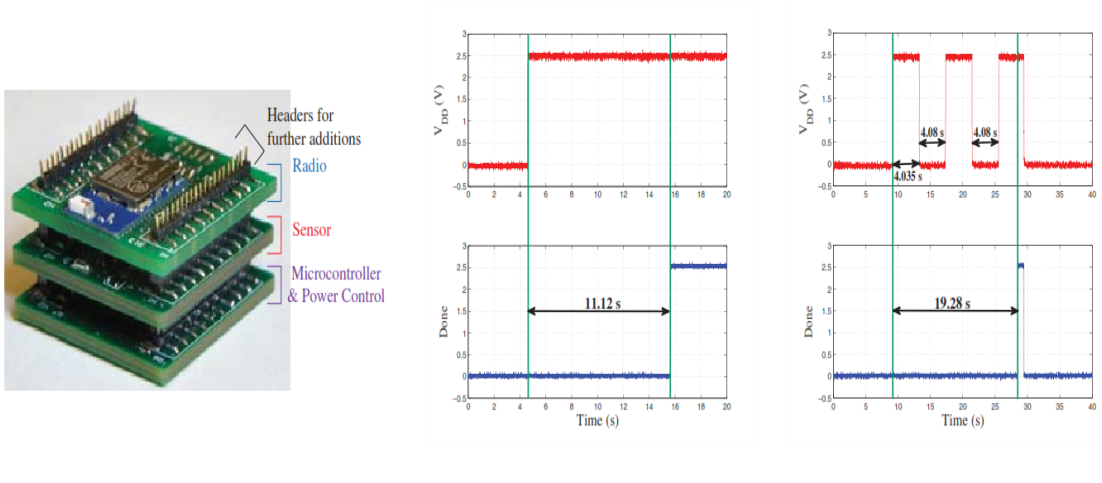
Electronic circuit components require a stable DC power supply to operate reliably. However, the output voltage of an energy harvester often varies significantly depending on the strength of the ambient power source (e.g., the light intensity or the amplitude of vibration). Therefore, the output of the harvester needs to be converted into an appropriate (and stable) voltage level through the use of a power conditioning circuit before it can be fed to an IoT device or transferred to an energy storage element. However, power conditioning for energy harvesting is not straightforward. For example, due to the stringent form factor constraint in most IoT devices, the output power of the harvester is very small, often only a few mW. The conditioning circuit should deliver as much of this power as possible to the IoT device with minimal loss, which requires extremely careful design. Further, some harvesters generate only tens of mV at their output, such as TEGs in body worn devices. In such cases, a boost regulator that accepts an ultra low input voltage is required [19].

In addition to voltage regulation, power conditioning also plays an important role in maximizing harvesting efficiency. Most energy harvesters have an optimal operating point (called the maximum power point or MPP) at which their power output is maximized. Since the MPP changes dynamically based on ambient conditions, the power conditioning unit should continuously maintain operation at the MPP, a process referred to as MPP tracking. MPP tracking is a feature available in many commercial power conditioning ICs [55, 38]. Design considerations for MPP tracking are described in [42, 36]. In [57], MPP tracking is done by modulating the average power consumption of the device, without a dedicated power conditioning unit.

2.3 Energy Storage

Since the amount of power available from an energy harvester is dynamic and unpredictable, an energy storage element is needed in IoT devices for uninterrupted operation when ambient power is not available. Often, the energy storage element is the bulkiest part of an embedded system. Therefore, energy storage elements with a high energy density are highly desirable for IoT devices to maximize lifetime and minimize device size.

Batteries are the most widely used energy storage element in untethered devices. A solid state thin film battery that uses solid electrolytes is a promising battery technology for IoT devices [47]. It has low power density but high energy density, making it suitable for long lasting low power IoT devices. Such a thin, bendable battery can also be easily integrated into small IoT devices [27]. A solid state battery can be manufactured in conventional IC packages or even be integrated with an IC in a single package, such as Cymbet's EnerChip [22]. This enables a significant reduction in size and system integration cost. Compared to batteries, super capacitors have a much higher cycle efficiency and extremely long cycle life. However, they require the power conditioning unit to be able to cope with their large voltage variation, in particular, the very low voltage during cold boot. Dynamic reconfiguration of multiple supercapacitors can mitigate the voltage variation issue and improve cold boot speed [20].



(a) (b) (c)

FIGURE 3: (a) Qube: A modular embedded platform (1" by 1") that facilitates easy prototyping and addition/removal of features through modules, (b) Time taken (11.12s) to complete RSA encryption of 128 characters on Qube in the presence of continuous power supply. The Done signal is raised at the end of the computation, and (c) QuickRecall implemented on Qube. RSA encryption is successfully performed across multiple power cycles with negligible overhead (19.28s − 2 × 4.08s = 11.12s).

Recent advances in nanotechnology have also enabled flexible supercapacitors on a thin film substrate, which are well suited for wearable applications [44].

3. LOW POWER HARDWARE FOR THE IOT DEVICES

The most effective way to improve the battery life of an IoT device is to decrease the power consumed by its constituent hardware components. Even in IoT devices such as Driblet [3] and SPAN [10] that are powered through energy harvesting (discussed in Section 2), it is imperative to use low power hardware to achieve near perpetual operation. It is useful to note that many IoT devices are architecturally similar to wireless sensor node platforms [25, 45] and low power design techniques used for these platforms are equally applicable to the design of IoT devices [21, 49]. The following subsections discuss recent advances in low power hardware for the computation and communication subsystems of an IoT device, respectively.

3.1 Computation Subsystem

Microcontrollers (MCUs) are at the heart of every embedded system that interfaces to (and interacts with) the real world, including IoT devices. As described in Section 1, many of these systems need to operate unattended for several years without the need for battery replacement [43,46]. Achieving such long operational lifetime requires extreme levels of energy efficiency. Fortunately, many sensing applications operate in a heavily duty cycled mode, wherein the system is active only for very short bursts of time (of ten, only milliseconds) separated by long idle intervals (of ten, many tens of seconds) during which the system can be placed in a low power, sleep mode. Since the system spends greater than 90% of its time in the sleep mode, the cumulative energy spent in this mode is often the bottleneck for battery lifetime. Therefore, it is important to select an MCU that has a very low power consumption in idle state in addition to being power efficient during active computation. To minimize idle mode power consumption, most MCUs feature multiple low power (or sleep) modes. For example, the STM32L1 series of MCUs (based on the ARM Cortex M3 core) supports up to 7 different sleep modes. The sleep modes found in MCUs are of two types. The first is a shallow sleep mode, in which the MCU core is stopped, peripherals are disabled, and clock sources are turned off. However, the MCU stays powered up, which means that state information (consisting of the MCU registers and the contents of on chip SRAM) is preserved during sleep. Although waking up from shallow sleep is very fast, it is (as expected) not the lowest power sleep mode possible. Hypnos [33] addresses

this problem based on the observation that the minimum voltage required for SRAM data retention is often much lower (by as much as 10x) than the minimum operating voltage of the MCU. By lowering the supply voltage when the MCU is in sleep mode to just above the SRAM data retention voltage, Hypnos achieves dramatic reductions in sleep mode power. The second type of sleep mode is deep sleep, in which the entire MCU, including the on-chip SRAM, is powered down. While this results in the lowest power consumption possible during sleep, it does not preserve SRAM state. Therefore, the contents of the SRAM need to be saved to non-volatile storage such as the on-chip Flash of the MCU before entering this mode. When the MCU wakes up next, the saved state is restored from the Flash to the SRAM and the MCU resumes execution. Unfortunately, due to the high erase/write time and power of Flash, the energy overhead of saving and restoring state is substantial. Recent work [32] to address this problem uses emerging non-volatile memory (NVM) technologies such as magnetoresistive RAM.

Processors and MCUs (Freq = 8 MHz)				**Wireless Standards**				**Sensors**			
Product	Architecture Family	Current Active (mA)	Current Sleep (µA)	Standard (Product)	Current Tx (mA)	Current Rx (mA)	Current Sleep (µA)	Sensor	Product	Current Active (µA)	Current Sleep (µA)
MSP430F5438A	MSP430	1.84	0.1	WiFi (TI CC3200)	229	59	4	Temperature	TMP102	85	0.5
STM32L051x6	ARM CM0+	1.55	0.29					Humidity	SHT21	300	0.15
STM32L100C6	ARM CM3	2.16	0.3	IEEE 802.15.4 (Atmel AT86RF231)	14	12.3	0.02	Accelerometer	ADXL362	13	0.01
SAM4S	ARM CM4	4.5	1.8					Light	ISL29033	65	0.01
PIC24FJ128GC010	PIC	1.5	0.075	Bluetooth Smart (Nordic nRF8001)	12.7	14.6	0.5	Proximity	AD7150	100	1

TABLE 1: Power consumption of a few representative hardware components used in IoT devices (sourced from datasheets).

(MRAM) [37] or ferroelectric RAM (FRAM) [26]. These memories combine the flexibility and endurance of SRAM with the non volatility of Flash, all at a very low power consumption. Low power MCUs with these emerging NVMs integrated are already available [48, 61]. In these MCUs, software can save the processor state and the contents of SRAM to the NVM before the MCU enters sleep mode, avoiding the need for keeping the SRAM powered during sleep. Building on this idea, recent research has led to the emergence of a new class of processors called non volatile processors [35, 53]. In these processors, NVM memory elements are distributed throughout the MCU such that it can automatically save the contents of all the registers in these NVM elements before it is shutdown, resulting in a (nearly) zero power sleep mode with state retention and rapid wakeup.

Minimizing power consumption in active mode has been extensively investigated for the past few decades and numerous techniques such as dynamic voltage and frequency scaling (DVFS), voltage islands, etc., have been proposed and shown to be effective in reducing power consumption. Continued voltage scaling has led to the emergence of near threshold and subthreshold processors [17, 58] that aim to operate at an optimal energy point. For example, the Phoenix processor [29] is an event-driven subthreshold processor that has an sleep power consumption of only 30 pW. The use of such ultra low power MCUs, if applicable, will provide a significant boost to the battery life of IoT devices.

Table 1 shows the active mode and sleep mode power consumption of a few off-the-shelf hardware components (including MCUs, radios, and sensors) that are commonly used in IoT devices. As seen, most of these hardware components feature highly power-efficient sleep modes in which the power consumption is decreased by several orders of magnitude compared to the active mode.

3.2 Communication Subsystem

The IoT concept fundamentally depends on the fact that devices will communicate either directly with each other or with a cloud based service accessible through the Internet. Hence, reliable wireless communication is an integral component of any IoT device. Typically, wireless communication is more power hungry than other tasks such as sensing or computation. In addition, different types of IoT devices have different communication requirements depending on their deployment locations, longevity constraints, traffic patterns, etc. Therefore, choosing an appropriate wireless technology that is power effcient is a vital design choice.

Despite its relatively high power consumption, WiFi is the preferred wireless standard for many IoT applications due to its near ubiquitous nature WiFi hotspots are present in most homes, offices, and public spaces and the fact that it enables convenient and straightforward access to the Internet. Advances in wireless communication have also seen the development of numerous low power wireless standards such as Bluetooth Smart, IEEE 802.15.4, etc. The IEEE 802.15.4 standard targets low data rate applications (e.g., remote monitoring and control systems) and defines the physical and medium access control layers upon which the Zigbee and 6LoWPAN network stacks are built. The standard allows for multi-hop wireless topologies and several power efficient IEEE 802.15.4 compliant radios are commercially available. However, one disadvantage of using IEEE 802.15.4 for IoT applications, compared to WiFi, is the need for an additional gateway device to achieve Internet access (if required). Particularly for Type II IoT devices, it is difficult to converge on the use of a single wireless standard due to the varying nature of applications as well as the large number of product vendors involved. Hence, it is likely that future smart homes will use IoT hubs such as Revolve [9] or Ninja Spheramid [11] that support a variety of wireless standards such as WiFi, Bluetooth Smart, Zigbee, Z-Wave, Insteon, etc. In addition to existing wireless standards, innovative approaches such as using the existing power line wiring in the home as an antenna have also been proposed [12].

Bluetooth Smart is an enhanced version of the well known Bluetooth standard that was designed for low power communication [16]. Bluetooth based IoT devices, such as Estimote Beacon [23], Lively [41], tado Cooling [56], etc., can directly communicate with smart phones, which are already Bluetooth equipped. This is a key advantage that will likely cement Bluetooth Smart's position as the wireless standard of choice for IoT devices that need to frequently communicate with mobile devices such as smart phones and tablets.

Other IoT applications such as manufacturing and asset tracking could use RFID based communication. Passive RFID technology allows devices such as battery less smart tags to operate using power harvested from a nearby reader's RF transmissions. Recent work [40] proposed the idea of ambient backscatter, a novel technique that allows two battery less devices to communicate with each other by backscattering existing wireless signals from TV stations and cellular transmissions. Although the technique is mainly intended for low throughput applications, it is a significant step forward because it enables tiny IoT devices to exchange small amounts of information without the need for a battery or a nearby RFID reader.

4. CONCLUSION

This paper showed some guidance to address the problem of powering the devices that form the IoT. We believe that a comprehensive solution to this problem involves two main building blocks including intelligent system-level power management techniques and (perhaps, most promising) is to make IoT devices self powered by harvesting energy from their operating environment. Doing so raises the possibility of perpetual operation of these devices, thus decreasing their dependence on batteries and the need for frequent battery replacement.

5. REFERENCES

[1] Belkin Wemo. http://www.belkin.com/us/Products/homeautomation/c/wemohome automation/.

[2] CubeSensors. https://cubesensors.com/.

[3] Driblet. http://driblet.co/.

[4] Fitbit. http://www.fitbit.com/.

[5] Flood Beacon. http://floodbeacon.com.

[6] Pavegen. http://www.pavegen.com/.

[7] Pebble. https://getpebble.com/.

[8] Quirky Wink. https://www.quirky.com/ge.

[9] Revolv Home Automation hub. http://revolv.com/.

[10] Self-Powered Ad-Hoc Network. http://www.lockheedmartin.com/us/products/span.html.

[11] Spheramid Gateway for Ninjasphere. http://ninjablocks.com/.

[12] Wally. https://www.wallyhome.com/.

[13] Wireless Sensor Tags. https://www.mytaglist.com/.

[14] B. Allen et al. Harvesting energy from ambient radio signals: A load of hot air? In LAPC, pages 1–4, 2012.

[15] Ambiq Micro. AM08X5 real-time clock family. http://ambiqmicro. com/sites/default/files/AM08X5 Data Sheet DS0002V1p1.pdf .

[16] Bluetooth Special Interest Group. https://www.bluetooth.org/.

[17] D. Bol et al. SleepWalker: A 25-MHz 0.4-V sub- mm2 7 − µW/MHz microcontroller in 65-nm LP/GP PCMOS for low-carbon wireless sensor nodes. IEEE J SOLID-ST CIRC, pages 20–32, 2013.

[18] C. Brown. Low-power sampling techniques using kinetis l, 2013.

[19] E. Carlson et al. A 20 mv input boost converter with efficient digital control for thermoelectric energy harvesting. IEEE J SOLID-ST CIRC, pages 741–750, 2010.

[20] C.-Y. Chen and P. H. Chou. Duracap: A super capacitor-based, power-bootstrapping, maximum power point tracking energy-harvesting system. In ISLPED, pages 313–318, 2010.

[21] G. Chen et al. Circuit design advances for wireless sensing applications. Proc. IEEE, pages 1808–1827, 2010.

[22] Cymbet. EnerChip. http://www.cymbet.com/.

[23] Estimote. Estimote beacons. http://estimote.com/.

[24] D. Evans. The internet of things: How the next evolution of the internet is changing everything. http://www.cisco.com/web/about/ ac79/docs/innov/IoT IBSG 0411FINAL.pdf , 2011.

[25] M. Fo jtik et al. A millimeter-scale energy-autonomous sensor system with stacked battery and solar cells. IEEE J SOLID-ST CIRC, pages 801–813, 2013.

[26] G. R. Fox et al. Current and future ferroelectric nonvolatile memory technology. J VAC SCI TECHNOL B, pages 1967–1971, 2001.

[27] M. Gorlatova et al. Energy harvesting active networked tags (EnHANTs) for ubiquitous ob ject networking. IEEE WC, pages 18–25, 2010.

[28] M. Gorlatova et al. Movers and shakers: Kinetic energy harvesting for the internet of things. In (to appear in) ACM SIGMETRICS, 2014.

[29] S. Hanson et al. A low-voltage processor for sensing applications with picowatt standby mode. IEEE J SOLID-ST CIRC, pages 1145–1155,2009.

[30] A. M. Hawkes et al. A microwave metamaterial with integrated power harvesting functionality. Applied Physics Letters, 103(16), 2013.

[31] H. Jabbar et al. RF energy harvesting system and circuits for charging of mobile devices. IEEE T CONSUM ELECTR, pages 247–253, 2010.

[32] H. Jayakumar et al. QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. In VLSID, pages 330–335, 2014.

[33] H. Jayakumar et al. HYPNOS: An Ultra-Low Power Sleep Mode with SRAM Data Retention for Embedded Microcontrollers. CODES+ISSS '14, 2014 (to appear).

[34] C. Y. Jiang et al. High-bendability flexible dye-sensitized solar cell with a nanoparticle-modified ZnO-nanowire electrode. APPL PHYS LETT, 2008.

[35] S. Khanna et al. An FRAM-based nonvolatile logic MCU SoC exhibiting 100% digital state retention at vdd= 0 V achieving zero leakage with < 400-ns wakeup time for ulp applications. IEEE J SOLID-ST CIRC, pages 95–106, 2014.

[36] Y. Kim et al. Maximum power transfer tracking for a photovoltaic-supercapacitor energy system. In ISLPED, pages 307–312, 2010.

[37] H. Li and Y. Chen. Nonvolatile Memory Design: Magnetic, Resistive, and Phase Change. 2011.

[38] Linear Technology. LT8490-high V, high I, buck-boost battery charge controller with MPPT.

[39] J.-Q. Liu et al. A MEMS-based piezoelectric power generator array for vibration energy harvesting. MICROELECTR J, pages 802–806,2008.

[40] V. Liu et al. Ambient backscatter: Wireless communication out of thin air. COMPUT COMMUN REV, pages 39–50, 2013.

[41] Lively. Lively. http://mylively.com/.

[42] C. Lu et al. Maximum power point considerations in micro-scale solar energy harvesting systems. In ISCAS, pages 273–276, 2010.

[43] S. J. A. Ma jerus et al. Wireless, ultra-low-power implantable sensor for chronic bladder pressure monitoring. JETC, pages 11:1–11:13,2012.

[44] C. Meng et al. Ultrasmall integrated 3D micro-supercapacitors solve energy storage for miniature devices. Advanced Energy Materials,2014.

[45] P. P. Mercier et al. Energy extraction from the biologic battery in the inner ear. NAT BIOTECHNOL, pages 1240–1243, 2012.

[46] J. Nickels et al. Find my stuff: Supporting physical ob jects search with relative positioning. In UbiComp, pages 325–334, 2013.

[47] P. H. L. Notten et al. 3-D integrated all-solid-state rechargeable batteries. ADV MATER, pages 4564–4567, 2007.

[48] Panasonic. MN101LR05D/04D/03D/02D datasheet. http://www.semicon.panasonic.co.jp/ds4/MN101L05 E.pdf .

[49] V. Raghunathan and P. Chou. Design and power management of energy harvesting embedded systems. In ISLPED, pages 369–374,2006.

[50] V. Raghunathan et al. Emerging techniques for long lived wireless sensor networks. IEEE COMMUN MAG, pages 108–114, 2006.

[51] Y. Ramadass and A. Chandrakasan. A battery-less thermoelectric energy harvesting interface circuit with 35 mV startup voltage. IEEE J SOLID-ST CIRC, pages 333–341, 2011.

[52] B. Ransford. Transiently Powered Computers. PhD thesis, University of Massachusetts Amherst, Jan. 2013.

[53] N. Sakimura et al. A 90 nm 20 MHz fully nonvolatile microcontroller for standby-power-critical applications. In ISSCC, pages 184–185,2014.

[54] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. IEEE DES TEST COMPUT, pages 62–74,2001.

[55] STMicroelectronics. SPV1050-ULP energy harvester and battery charger with embedded MPPT and LDOs.

[56] tado. tado cooling. http://www.tado.com/.

[57] C. Wang et al. Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor. In ASP-DAC, pages 379–384, 2014.

[58] B. Zhai et al. A 2.60pJ/Inst subthreshold sensor processor for optimal energy efficiency. In Symposium on VLSI Circuits, pages 154–155, 2006.

[59] P. Zhang et al. QuarkOs: Pushing the operating limits of micro-powered sensors. In HotOS, 2013.

[60] P. Zhang and D. Ganesan. Enabling bit-by-bit backscatter communication in severe energy harvesting environments. In NSDI, pages 345–357, 2014.

[61] M. Zwerg et al. An 82 µA/MHz microcontroller with embedded FeRAM for energy-harvesting applications. In ISSCC, pages 334–336,2011.

Dynamic Taint Analysis Tools: A Review

Abdullah Mujawib Alashjaee *alas0145@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Northern Borders University
Arar, 73222, Saudi Arabia

Salahaldeen Duraibi *dura6540@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Jazan University
Jazan, 45142, Saudi Arabia

Jia Song *jsong@uidaho.edu*
Computer Science Department
University of Idaho
Moscow, ID, 83844, USA

Abstract

Taint analysis is the trending approach of analysing software for security purposes. By using the taint analysis technique, tainted tags are added to the data entering from the sensitive sources into the applications, then the propagations of the tainted data are monitored carefully. Taint analysis can be done in two ways including static taint analysis where analysis is conducted without executing the program, and dynamic taint analysis where the tainted data is monitored during the program execution. This paper reviews the taint analysis technique, with a focus on dynamic taint analysis. In addition, some of the existing taint analysis tools and their application areas are reviewed. In the end, the paper summarises the defects associated with each of the tools and presents some of them.

Keywords: Taint Analysis, Static Analysis, Dynamic Analysis.

1. INTRODUCTION

Software security analysis is important for testing Commercial off the Shelf (COTS) systems. It can be accomplished by employing source code or binary code. However, source code is not available in most of the cases for software security analysis. Hence, binary code analysis is used for a number of reasons, including software forensics [1, 2], malware analysis [4], and performance analysis and debugging [3]. A number of binary code analysis approaches are in the literature, and the most popular ones include symbolic execution, concolic execution, static taint analysis and dynamic taint analysis [5].

Capitalizing on the issue of efficiency identified in the fuzzing techniques, symbolic execution, which is another conventional binary code analysis approach, has come into being [6]. Different from other techniques, such as concrete execution that take concrete input values, symbolic execution uses symbols that abstractly represent specified input values for vulnerability analysis [7]. However, the technique is suffering from the famous path-explosion problem when symbolically executing large programs [8].

In view of improving the problems identified in symbolic execution, researchers have started working on taint analysis. According to Xiajing Wang and colleagues [32], taint analysis has first been proposed by Funnywei [75]. Taint analysis works in a triple form manner of source, sink, and sanitizer. The source is where some untrusted or confidential input data is introduced to the application, probably from the application API or network interface. The sink is the sensitive point in the application that performs secure operations, such as sensitive banking transactions, and needs to be protected from violation of integrity, confidentiality, and availability of the application. Sanitizer refers to the process where the tainted input data is no longer considered harmful to the information security of the application by means of removal of harmful operations such as malicious programs that may cause the application to function out of its intended operation [7]. In short, taint analysis helps software analyzers to take an informed decision on whether the data introduced at the input point or source of the application can be allowed to propagate to the sink point without harm, or else the application will suffer from some security issues such as data leakage or other more dangerous operations such as buffer overflow.

There are two types of taint analysis approaches, Static Taint Analysis (STA) and Dynamic Taint Analyses (DTA). In STA, analyzers test an application by examining the intermediate code without the execution of the application. Static taint analysis is mostly carried out in a two-step manner, including disassembly of the intermediate code and conducting analysis on the resulting assembly code [9, 10]. It may sometimes use binary codes for application security analysis. However, since source code rarely comes with COTS software, it makes the STA approach harder to combat malicious programs, thus reducing its application. Similarly, analyzing binary codes with STA approaches have endured complications and challenges [11]. For instance, malware with strong evasion techniques can easily escape the STA approach [4]. These limitations have motivated the identification of alternative approaches that can overcome such defects to analyze applications accurately and reliably.

On the other hand, in DTA, applications are tested during runtime for possible vulnerabilities [12]. Both STA and DTA approaches have weaknesses and strengths. For example, when conducting information flow analysis in an application, DTA can suffer from runtime overhead which may make it fall short of analyzing all the code, causing it not to discover some potential threats. On the other hand, since STA analyzes the application code without executing it, it may suffer from an accuracy issue. As a result, some researchers proposed tools that mix the two techniques to analyze flaws or vulnerabilities in applications [13-15]. Some researchers have used the STA approach before or after DTA [15, 16]. In doing so, for example, STA is employed after DTA in order to see whether analysis has missed anything suspicious after using DTA. STA can be employed before DTA to analyze the behavior of the application prior to the code execution in a live environment.

Conducting vulnerability analysis on software in cases where the source code is not available, for example, COTS, software security analysts use the DTA approach as the ideal option [17].

Usually, DTA methods are implemented at the hardware level or code level. For instance, some of the DTA methods are implemented within the hardware [18-21]. Although this implementation relatively provides the lowest overhead, it is less flexible and the least practical because it requires significant architectural and microarchitectural changes to the processor. By using source code instrumentation to track the propagation of the tainted data, DTA can also be performed at the code level of the software [22-26]. This approach is also less practical since source code is hardly available for security analysis in most applications. However, to perform data flow tracking without hardware modification or source code, the DTA methods such as Dytan [27], Libdft [22], Argos [28], BitBlaze [29], and DTA++ [30] use binary code to perform security analysis. This approach is used more prevalently because it enables a wide variety of analysis. For this paper, the DTA approaches that use binary code is the focus of interest. This paper is aimed at presenting an analytical view of static and dynamic taint tools.

The rest of the paper is organized as follows: Section 2 is the background of the study providing basic knowledge of DTA. Section 3 presents a review on several commonly used STA tools, and Section 4 presents the review of DTA tools. Section 5 presents lessons learned, and Section 6 concludes the paper.

2. BACKGROUND

The primary focus of this paper is on the use of DTA approaches for software security testing. In the following subsections, we will provide the readers with the preliminary details for understanding the purpose, techniques, and key concerns of the research work of taint analysis. Different aspects of DTA are in a general manner summarized in Section 2.1.

2.1 Concepts of Dynamic Taint Analysis Approach

This section, explains basic concepts about DTA. DTA is also referred to as dynamic information flow tracking. The approach is about observing the behaviour of certain untrusted programs as they execute in a monitored environment. The central idea of the DTA approach is to label some incoming data values as tainted and to propagate them through operands as instructions execute. This happens by marking certain values in the CPU registers or memory locations as tainted, and observing the tainted data as they propagate during the code execution. A taint propagation policy is associated with each instruction to specify whether each output operand should be tainted or untainted based on the taint status of the input operands.

2.1.1 Analysis Techniques

DTA can be accomplished either by control or data flow tracking [1]. Control flow tracking is an approach to show how the hierarchical flow of control in a given application is sequenced. It makes an easier analysis of all possible execution paths of an application. The output of control flow analysis is usually expressed in Control Flow Graphs (CFG), where each instruction or a block of instructions is represented by a node and the control flow between two nodes is indicated by direct edges. On the other hand, based on the problem that needs to be investigated, DTA computes a set of possible values at every point in an application. That is, data flow tracking is for monitoring programs from the perspective of how the program processes the data [2].

2.1.2 Offline and Online Dynamic Taint Analysis

Dynamic information flow tracking can be performed offline or online. In offline analysis, the trace of program execution is recorded into trace files and later analyzed by replaying those trace files. In online analysis, the security analysis is conducted by monitoring the program execution. Online analysis is considered to be more accurate and easier to implement, but it suffers from slow execution [32]. Using traces for later analysis will let analysts get thorough information about what has happened, but the raw trace file may become complicated to understand [8]. On the other hand, in doing online analysis, incident response can be performed in a timely manner, but it may sometimes end up as a false alert [8, 32].

2.1.3 Modes of Implementation

Dynamic information flow tracking tools can be implemented at the user or kernel level of an operating system. This depends on the type of security matter under investigation and the level of information extraction needed for the analysis [3]. For instance, programs such as word processing and imaging applications are executed at the user level of operating systems. On the other hand, operating systems perform their operations at the kernel level. Hence, DTA tools can be developed as targeting either the analysis of the user applications that work at the user level or the analysis of the privileged applications that have direct access to the kernel level processes.

2.1.4 Taint Granularity

The granularity of tracking the application has important implications for the usage of DTA tools. In DTA, analysis can be conducted in a fine-grained or coarse-grained manner [34, 35]. In coarse-grained information flow, the tainted data is tracked at the granularity of a whole system level, while in fine-grained analysis, tainted data is tracked at the granularity at the process level.

At the data level, data units can be tagged as small as a bit or as large as chunks of memory [35]. That is, in coarse-grained analysis, fewer tags are required compared to fine-grained analysis. Coarse-grained analysis tools are often easier to design and implement but may inherit trackless information causing false alarms [35]. Conversely, by tracking the information flow at the fine granularity, the analysis is more flexible and more precise but may require more memory space [35]. In most cases, researchers consider one over another believing that, for example, one is more effective than the other. However, Vassena et al. argued that both coarse and fine granularities are equally important in DTA [34].

2.1.5 Dynamic Binary Instrumentation (DBI)

Dynamic Binary Instrumentation refers to the analysis of an executable code through injecting additional code into the compiled code at runtime. This is usually implemented using a Just-in-Time (JIT) compiler. In DBI, code is executed in basic blocks, and the code at the end of each block is modified so that control is passed to the analysis engine to perform a number of checks, such as whether a system call is being executed [6]. Two of the most popular frameworks for achieving dynamic instrumentation in Windows are DynamoRIO [7] and Intel Pin [8].

2.2 Challenges in Dynamic Taint Analysis

Challenges that DTA has to face when analyzing applications for security can include soundness, precision, and overhead. In some papers, these are referred to as over-tainted, under-tainted, and overhead [17]. Under-tainted refers to a situation where values expected to be marked as tainted are not, while over-tainted occurs when too many values are marked as tainted. For instance, tools are still suffering from the issue of accuracy where in some cases taint may spread too much or happen to be missing, causing over-tainting or under-tainting respectively. The issue of balancing speed and accuracy is another challenge [8]. DTA tools sometimes cause overhead, minimizing the performance of the system [12].

3. STATIC TAINT ANALYSIS TOOLS

The STA technique is used for application vulnerability testing. There are a number of software vulnerability testing tools that utilize STA for deep and exhaustive tracking and prevention of suspicious data. In most cases, STA is conducted outside the testing environment, but it provides better code coverage analysis compared to DTA [40]. Existing researches employing STA can be categorized into three main areas including software privacy analysis [41], software forensics [42], web application vulnerability analysis [43, 44, 45, 46].

Conventional privacy-enhancing technologies have fallen short of assessing and auditing the privacy of cutting edge technologies. Detailed and often manual examination that is needed for these technologies makes privacy assessment a more complex, time-consuming, and tiresome task. Taint analysis has recently been used for realtime privacy monitoring of system privacy [41]. For instance, Celik et al. present SAINT, a system that can be used by the IoT consumers to assess the privacy risks that can come with the adoption of IoT devices [47]. Likewise, in digital forensics identifying potential evidence is at the center of any investigation. Evidence identification is challenging where only executable code is available; for example, identifying the existence of malware at the memory of a system where there is no source code [48]. Fordroid is a fully automated forensics tool developed based on the STA approach [42].

Another security area where the STA approach is widely used is web application vulnerability analysis. Tripp et al., use an STA in the design and implementation of TAJ to analyze web application security vulnerability [49]. TAJ has later been improved into a more scalable and precise version called ACTARUS [50]. A method proposed by Kurniawan et al. detects web file injection vulnerability in web applications using a PHP parser to traverse abstract syntax trees of the source code [51], while the method uses source codes for web application vulnerability assessment [52]. F4F makes use of an augmented taint analysis engine that generates a web application's source code in a simple Web Application Framework Language (WAFL) [53]. Tripp et al. proposed the most popular Web application security analysis tool called ANDROMEDA [54].

Sources	Year	Tools	Security Focus area	Need Source Code	Used Platform	Automated/Manual	Specific Area
[47]	2018	SAINT	Data leak (Privacy)	YES	SmartThings/ OpenHAB/ Apple's HomeKit	automated	Commodity IoT
[41]	2018	Fordroid	digital forensics	YES	Android	automated	Android applications
[49]	2009	TAJ	Vulnerability analysis	YES	Java	automated	Web Applications
[50]	2011	ACTARUS	Vulnerability analysis	YES	Java	automated	Web Applications
[53]	2011	F4F	Vulnerability analysis	YES	Java	Manual	Web Applications
[54]	2013	ANDROMEDA	Vulnerability analysis	YES	Java	automated	Web Applications

TABLE 1: STA Tools.

Table 1 summarizes and compares the STA tools reviewed in this section. Most of the tools are vulnerability mining tools that require source code for the analysis.

4. DYNAMIC TAINT ANALYSIS TOOLS

There are a number of research areas where DTA has been used for solving security problems, including private data leak detection, application vulnerability detection, malware analysis, and forensics [17]. For example, several researchers presented a Privacy Scope approach that uses DTA to find application leaks [55]. The approach is believed to be accurate and efficient and is implemented at the user environment to help pinpoint information leaks even if the sensitive data is encrypted. This approach uses function call summaries to handle taint propagation to reduce the overhead of the information flow tracking. In addition, this approach uses on-demand instrumentation to enable fast loading and to be able to run on large applications to precisely track information. Different from TightLip [56] and Privacy Oracle [57], information leakage detecting tools that are limited to applications whose outputs only depend on inputs, Privacy Scope can trace multiple input data.

TaintEraser is another DTA tool proposed for the prevention of sensitive data leaks [58]. TaintEraser conducts its analysis at the application level to let off-the-shelf application users run their applications while preventing unwanted information exposure. Similarly, researchers implement the taint propagation within the kernel for a reduced overhead in tracking in which they try to achieve near-real-time analysis. TaintEraser uses on-demand instrumentation to enable fast loading of large applications, and a semantic-aware instruction-level tainting for increased accuracy. The tool is tested with Internet Explorer, Yahoo! Messenger, and Windows Notepad where it generated no false positives, precisely preventing user sensitive data that would have otherwise been leaked to unwanted channels [58]. TaintEraser uses PIN [58] as a dynamic binary translator to accomplish its application-level analysis. The tool supports a simple privacy policy whereby a user first specifies sensitive input data to monitor, and subsequently TaintEraser blocks any data derived from the sensitive input data from moving to output channels that are specified as restricted. In doing so, TaintEraser monitors applications with input data marked 'sensitive.' Once such applications are moving out of the network, TaintEraser would replace sensitive bytes in those applications with randomly chosen bytes [58].

Information flow tracking is one of the widely used information leak detection methods for smartphones. For instance, TaintDroid is a tool that provides Android smartphone users a means of testing whether third-party applications collect and share their private data [59]. TaintDroid uses a system-wide information flow tracking to analyze Android apps for data leakage. The system is a near-real-time tool and is capable of tracking multiple sources of sensitive data at one time. Researchers benchmarked their work with Android's Activity Manager. It is detected that Taintdroid adds 3% overhead. In addition, by employing Taintdroid to monitor the behavior of 30 Android apps, 68 instances of potential misuse of the users' private data were detected. At the time of its development, according to the authors, TaintDroid was the most effective and efficient privacy testing tool for Android apps [59]. In this light, Taindroid is the prime candidate tool that can help Android smartphone users make an informed use of third-party applications.

DTA is used for unknown vulnerability detection by looking for misuses of user input during a program execution [17]. Vigilante [60] is an end-to-end approach that collaboratively detects vulnerability at the end host. The tool runs instrumented software to detect worms at the host and broadcasts alerts upon the detection of one. Subsequently, once an alert is broadcasted the host automatically generates filters that would block infection of the suspected worm without blocking innocuous traffic. With the use of Vigilante, there is no need for trust between hosts because it uses a cooperative worm detection mechanism distributed all over the network, thereby making it hard for worms to evade from detectors. However, Vigilante requires hosts to run expensive detection engines that can spread highly accurate detection loads once a worm is detected over the network.

Lift [61] is another vulnerability detecting approach with a low-overhead information flow tracking mechanism. The tool is generic in the sense that it does not only target specific vulnerability exploits such as worm, buffer overflow, format string, etc. Rather, Lift is a software-only approach that exploits dynamic binary instrumentation and optimizations for detecting various types of security attacks. Likewise, Lift is more specific in selecting tag propagation paths because it eliminates unnecessary tracking, coalesces information checks, and efficiently switches between target programs and instrumented information flow tracking code. The tool is implemented on StarDBT [61], a dynamic binary translator, on Windows experimenting web applications from server and client sides. Compared to previous works, the tool shows relatively better results [61].

Newsome and Song propose another host-based DTA tool that automatically detects Format String and Overwrite attacks exploits on commodity software [62]. These researchers referred to their tool as TaintCheck. TaintCheck has been employed in testing a number of programs and turned out to not have false positives for any of the programs. Likewise, TaintCheck enables an automatic semantic analysis to generate a signature for attack filtering after an exploited attack has been detected.

Previous studies focused on the use of DTA for securing centralized software. However, implementing such tools to distributed systems have raised issues of applicability, tool portability and analysis scalability [63]. Hence, the development of dedicated DTA tools that can be used for distributed systems is sought to be necessary. DistTaint, an application-level dynamic taint analyzer, is proposed for this aspect [64, 65]. However, the tool uses Java source code for its analysis.

Some researchers have taken one step beyond and have tried to secure cloud computing with DTA tools. For example, Papagiannis and Pietzuch proposed CloudFilter [66], a DTA tool that allows a cloud consuming organization to have control of its sensitive data and not be leaked to the cloud without its consent. CloudFilter intercepts file transfers between the consumer organization and cloud services, and subsequently performs logging and enforces propagation policies. Similarly, the tool controls where files propagate after they have been uploaded to the cloud and ensure that only authorized users may gain access to them. The researchers successfully applied CloudFilter to Dropbox and GSS whereby they were able to control the data propagation [66].

CloudFence is another data flow tracking service model [67] that monitors data leaks in cloud services. Researchers propose the tool to be hosted by the cloud providers for consumers to independently audit their data residing in that same cloud. The tool can also give cloud brokering companies to confine the propagation of sensitive data of their customers within well-defined domains. CloudFence is based on runtime binary instrumentation that supports byte-level data tagging and uses PIN as a dynamic binary translator. Similarly, CloudFence enables fine-grained data tracking for up to four billion users. To evaluate the effectiveness and practicality of the tool, the researchers implemented a CloudFence prototype using two publicly disclosed data leakage vulnerabilities in two real-world applications. Compared to the DTA tools Libdft and SiteBar; CloudFence shows a runtime overhead which is comparable to that of Libdft and larger performance impact in comparison to SiteBar.

Some researchers have amazingly employed dynamic information flow tracking for forensics readiness purpose, where system call level logging is conducted in order to ease “after-the-myth” investigation of attacks. For instance in one of the latest developments of this aspect, researchers proposed Rain, an attack investigation system, which uses a record-replay technology to record system-call events during runtime [68]. The system has the ability to perform instruction-level DTA that can filter out processes unrelated to the case to minimize the number of processes to be investigated for attack causality accuracy. In previous works, for example, Xiao et al. proposes PoL-DFA, a forensic system that can log the execution traces of the processes being monitored for investigating applications data leakage and contamination. Likewise, Sun and Oliveira propose an IoT forensics framework DDIFT [70] that uses a DTA module running in the IoT system controlling a mobile device, a forensics analysis module running in the cloud, and distributed optimization to conduct a decentralized forensic analysis of IoT applets.

One of the research areas where the DTA approach is exhaustively used is in dynamic malware analysis. The use of DTA is preferred for malware analysis because it is not easily defeated by techniques such as obfuscation and polymorphism. In this paper, we will review some of the most popular Malware analysis tools developed based on the DTA approach. Some malware analysis tools such as *Panorama* [71] and *Ether* [72] use hardware instrumentation. These types of malware analysis tools are not in our scope and therefore were not studied in this paper. However, TQana is an internet explorer browser plug-in tool that uses binary instrumentation for the analysis of malicious codes [73]. TQana performs at the kernel level to monitor all calls made by the malware. It observes both the functional behavior and information traces of the malware execution. Whenever a URL is entered into the address bar of the internet explorer, TQana implements information flow tracking using the Navigate event of the web browser which in turn introduces taints to the system. Another binary instrumentation based malware analysis tool is Cloudtaint [74]. Cloudtaint uses elastic taint tracking based on data flow tracking as well as control flow for malware detection of cloud-based applications. One of the best-known analysis tools developed based on the DTA approach is Dyton [27]. Dyton uses PIN for binary instrumentation providing an API where its user can configure the source and the sink to track the control of the information flow.

Tables 2 (a) and (b) show a summary of the DTA based tools reviewed in this paper. In tables 2 (a) and (b), the sign (✓) shows the existence or use of the parameter, listed in the tables, by the tools. In cases where cells are left blank, the corresponding parameter is neither used nor discussed in the papers reporting about the tools.

5. COMPARATIVE DISCUSSION

In tables 2 (a) and (b), the tools were comparatively analyzed for their employment of certain parameters. For instance, starting from the left, the tools were evaluated based on their area of focus. Of the 15 DTA based tools reviewed in this paper, 5 were for analyzing data leaks. In the literature of DTA, some researchers were categorically referring DTA based tools as data privacy suitable tools. So no wonder that most of the reviewed tools are dedicated to data leak analysis. The application of DTA based tools towards digital forensics is now getting the momentum. Three of the 15 reviewed tools are developed for digital forensics. Starting from its early days the DTA approach was used for malware analysis. Some DTA tools are generic in a way that they are not specific for their implementation area. For example, tools such as Vigilante, Lift, TaintCheck, and DistTaint are in general for application vulnerability analysis.

The tools were also evaluated for the type of analysis techniques they followed. The four columns under the analysis techniques section of Table 2 (a) show that most tools explicitly follow the data flow tracking analysis method. A good Handful of the tools including, Vigilante, Lift, DistTaint, Rain, DDIFT, TQama, CloudTaint, and Dyton, use both dataflow and control-flow for their analysis. Usually, tools that track data at a fine-grained level have shown law performance compared to those performing tracking at a coarse-grained level. However, 9 out of the 15 tools reviewed have implemented tracking at data or process (fine-grained) level analysis. Likewise, in

Table 2 (a), the binary instrumentation tool used in each of the tools is depicted in the DBI tools column. Some the tools do not specify which binary instrumentation tool they employ. However, undeniable number of tools have used the most popular binary instrumentation tool PIN. That is, PIN is a good candidate for any prospected DTA tools.

In Table 2 (b), we evaluated the mode of the tools' implementations. Usually, DTA tools perform their analysis at the user or/and kernel levels of the operating system. Only 4 tools can conduct analysis at both user and kernel levels, while the remaining 11 tools do analysis at the user or kernel levels. In Table 2 (b), the soundness, precision, and performance of each of the tools are evaluated. We could hardly grab soundness of the tools because most of the researchers did not discuss in the relative papers. However, only have shown interest in indicating the soundness of the tools. We mostly based our evaluation on the literature, particularly what other researchers have said about the tools. As a result, most of fine-grained tools have shown high overhead. Furthermore, we have studied what kind of environment the model has been implemented. As indicated in the last two columns of Table 2 (b) most of the tools are implemented in virtualized environments.

Sources	*Year*	*Tools*	*Security Focus area*	*Analysis Technique*				*Taint Granularity*		**DBI tool**
				DataFlow	ControlFlow	Implicit	Explicit	Fine-grained	Coarse-grained	
[55]	2009	Privacy Scope	Data leak	✓			✓	✓		PIN
[58]	2011	TaintEraser	Data leak	✓			✓	✓		PIN
[46]	2014	TaintDroid	Data leak	✓		✓			✓	
[59]	2005	Vigilante	Vulnerability analysis	✓	✓				✓	Nirvana
[60]	2006	Lift	Vulnerability analysis	✓	✓		✓	✓		StarDBT
[61]	2005	TaintCheck	Vulnerability analysis		✓			✓		Valgrind
[63, 64]	2019	DistTaint	Vulnerability analysis	✓	✓	✓		✓		
[65]	2012	CloudFilter	Data leak	✓			✓		✓	
[66]	2013	CloudFence	Data leak	✓			✓	✓		PIN
[67]	2017	Rain	Digital Forensics	✓	✓			✓		PIN
[68]	2016	PoL-DFA	Digital Forensics	✓			✓	✓		ET-Analyzer
[69]	2018	DDIFT	Digital Forensics	✓	✓				✓	
[72]	2007	TQana	Malware analysis	✓	✓				✓	PIN
[73]	2014	Cloudtaint	Malware analysis	✓	✓			✓		PIN
[26]	2007	Dyton	Malware analysis	✓	✓				✓	PIN

TABLE 2 (a): Dynamic Taint Analysis Tools.

Sources	*Year*	*Tools*	*Security Focus area*	*Modes of Implementation*		*Challenges faced*			*Environment*	
				User	Kernel	Soundness	Precision	Overhead	Emulated	Virtual
[55]	2009	Privacy Scope	Data leak	✓	✓		High	Low		✓
[58]	2011	TaintEraser	Data leak		✓		High	Low		✓
[46]	2014	TaintDroid	Data leak	✓			High	High		✓
[59]	2005	Vigilante	Vulnerability analysis		✓	Low	High	High		✓
[60]	2006	Lift	Vulnerability analysis	✓	✓	High	High	Low		✓
[61]	2005	TaintCheck	Vulnerability analysis	✓		Low	High	High	✓	
[63, 64]	2019	DistTaint	Vulnerability analysis	✓	✓	Low	High	High		
[65]	2012	CloudFilter	Data leak	✓				High		
[66]	2013	CloudFence	Data leak	✓	✓	High	Low	High		✓
[67]	2017	Rain	Digital Forensics		✓	High	High	High	✓	
[68]	2016	PoL-DFA	Digital Forensics	✓			High	High		✓
[69]	2018	DDIFT	Digital Forensics	✓			High	Low		
[72]	2007	TQana	Malware analysis		✓			High	✓	
[73]	2014	Cloudtaint	Malware analysis		✓		High	High	✓	✓
[26]	2007	Dyton	Malware analysis	✓			High	High	✓	

TABLE 2 (b): Dynamic Taint Analysis Tools.

6. LESSONS LEARNED

Based on our review and current status of the DTA tools, it is believed that there is an urgent need of designing DTA based vulnerability analysis tools with a reduced false reporting rate. On the other hand, such tools may optimize the efficiency of the DTA by selectively controlling the

number of taints to be spread for each analysis. This can be accomplished by removing unnecessary taints from the system.

In addition, the DTA tools in the literature can only detect some specific vulnerabilities. Hence, the development of a generic tool that combines existing techniques in order to detect myriad security vulnerabilities will be a value add to the domain. The literature is also lacking tools that can analyze inter-applications or inter-systems data leaks.

Adopting DTA to the analysis of cutting edge technology is also lagging behind. There are only a number of tools that have been applied to cloud computing and IoT environments. Worth mentioning is that none of these tools focused on the vulnerability analysis of cloud or IoT applications. Some focused on data leak detection while others were for either digital forensics or malware analysis. The primary reason why the vulnerability analysis DTA based tools are not extended to cloud and IoT technologies is because of the infancy of the two areas. Other reasons may include the heterogeneous nature of devices and applications involved in cloud and IoT technologies. Furthermore, how data is distributed, aggregated and processed in cloud and IoT technologies may pose challenges in the data flow tracking. Particularly, different types of IoT technologies, Operating systems, and network protocols from different vendors make it hard implementation of DTA tools to the IoT ecosystem.

6. CONCLUSION

In this paper, taint analysis tools have been studied. At first, different areas where the taint analysis approach is implemented are discussed. Subsequently, a brief overview of the STA and a number of tools that have been developed based on STA are presented. Likewise, the section about DTA is starting with the basics and definitions to consequently build on the description of the tools and frameworks in the literature. A number of DTA based tools are thoroughly reviewed. Their areas of implementation were studied together with the shortcomings reported in each of the tools. A deeper understanding of the DTA approach and the effective adaption of its tools will have an improving effect on software security analysis.

7. REFERENCES

[1] D Zou, J Zhao, W Li, Y Wu, W Qiang., "*A Multigranularity Forensics and Analysis Method on Privacy Leakage in Cloud Environment."* IEEE Internet of Things Journal, 2018. 6(2): p. 1484-1494.

[2] A.N. Moussa, N. Ithnin, and A. Zainal, "*CFaaS: bilaterally agreed evidence collection."* Journal of Cloud Computing, 2018. 7(1): p. 1.

[3] X. Meng, and B.P. Miller. "*Binary code is not easy*." in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 2016. ACM.

[4] M. Shudrak, and V. Zolotarev. "*The technique of dynamic binary analysis and its application in the information security sphere*." in *Eurocon 2013*. 2013. IEEE.

[5] C Chen, B Cui, J Ma, R Wu, J Guo, W Liu. "*A systematic review of fuzzing techniques."* Computers & Security, 2018. 75: p. 118-137.

[6] R Baldoni, E Coppa, DC D'elia, C Demetrescu. "*A survey of symbolic execution techniques."* ACM Computing Surveys (CSUR), 2018. 51(3): p. 50.

[7] Z Feng, Z Wang, W Dong. "*Bintaint: A STA Method for Binary Vulnerability Mining*." in *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*. 2018. IEEE.

[8] J Cai, P Zou, J Ma, J He. "*Sworddta: A dynamic taint analysis tool for software vulnerability detection."* Wuhan University Journal of Natural Sciences, 2016. 21(1): p. 10-20.

[9] K. Liu, H.B.K. Tan, and X. Chen, "*Binary code analysis.* Computer," 2013. 46(8): p. 60-68.

[10] C. Cadar, D. Dunbar, and D.R. Engler. "*KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*." in *OSDI*. 2008.

[11] W. Aman, "*A framework for analysis and comparison of dynamic malware analysis tools."* arXiv preprint arXiv:1410.2131, 2014.

[12] J. Kim, T. Kim, and E.G. Im. "*Survey of dynamic taint analysis*." in *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*. 2014. IEEE.

[13] E Zhu, X Li, F Liu, X Li, Z Ma. "*Constructing a hybrid taint analysis framework for diagnosing attacks on binary programs."* Journal of Computers, 2014. 9(3): p. 566-575.

[14] M Ahmad, V Costamagna, B Crispo "*TeICC: targeted execution of inter-component communications in Android*." in *Proceedings of the Symposium on Applied Computing*. 2017. ACM.

[15] M. Monga, R. Paleari, and E. Passerini. "*A hybrid analysis framework for detecting web application vulnerabilities*." in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*. 2009. IEEE Computer Society.

[16] A. Getman, V. Padaryan, and M. Solovyev. "*Combined approach to solving problems in binary code analysis"*. in *Proceedings of 9th International Conference on Computer Science and Information Technologies (CSIT'2013)*. 2013.

[17] P. Dai, Z. Pan, and Y. Li. "*A Review of Researching on Dynamic Taint Analysis Technique*." in *2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*. 2018. Atlantis Press.

[18] S Chen, J Xu, N Nakka, Z Kalbarczyk. "*Defeating memory corruption attacks via pointer taintedness detection*." in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005. IEEE.

[19] GE Suh, JW Lee, D Zhang, S Devadas. "*Secure program execution via dynamic information flow tracking*." in *ACM Sigplan Notices*. 2004. ACM.

[20] G Venkataramani, I Doudalis, Y Solihin. "*Flexitaint: A programmable accelerator for dynamic taint propagation*." in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. 2008. IEEE.

[21] J Shin, H Zhang, J Lee, I Heo, YY "Chen *A hardware-based technique for efficient implicit information flow tracking*." in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016. IEEE.

[22] VP Kemerlis, G Portokalidis, K Jee, AD Keromytis. "*libdft: Practical dynamic data flow tracking for commodity systems*." in *Acm Sigplan Notices*. 2012. ACM.

[23] W. Xu, S. Bhatkar, and R. Sekar. "*Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks*." in *USENIX Security Symposium*. 2006.

[24] V. Ganesh, T. Leek, and M. Rinard. "*Taint-based directed whitebox fuzzing*." in *Proceedings of the 31st International Conference on Software Engineering*. 2009. IEEE Computer Society.

[25] TR Leek, GZ Baker, RE Brown, MA Zhivich, "*Coverage maximization using dynamic taint tracing*." 2007, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

[26] R Wang, G Xu, X Zeng, X Li, Z Feng *TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting.* Journal of Parallel and Distributed Computing, 2018. 118: p. 100-106.

[27] J. Clause, W. Li, and A. Orso. "*Dytan: a generic dynamic taint analysis framework*." in *Proceedings of the 2007 international symposium on Software testing and analysis*. 2007. ACM.

[28] G. Portokalidis, A. Slowinska, and H. Bos."*Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation*." in *ACM SIGOPS Operating Systems Review*. 2006. ACM.

[29] D Song, D Brumley, H Yin, J Caballero, I Jager "*BitBlaze: A new approach to computer security via binary analysis*." in *International Conference on Information Systems Security*. 2008. Springer.

[30] MG Kang, S McCamant, P Poosankam, D Song *Dta++: dynamic taint analysis with targeted control-flow propagation*. in *NDSS*. 2011.

[31] L Li, TF Bissyandé, M Papadakis, S Rasthofer. "*Static analysis of android apps: A systematic literature review."* Information and Software Technology, 2017. 88: p. 67-95.

[32] X Wang, R Ma, B Dou, Z Jian, H Chen, "*OFFDTAN: A New Approach of Offline Dynamic Taint Analysis for Binaries."* Security and Communication Networks, 2018. 2018.

[33] M Nunes, P Burnap, O Rana, P Reinecke, "*Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis"* Journal of Information Security and Applications, 2019. 48: p. 102365.

[34] M Vassena, A Russo, D Garg, V Rajani, "*From fine-to coarse-grained dynamic information flow control and back."* Proceedings of the ACM on Programming Languages, 2019. 3(POPL): p. 76.

[35] H. Yin, and D. Song, "*Whole-system Fine-grained Taint Analysis for Automatic Malware Detection and Analysis."* Technical paper. College of William and Mary & Carnegie Mellon University, 2006.

[36] M Polino, A Continella, S Mariani, S D'Alessio *Measuring and defeating anti-instrumentation-equipped malware*. in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 2017. Springer.

[37] D. Bruening, E. Duesterwald, and S. Amarasinghe. *Design and implementation of a dynamic optimization framework for Windows*. in *4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4)*. 2001.

[38] CK Luk, R Cohn, R Muth, H Patil, A Klauser. "*Pin: building customized program analysis tools with dynamic instrumentation*." in *Acm sigplan notices*. 2005. ACM.

[39] L.K. Yan, and H. Yin, "*SoK: On the Soundness and Precision of Dynamic Taint Analysis."*

[40] D. Boxler, and K.R. Walcott. *STA Tools to Detect Information Flows*. in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. 2018. The Steering Committee of The World Congress in Computer Science, Computer ….

[41] M. von Maltitz, C. Diekmann, and G. Carle. *Privacy Assessment Using STA (Tool Paper)*. in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. 2017. Springer.

[42] X Lin, T Chen, T Zhu, K Yang, F Wei "*Automated forensic analysis of mobile applications on Android devices."* Digital Investigation, 2018. 26: p. S59-S66.

[43] Z Xing, Z Bin, F Chao, Z Quan "*Staticly Detect Stack Overflow Vulnerabilities with Taint Analysis*." in *ITM Web of Conferences*. 2016. EDP Sciences.

[44] C. Feng, and X. Zhang. *A Static Taint Detection Method for Stack Overflow Vulnerabilities in Binaries*. in *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*. 2017. IEEE.

[45] F. Pauck, and H. Wehrheim. *Together strong: cooperative Android app analysis*. in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[46] S Arzt, S Rasthofer, C Fritz, E Bodden, A Bartel "*Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*." in *Acm Sigplan Notices*. 2014. ACM.

[47] ZB Celik, L Babun, AK Sikder, H Aksu, G Tan "*Sensitive information tracking in commodity IoT*." in *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018.

[48] N. Rosenblum, X. Zhu, and B.P. Miller. *Who wrote this code? identifying the authors of program binaries*. in *European Symposium on Research in Computer Security*. 2011. Springer.

[49] O Tripp, M Pistoia, SJ Fink, M Sridharan, *TAJ: effective taint analysis of web applications.* ACM Sigplan Notices, 2009. 44(6): p. 87-97.

[50] S Guarnieri, M Pistoia, O Tripp, J Dolby *Saving the world wide web from vulnerable JavaScript*. in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. 2011. ACM.

[51] A Kurniawan, BS Abbas, A Trisetyarso *STA Traversal with Object Oriented Component for Web File Injection Vulnerability Pattern Detection.* Procedia Computer Science, 2018. 135: p. 596-605.

[52] M.L. Minsky, *Computation*. 1967: Prentice-Hall Englewood Cliffs.

[53] M Sridharan, S Artzi, M Pistoia, S Guarnieri *F4F: taint analysis of framework-based web applications*. in *ACM SIGPLAN Notices*. 2011. ACM.

[54] O Tripp, M Pistoia, P Cousot, R Cousot *Andromeda: Accurate and scalable security analysis of web applications*. in *International Conference on Fundamental Approaches to Software Engineering*. 2013. Springer.

[55] Y Zhu, J Jung, D Song, T Kohno, D Wetherall, *Privacy scope: A precise information flow tracking system for finding application leaks*. 2009, Citeseer.

[56] A.R. Yumerefendi,, B. Mickle, and L.P. Cox. *TightLip: Keeping Applications from Spilling the Beans*. in *NSDI*. 2007.

[57] J Jung, A Sheth, B Greenstein, D Wetherall "*Privacy oracle: a system for finding application leaks with black box differential testing*." in *Proceedings of the 15th ACM conference on Computer and communications security*. 2008. ACM.

[58] DY Zhu, J Jung, D Song, T Kohno, *TaintEraser: Protecting sensitive data leaks using application-level taint tracking.* ACM SIGOPS Operating Systems Review, 2011. 45(1): p. 142-154.

[59] W Enck, P Gilbert, S Han, V Tendulkar *TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones.* ACM Transactions on Computer Systems (TOCS), 2014. 32(2): p. 5.

[60] M Costa, J Crowcroft, M Castro, A Rowstron *Vigilante: End-to-end containment of internet worms*. in *ACM SIGOPS Operating Systems Review*. 2005. ACM.

[61] F Qin, C Wang, Z Li, H Kim, Y Zhou "*Lift: A low-overhead practical information flow tracking system for detecting security attacks*." in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 2006. IEEE.

[62] J. Newsome, and D.X. Song. *Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software*. in *NDSS*. 2005. Citeseer.

[63] X Wang, H Ma, K Yang, H Liang "*An Uneven Distributed System for Dynamic Taint Analysis Framework*." in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. 2015. IEEE.

[64] X. Fu, and H. Cai." *A dynamic taint analyzer for distributed systems*." in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[65] X. Fu, "*On the scalable dynamic taint analysis for distributed systems*." in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[66] I. Papagiannis, and P. Pietzuch. "*Cloudfilter: practical control of sensitive data propagation to the cloud*." in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*. 2012. ACM.

[67] V Pappas, VP Kemerlis, A Zavou *CloudFence: Data flow tracking as a cloud service*. in *International Workshop on Recent Advances in Intrusion Detection*. 2013. Springer.

[68] Y Ji, S Lee, E Downing, W Wang, M Fazzini "*Rain: Refinable attack investigation with on-demand inter-process information flow tracking*." in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017. ACM.

[69] G Xiao, J Wang, P Liu, J Ming, D Wu "*Program-object level data flow analysis with applications to data leakage and contamination forensics*." in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. 2016. ACM.

[70] N. Sapountzis, R. Sun, and D. Oliveira. "*DDIFT: Decentralized Dynamic Information Flow Tracking for IoT Privacy and Security*." in *Workshop on Decentralized IoT Systems and Security (DISS)*. 2018.

[71] H Yin, D Song, M Egele, C Kruegel "*Panorama: capturing system-wide information flow for malware detection and analysis*." in *Proceedings of the 14th ACM conference on Computer and communications security*. 2007. ACM.

[72] A Dinaburg, P Royal, M Sharif, W Lee "*Ether: malware analysis via hardware virtualization extensions*." in *Proceedings of the 15th ACM conference on Computer and communications security*. 2008. ACM.

[73] M Egele, C Kruegel, E Kirda, H Yin, D Song. "*Dynamic spyware analysis."* 2007.

[74] J Yuan, W Qiang, H Jin, D Zou. "*CloudTaint: an elastic taint tracking framework for malware detection in the cloud."* The Journal of Supercomputing, 2014. 70(3): p. 1433-1450.

[75] Funnywei, “Bufer Overfow Vulnerability Mining Model [Z/OL],” 2003, http://xcon.xfocus.net/XCon2003/archives/ Xcon2003 funnywei.pdf.

A Survey of Symbolic Execution Tools

Salahaldeen Duraibi *dura6540@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Jazan University
Jazan, 45142, Saudi Arabia

Abdullah Mujawib Alashjaee *alas0145@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Northern Borders University

Jia Song *jsong@uidaho.edu*
Computer Science Department
University of Idaho
Moscow, ID, 83844, USA

Abstract

In the software development life cycle (SDLC), testing is an important step to reveal and fix the vulnerabilities and flaws in the software. Testing commercial off-the-shelf applications for security has never been easy, and this is exacerbated when their source code is not accessible. Without access to source code, binary executables of such applications are employed for testing. Binary analysis is commonly used to analyze on the binary executable of an application to discover vulnerabilities. Various means, such as symbolic execution, concolic execution, taint analysis, can be used in binary analysis to help collect control flow information, execution path information, etc. This paper presents the basics of the symbolic execution approach and studies the common tools which utilize symbolic execution in them. With the review, we identified that there are a number of challenges that are associated with the symbolic values fed to the programs as well as the performance and space consumption of the tools. Different tools approached the challenges in different ways, therefore the strengths and weaknesses of each tool are summarized in a table to make it available to interested researchers.

Keywords: Symbolic Execution, Concrete Execution, Concolic Execution, Binary Analysis.

1. INTRODUCTION

In cases where applications are analyzed for defects and source code is not available, software analysts have to conduct analysis at the binary code level of the application. Engaging binary code for software analysis is referred to as binary analysis. It is commonly used for error identification, reverse engineering, and security analysis. In addition, binary analysis is well known for its use for discovering vulnerabilities in software, and this paper focuses on that aspect of the binary analysis. To reveal vulnerabilities, disassembly of the binary executable needs to be done first and then the vulnerability patterns, such as buffer overflow, can be recognized.

Conducting binary analysis is challenging, because a great deal of useful information, such as symbolic information, data types, program structures, is not carried to the binary code. What is more, in the early days of using binary code analysis, analysts used to have difficulty

distinguishing between data and code in binary, because data fragments and executable code are mixed in the binaries [1]. Therefore, researchers in the domain of software testing have proposed a number of binary analysis techniques, including taint analysis, symbolic execution, and concolic execution.

Taint analysis is used for information flow tracking, data entering from some specific sources such as user input, application APIs or network interfaces are marked as tainted (untrusted). Then the propagations of the tainted data are tracked throughout the program and the uses of the tainted data are carefully checked. There are two ways of performing taint analysis, static taint analysis and dynamic taint analysis. Static taint analysis tools usually conduct the analysis in a controlled environment where the data are monitored before run time [2]. Tools developed based on static taint analysis can offer better code coverage in their analysis compared to dynamic taint analysis [2]. However, such tools suffer in that they cannot detect runtime security defects of applications. Static taint analysis can be used for different aspects of security analysis including data leak analysis [3], digital forensics [4], web application vulnerability analysis [5, 6]. On the other hand, dynamic taint analysis is a principled approach for tracking information flow during program execution. Different from the static taint analysis that needs for its analysis to run in a confined environment, dynamic taint analysis usually conducts analysis when the application is running in its intended environment. Moreover, dynamic taint analysis can be implemented within the hardware level of a system to conduct analysis [7-10], or at the software stack by either using the source code [11-15] or binary code [16]. However, since dynamic taint analysis conducts applications security analysis at the runtime, it can only find flows that are executed. Hence, it has less code coverage compared to the static taint analysis [2].

Symbolic execution remains one of the favored techniques when it comes to error detections [17]. It is usually used for testing applications for defects and security matters [18]. Symbolic execution has been proposed as a solution to the concrete execution that explores a specific actual data input and a single control flow path at a time. Instead in symbolic execution, a program is explored for the different paths it can take when fed with different inputs. To that end, to accomplish this, symbolic execution does not take actual data as input, rather, it uses symbolic input values. As a result, the output is given as a function of the symbolic value and is considered as a sound analysis compared to the concrete. Symbolic execution has suffered from execution path explosion for large or complex programs [19].

Hence, in order to mitigate the path explosion issue, concolic execution is proposed. Concolic execution combines concrete execution and symbolic execution in order to overcome inherent defects identified in the symbolic execution including path explosion and handling calls to native libraries [20]. That is, the program is executed on some concrete input values provided by the analyst and then symbolic path constraints are generated for that specific execution. Concolic execution was first proposed in 2001 by Eric Larson and Todd Austin [21].

The rest of the paper is organized as follows: Section 2 is the background of the study providing basics of how symbolic execution works. The studies of different symbolic execution tools are presented in Section 3. Section 4 discusses the lessons learned from reviewing the common tools, and Section 5 concludes the paper.

2. BACKGROUND

Conventionally, symbolic execution is used for analyzing sequential programs with integer variables [22]. Symbolic Execution uses symbolic values as input data rather than actual data and symbolic expressions as program variables. Different from the concrete execution approach that tests programs on specific input with a single control flow path, symbolic execution rather tests programs with different inputs against multiple execution paths. In symbolic execution, programs are fed with symbolic values instead of concrete input values [23]. The approach uses an execution engine that collects a set of constraints combined and formulas across each explored path. Once instructions are evaluated the formula is updated accordingly. The execution forks

when a branching instruction is encountered. A constraint solver - typically one suited for satisfiability modulo theories (SMT) - is used to evaluate expressions involving symbolic values, as well as for generating concrete inputs that can be used to run the program concretely along the desired path [24].

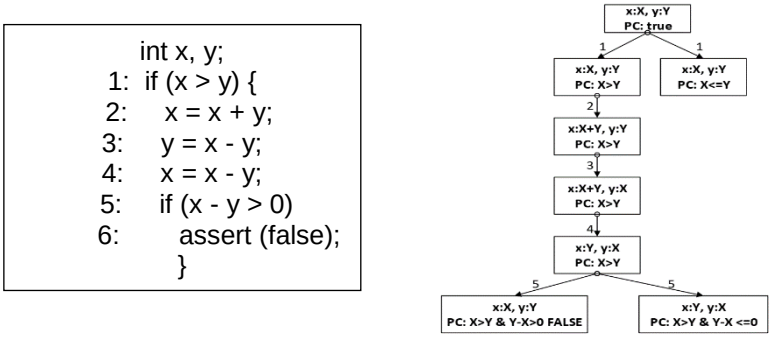


FIGURE 1: Code that swaps two integers and the corresponding symbolic execution tree [20].

The state of the symbolically executed program is usually depicted in a symbolic execution tree that shows the execution paths the input followed during the analysis. The nodes of the tree are the states of the program. Each execution path between states is represented by an arc labelled with a transition number. For example, in Figure 1, the code segment which swaps the value of integer variables x and y is shown to the left of the figure. A corresponding symbolic execution tree can be built and is depicted on the right of Figure 1. According to the symbolic execution tree, in the initial path condition, x and y have the symbolic values X and Y. In each transition, based on the input values, the path condition is updated. Following the execution of the initial statement, both 'then' and 'else' alternatives of the 'if' statement are possible, and the path condition is updated accordingly. Where inputs do not satisfy the path condition (false), it means the symbolic state of the program is not reachable, and as a result, the symbolic execution will not continue for that path of the program. For instance, statement number six (6) is unreachable in the symbolic execution tree in Figure 1.

In order to perform symbolic execution analysis, the program has to exercise a large set of paths through its execution tree that is because whenever more paths are explored, the higher the coverage of examined codes. Nevertheless, such an enumeration of execution paths is computationally expensive. Traditionally symbolic execution uses exhaustive exploration of the possible execution paths. However, this makes the analysis process to remain applicable only to small applications, causing analysts to aim for less ambitious goals. Having said that, a number of approaches that can ease the process of path exploration are employed by most of the recent symbolic execution tools. For example, some researchers proposed standard model checking tools for Java programs in order to perform the path selection process [25, 26].

Generally, symbolic execution challenges are related to four different areas that have been studied including memory, environment, state-space exploration, and constraint solving [27]. Memory related challenges are about the way the symbolic engine manipulates pointers, arrays and other complex objects that may give rise wrong symbolic values or expressions. Environment-related challenges are about the external call that may cause side effects to the execution. State-space exploration is about the control flow path that the execution engine should explore within a reasonable amount of time. Finally, the constraint solving related challenges are simply about the issues pertaining to the scalability of the constraint solver of a tool.

Likewise, according to Xu et al., challenges pertaining to the symbolic execution tools (software and program testing) can be referred to as *symbolic-reasoning* and *path-explosion* challenges [28]. Symbolic-reasoning challenges are related to the problems that cause symbolic execution tools to generate incorrect test results for particular control flows. These include a symbolic variable declaration, symbolic jumps, symbolic memories, contextual symbolic values, floating-

point numbers, buffer overflows, and arithmetic overflows. Likewise, path explosion challenges are related to the problems that introduce increased control flows to analye a program [29]. In other words, these are problems that cause symbolic execution tools to require increased time and resources on exploring the paths needed for analysis [30]. These include external function calls, loops, and crypto functions that may cause path-explosion issues to both large-sized and small-sized programs. In this paper, these challenges are considered and solutions from each tool in relation to the challenging area are discussed.

3. SYMBOLIC EXECUTION TOOLS

This section discusses some of the famous software testing tools which utilize symbolic execution techniques. The tools proposed in the literature are mostly used for testing input generation [31], regression testing [32], program deobfuscation [33], and dynamic software updates [34]. In addition, there is another group of tools that uses symbolic execution to guide exploit generation [30], vulnerability finding [35], and fuzzing [36].

DART [37] is one of the early works with automated unit testing (concrete execution) technique. It combines three approaches in order to conduct software analysis. It uses static source code parsing for code inspection of C programs. It performs automatic random testing in order to find software bugs specifically inter-procedural bugs and bugs caused by the use of library functions. Finally, it conducts dynamic analysis in order to test how the program behaves under random testing. It tests programs for standards errors such as crashes, assertion violations, and non-termination. As a concrete execution tool, DART does not employ path selection mechanisms because it uses specific input with a single path testing scenario. However, the random choosing of the value over the domain of potential inputs (random testing) followed by DART may lead to the same observation behaviour that may cause redundancy. Likewise, in random testing, the chance of selecting inputs that cause buggy behaviour may be small [38].

CUTE [39] is a software testing tool that uses the concolic execution technique that combines concrete and symbolic executions. Different from DART, CUTE tests programs with first trying NULL, and then, in a subsequent execution, a concrete address, rather than making random choices. CUTE uses concretization of address to maintain consistency across different executions and due to efficiency in constraint solving. In this tool, a logical input map is used to generate memory input graphs for the unit under test. Sen, K. et al. reported that the CUTE works efficiently in exploring paths in C code to expose software bugs resulting in assertion violations, segmentation faults, or infinite loops [39]. The main reason why CUTE uses combined symbolic and concrete execution is to generate test inputs to explore different execution paths with which the execution proceeds [27]. In addition, the tool uses a constraint solver tool that facilitates the incremental generation of the input. Sen et al. proposed an implementation of CUTE in finding algebraic security attacks in cryptographic protocols and security breaches in unsafe languages, but have never published their work in this regard [40]. In another study jCUTE, the tools have been extended for Java programs [41].

Cadar et al. proposed EXE, an effective bug-finding tool [42]. EXE uses the concolic execution technique that runs symbolic inputs to track the constraints in memory locations. The tool uses real code in finding bugs and capitalizes on the effect of running a single code path by automatically generating concrete inputs that can run into multiple program execution paths. What makes EXE different is that once a path hits a bug it automatically generates a test case using the value that has triggered the bug as concrete values. The tool uses search heuristics for path selection. The researchers use two performance optimizations including cashing constraints to avoid calling Simple Theorem Prover (STP) solver and removing irrelevant constraints from the queries the tools send to STP solver.

SAGE proposed by Godefroid et al. is a concolic execution software testing tool that is internally used by Microsoft [43]. SAGE is considered as a general tool because it works at the instruction level to track integer constraints (bit-vectors). In another research, SAGE has been utilized as a

security testing tool [44]. The tool uses the random test style firstly employed by DART but mutates well-formed inputs using grammars. SAGE introduced a generational search as a constraint solver to explore the state space of large applications executed with large inputs. In addition, the tool uses a number of optimization techniques to improve the performance and memory usage of the constraint generation, such as tag caching where structurally equivalent tags are mapped to the same physical object, and local constraint caching. Moreover, the tool also uses the constraint subsumption optimization technique for structured-file parsing applications.

According to Tillmann and De, PEX is a software testing tool developed for .NET that produces a small test suite with high code coverage [45]. PEX performs analysis using dynamic symbolic execution. To reason about the feasibility of execution paths, PEX uses constraint solver Z3. Z3 [46] is an efficient satisfiability modulo theory solver which is commonly used in software verification and application analysis. By using Z3, PEX is able to reason operations such as substring, concatenation, and replacement. Z3 also offers binding for certain programming languages [46].

A recently-developed tool, Tracer, is another software testing tool that is developed based on symbolic execution approach [47]. Tracer is a verification tool for the finite-state of sequential C programs. The tool uses constraint logic programming (CLP) as a resolver. In addition, the tool uses interpolation methods including the strongest postconditions and weakest preconditions.

Identifying vulnerability in binary code is a complicated task. BitBlaze is one of the projects that focused on the analysis of binary codes for vulnerability analysis [2]. The BitBlaze project contributed to the community three tools each focusing different approaches of preforming binary analysis. The tools include Vine, a static taint analysis component, TEMU, a dynamic taint analysis component, and Ruder, a Concolic execution component. Rudder has core utilities and interfaces that enable users to take a snapshot and reload the exploration state providing user-specific path selection policies. The tool uses the ‘Lazy’ approach which collects necessary information in the symbolic machine during the execution. Moreover, the tool uses STP for symbolic-reasoning and breadth-first search approach for path selection [2].

BAP is one of the early binary analysis tools that is developed based on the symbolic execution approach [48]. BAP is a redesigned type of Vine [2] with the goal of including useful analysis and verification techniques that may be appropriate for binary code analysis and allowing user-level analysis. It assembles binary code into an optimized intermediate language (IL) and subsequently performs analysis at the IL level.

Automatic Exploit Generation (AEG) is developed based on concrete and symbolic execution approaches [49]. The tool identifies exploitable paths in a program. Hence to address the path-reasoning challenge, the tool employs a novel technique called preconditioned symbolic execution with which it targets paths that are more likely to be exploitable. In the report, the researchers have proposed five challenging areas where tools like AEG should focus on doing their analysis. The five challenging areas include the state space explosion problem, the path selection problem, the environment modelling problem, the mixed analysis challenge, and the exploit verification problem.

Different from the AEG, Mayham is a concolic execution tool that finds exploitable bugs in binary code without debugging information [50]. There are four design principles adopted by Mayham that make it different from the tools discussed previously. The tool makes forward symbolic execution with arbitrary time, it does not repeat work for maximized performance, the tool keeps the works of previous analysis for reusability, and finally, the tool can reason about symbolic memory. In addition, the tool is known for its hybrid way of combining offline and online executions. Another work similar to that provided in Mayham is proposed in Veritesting [51]. Veritesting is a binary only symbolic execution tool targeting large scale testing of commodity off-

the-shelf software. It uses dynamic symbolic execution for testing and static symbolic execution for verification.

A recent tool, Firmalice, is proposed for the analysis of privacy-sensitive and security-critical applications installed on the IoT devices specifically [52]. The tool mainly focuses on the identification of the existence authentication bypass activities and existing backdoor. The tool uses concretizing user input as a constraint solver.

Driller, developed by Stephens et al. is a hybrid vulnerability excavation tool that uses concolic execution to guide fuzzing [53]. The concolic execution component analyses the program, traces user input and utilizes its constraint-solving engine to guide fuzzing too take different paths, therefore it finds bugs located deeper in the code. Helping with a concolic execution component, Driller can detect more vulnerabilities, however, it requires a lot of computing power and may quickly run into the path explosion problem [53].

Table 1 summarizes the tools reviewed in this paper together with their techniques used to overcome challenges associated with symbolic-reasoning, path-reasoning, and the optimization approaches the tools employed in order to boost the performance or reduce the space required for the analysis. Only three tools have used optimization techniques (EXE, SAGE, and BAP). However, SAGE seems relatively more efficient in path reasoning and symbolic reasoning, while EXE is only good in path reasoning and BAP has shown less accurate. Four tools are language-dependent including CUTE, DART, EXE, and Tracer. Nevertheless, most of the tools reviewed in this paper are language independent and employ binary codes for their analysis. There has been an increase since 2012, which may show that symbolic execution tools are becoming more robust and main vulnerability analysis. The Concolic execution techniques have gained a higher bar of acceptance for the past decade. The tools have used different constraint solvers, however, the most used are SMT solvers. Of the 13 symbolic execution tools reviewed in this paper, 6 use Concolic approaches that combine symbolic with concrete executions.

The first five columns provided in Table 1 capture the nature of the tools and little do they say about the evaluation of the tools; the last column capture the performance of the tools. Most of the tools do not perform quantitative evaluation of their results. However, based on the reviewing we were able to quantitatively compare different claims reported in each of the papers. As a result, the last column of Table 1 discusses the overall performance of each of the tools in relation to other similar works reviewed here.

Sources	Tools	SE or CE	Targeting Language or Binary	Constraint solvers	Used Oprimization technique	Strengths & weaknesses
[39]	CUTE	CE	Language	approximate pointer constraints		its weakness is that it targets only C language programs
[37]	DART	SE	Language	depth first exploration		It lucks constraint solver for path selection, as it uses concrete inputs, that is it is time consuming
[42]	EXE	CE	Language	best-first search (BFS) heuristic and depth-first search	Constraint caching and Constraint independence	It is relatively more efficient in path reasoning.
[45]	PEX	CE	Binary	Z3 for both symbolic and path reasoning		its effectiveness relies on good run-time checks in the code or the run-time system.
[43]	SAGE	CE	Binary	code-coverage maximizing heuristic, compositionally (function summaries), Generational Search	tag caching, local constraint caching, and constraint subsumption.	It is relatively more efficient in path reasoning and symbolic reasoning cause it optimizes using caching.
[47]	Tracer	SE	Language	constraint logic programming		relatively less pupolar because language specific and does not use known way of symbolic reasoning
[2]	Rudder	CE	Binary	STP as the solver for symbolic reasoning, and breadth-first search for path reasoning.		one of the most famous among security testing concolic execution tools
[48]	BAP	SE	Binary	SMT solvers	Optimizes intermediate language (IL), making syntaxdirected analysis possible	It does not support floating point and privileged instructions, hence lass accurate
[49]	AEG	CE	Binary	preconditioned symbolic execution and path prioritization technique for path selection.		resistant to buffer overflows, and it an end-to-end fully automated tool
[50]	Mayham	SE	Binary	SMT solver		MAYHEM does not have models for all system/library calls
[51]	Veritesting	SE	Binary	SMT solver for path reasoning		Can do some modern defenses such as canaries
[52]	Firmlice	SE	Binary	concretizing user input		used for modern application testing such mobile and IoT applications
[53]	Driller	SE	Binary	Mutated inputs		supports fuzzing with sysmbolic execution

TABLE 1: Summary of Symbolic Execution Tools.

4. DISCUSSIONS AND FUTURE DIRECTIONS

In this section, potential directions to enhance the state of art of symbolic execution tools are discussed. According to our review, the scalability of such reviewed tools is an open direction that can be taken as future research. Researchers have only slightly worked on the optimizations of both performance and memory space of the tools. Investigating new optimization methods to lower the overhead of symbolic execution and concolic execution tools could make the tools more useable. The well-known path explosion problem is still a main concern of the symbolic execution tools. Therefore, finding a way to limit the paths or possibly reduce the number of less important paths may be helpful to slow down the path explosion problem.

Taking symbolic execution tools that may detect problems, such as authentication bypass, towards cloud and mobile applications could be an interesting future direction. In addition to the symbolic execution engines, SMT solvers are decision procedures that solve problems that arise from the use of logic formulas. SMT solvers are predominately used in the security testing tools, however, software testing tools make little use of these solvers, and their support for non-linear real and integer arithmetic is still in its infancy.

5. CONCLUSION

Without access to source code, binary analysis becomes an effective method for finding vulnerabilities from programs. Researchers have proposed and developed many techniques to help with the binary analysis process, for example, taint analysis, symbolic and concolic executions. In this paper, symbolic and concolic execution techniques are discussed in detail. Tools utilize symbolic and/or concolic execution are reviewed as well. These tools mostly focus on software security testing, and they usually use symbolic execution or concolic execution to help with the test generation and program analysis. The work related to this area is vast and cannot be covered in a single review paper. However, this survey paper discusses well-known and usually referenced tools that cannot be overlooked while studying this area. The comparison table built from the review can be used by researchers in this area to provide a guide to the commonly used tools which employs symbolic and/or concolic execution techniques.

6. REFERENCES

[1] D Andriesse. "*Practical Binary Analysis*", no starch press. 2019.

[2] D Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Prateek Saxena. "*BitBlaze: A new approach to computer security via binary analysis*." in International Conference on Information Systems Security. 2008. Springer.

[3] M von Maltitz, C Diekmann, G Carle. "*Privacy Assessment Using Static Taint Analysis (Tool Paper)*." in International Conference on Formal Techniques for Distributed Objects, Components, and Systems. 2017. Springer.

[4] X Lin, T Chen, T Zhu, K Yang, F Wei, "*Automated forensic analysis of mobile applications on Android devices.*" Digital Investigation, 2018. 26: p. S59-S66.

[5] Z Xing, Z Bin, F Chao, Z Quan. "*Staticly Detect Stack Overflow Vulnerabilities with Taint Analysis.*" in ITM Web of Conferences. 2016. EDP Sciences.

[6] C Feng, X Zhang. "*A Static Taint Detection Method for Stack Overflow Vulnerabilities in Binaries.*" 4th International Conference on Information Science and Control Engineering (ICISCE). 2017. IEEE.

[7] S Chen, J. Xu , N. Nakka, Z. Kalbarczyk, R.K. Iyer. "*Defeating memory corruption attacks via pointer taintedness detection.*" in 2005 International Conference on Dependable Systems and Networks (DSN'05). 2005. IEEE.

[8] GE Suh, JW Lee, D Zhang, S Devadas. "*Secure program execution via dynamic information flow tracking*." in ACM Sigplan Notices. 2004. ACM.

[9] G Venkataramani, Ioannis Doudalis, Yan Solihin, Milos Prvulovic. "*Flexitaint: A programmable accelerator for dynamic taint propagation.*" in 2008 IEEE 14th International Symposium on High Performance Computer Architecture. 2008. IEEE.

[10] J Shin, Hongce Zhang, Jinyong Lee, Ingoo Heo, Yu-Yuan Chen, Ruby Lee, Yunheung Paek. "*A hardware-based technique for efficient implicit information flow tracking.*" in 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). 2016. IEEE.

[11] VP Kemerlis, G Portokalidis, K Jee, AD Keromytis. "*libdft: Practical dynamic data flow tracking for commodity systems.*" in Acm Sigplan Notices. 2012. ACM.

[12] W Xu, S Bhatkar, R Sekar. "*Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks*." in USENIX Security Symposium. 2006.

[13] V Ganesh, T Leek, M Rinard. "*Taint-based directed whitebox fuzzing.*" in Proceedings of the 31st International Conference on Software Engineering. 2009. IEEE Computer Society.

[14] TR Leek, GZ Baker, RE Brown, MA Zhivich. "*Coverage maximization using dynamic taint tracing.*" 2007, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

[15] R Wang, G Xu, X Zeng, X Li, Z Feng. "*TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting*." Journal of Parallel and Distributed Computing, 2018. 118: p. 100-106.

[16] J Clause, W Li, A Orso. "*Dytan: a generic dynamic taint analysis framework.*" in Proceedings of the 2007 international symposium on Software testing and analysis. 2007. ACM.

[17] C Cadar, K Sen. "*Symbolic execution for software testing: three decades later.*" Commun. ACM, 2013. 56(2): p. 82-90.

[18] S Shen, Shweta Shinde, Soundarya Ramesh, Abhik Roychoudhury, Prateek Saxena. "*Neuro-Symbolic Execution: Augmenting Symbolic Execution with Neural Constraints*." in NDSS. 2019.

[19] Q Yi ; Zijiang Yang ; Shengjian Guo ; Chao Wang ; Jian Liu ; Chen Zhao. "*Eliminating path redundancy via postconditioned symbolic execution.*" IEEE Transactions on Software Engineering, 2017. 44(1): p. 25-43.

[20] CS Păsăreanu and W. Visser. "*A survey of new trends in symbolic execution for software testing and analysis.*" International journal on software tools for technology transfer, 2009. 11(4): p. 339.

[21] E Larson, T Austin. "*High Coverage Detection of Input Related Security Faults*," 12th USENIX Sec. 2001. Symposium.

[22] CS Pasareanu, R Kersten, K Luckow, QS Phan. "*Symbolic Execution and Recent Applications to Worst-Case Execution,*" Load Testing and Security Analysis.

[23] C Cadar, P Godefroid, S Khurshid. "*Symbolic execution for software testing in practice: preliminary assessment.*" in 2011 33rd International Conference on Software Engineering (ICSE). 2011. IEEE.

[24] R Baldon, iEmilio Coppa, Daniele Cono D'Elia "*Assisting malware analysis with symbolic execution: A case study.*" in International Conference on Cyber Security Cryptography and Machine Learning. 2017. Springer.

[25] S Khurshid, C.S. Păsăreanu, and W. Visser. "*Generalized symbolic execution for model checking and testing*." in International Conference on Tools and Algorithms for the Construction and Analysis of Systems. 2003. Springer.

[26] X Deng, J Lee "*Bogor/kiasan: A k-bounded symbolic execution for checking strong heap properties of open systems.*" in 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06). 2006. IEEE.

[27] R Baldoni, E Coppa, DC D'elia, C Demetrescu. "*A survey of symbolic execution techniques.*" ACM Computing Surveys (CSUR), 2018. 51(3): p. 50.

[28] H Xu, Z Zhao, Y Zhou, MR Lyu. "*Benchmarking the Capability of Symbolic Execution Tools with Logic Bombs*." IEEE Transactions on Dependable and Secure Computing, 2018.

[29] A Arusoaie, D Lucanu, V Rusu. "*Symbolic execution based on language transformation*." Computer Languages, Systems & Structures, 2015. 44: p. 48-71.

[30] AJ Kahn, Y Drougas, AP Shendarkar. "*Symbolic execution for web application firewall performance.*" 2019, Google Patents.

[31] L Arquint, M Schwerhoff. "*Profiling Symbolic Execution.*" 2019.

[32] T Kuchta, H Palikareva, C Cadar. "*Shadow symbolic execution for testing software patches.*" ACM Transactions on Software Engineering and Methodology (TOSEM), 2018. 27(3): p. 10.

[33] M Liang, Z Li, Q Zeng, Z Fang. "*Deobfuscation of Virtualization-Obfuscated Code Through Symbolic Execution and Compilation Optimization.*" in Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6-8, 2017, Proceedings. 2018. Springer.

[34] S Guo "*Efficient Symbolic Execution of Concurrent Software*." 2019, Virginia Tech.

[35] G Wang, S Chattopadhyay, AK Biswas, T Mitra. "*KLEESPECTRE: Detecting Information Leakage through Speculative Cache Attacks via Symbolic Execution.*" arXiv preprint arXiv:1909.00647, 2019.

[36] C Chen, B Cui, J Ma, R Wu, J Guo, W Liu. "*A systematic review of fuzzing techniques.*" Computers & Security, 2018. 75: p. 118-137.

[37] P Godefroid, N. Klarlund, and K. Sen. "*DART: directed automated random testing."* in ACM Sigplan Notices. 2005. ACM.

[38] AJ Offutt, JH Hayes. "*A semantic model of program faults.*" in ACM SIGSOFT Software Engineering Notes. 1996. ACM.

[39] K Sen, D Marinov, G Agha "*CUTE: a concolic unit testing engine for C.*" in ACM SIGSOFT Software Engineering Notes. 2005. ACM.

[40] R Ahmadi, K Jahed, J Dingel. "*mCUTE: A Model-level Concolic Unit Testing Engine for UML State Machines.*" in 34th IEEE/ACM International Conference on Automated Software Engineering (ASE 2019), Demonstration Track, page to appear. ACM. 2019.

[41] K Sen, G Agha. "*CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools.*" in International Conference on Computer Aided Verification. 2006. Springer.

[42] C Cadar, V Ganesh, PM Pawlowski, DL Dill. "*EXE: automatically generating inputs of death.*" ACM Transactions on Information and System Security (TISSEC), 2008. 12(2): p. 10.

[43] P Godefroid, MY Levin, DA Molnar. "*Automated Whitebox Fuzz Testing.*" in NDSS. 2008. Citeseer.

[44] P Godefroid, MY Levin, D Molnar. "*SAGE: whitebox fuzzing for security testing*." Communications of the ACM, 2012. 55(3): p. 40-44.

[45] N Tillmann, J De Halleux. "*Pex–white box test generation for. net.*" in International conference on tests and proofs. 2008. Springer.

[46] L De Moura, N Bjørner. "*Z3: An efficient SMT solver.*" in International conference on Tools and Algorithms for the Construction and Analysis of Systems. 2008. Springer.

[47] J Jaffar, V Murali, JA Navas, AE Santosa. "*TRACER: A symbolic execution tool for verification.*" in International Conference on Computer Aided Verification. 2012. Springer.

[48] D Brumley, I Jager, T Avgerinos. "*BAP: A binary analysis platform.*" in International Conference on Computer Aided Verification. 2011. Springer.

[49] T Avgerinos, SK Cha, BLT Hao, D Brumley. "*AEG: Automatic exploit generation*." 2011.

[50] SK Cha, T Avgerinos, A Rebert. "*Unleashing mayhem on binary code*." in 2012 IEEE Symposium on Security and Privacy. 2012. IEEE.

[51] T Avgerinos, A Rebert, SK Cha, D Brumley. "*Enhancing symbolic execution with veritesting.*" in Proceedings of the 36th International Conference on Software Engineering. 2014. ACM.

[52] Y Shoshitaishvili, R Wang, C Hauser, C Kruegel. "*Firmalice-automatic detection of authentication bypass vulnerabilities in binary firmware.*" in NDSS. 2015.

[53] N Stephens, J Grosen, C Salls, A Dutcher, R Wang. "*Driller: Augmenting Fuzzing Through Selective Symbolic Execution*." in NDSS. 2016.

Discovering and Understanding The Security Issues In IoT Cloud

Nawaf Almolhis *almo3113@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Jazan University
Jazan, 45142, Saudi Arabia

Michael Haney *mhaney@uidaho.edu*
Computer science department
University of Idaho
Idaho Falls, ID, 83401, USA

Fahad Alqahtani *alqa0199@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Prince Sattam Bin Abdulaziz University
Al-Kharj, 16278, Saudi Arabia

Khalid Makdi *alma0138@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Najran University
Najran, 66223, Saudi Arabia

Abstract

The rapid growth and adoption of IoT technologies in sectors of life are challenged by the resources constrained IoT devices. However, the growth of IoT technologies can be enhanced by integrating them with cloud computing. Hence, a new area of computing called IoT Cloud or CloudIoT has emerged. That is, the data collected from the IoT technologies are stored and processed in the cloud infrastructure so that IoT technologies are relived from resources constrained issue. As a result, some new classes of security and privacy issues are introduced. This paper presents security issues pertaining to IoT cloud.

Keywords: IoT, Cloud, Privacy, Security, CloudIoT.

1. INTRODUCTION

Internet of Things (IoT) is the fast-growing information technology paradigm of this digital era. The number of IoT consumers is increasing due to the deployment of IoT technologies in all sorts of life [1]. At present, the IoT technologies are vastly deployed in the health sector [2, 3], smart cities [4], and smart homes [5, 6]. However, IoT technology alone cannot fully satisfy the increasing number of consumers and their computational requirements. Hence, the need for offloading IoT computations to the cloud has become paramount.

The notion of IoT cloud computing (IoT-Cloud) is concerned with the integration of IoT technologies with cloud computing resources [7-9]. IoT technologies are integrated with cloud mainly for two reasons; first, the IoT providers want to benefit of characteristics of the cloud

computing such as on-demand self-service, resource pooling, broad network, measured service, and rapid elasticity [10]; second, it is for the sake of alleviating the high demands of data storage and processing from the resource-limited IoT technologies [11]. As a result, from a high-level view, IoT technologies appear to be well-integrated with the cloud to establish a uniform infrastructure for IoT cloud applications [12]. This phenomenon of integrating IoT technologies with the cloud is also referred to as the Cloud of Things [13], CloudIoT [14], or Edge IoT [15]. Apart from alleviating the resources constrained behavior, and improving the system performance of IoT technologies, the IoT cloud also enables a new venue of designing and deploying security solutions for IoT technologies [15]. The amalgamation of IoT, cloud, and big data is currently trending [16].

In fact, IoT cloud has come with its own challenges including security issues that may dismay the whole paradigm. IoT cloud security issues are the aggregate of IoT technologies security [17, 18], cloud security [19, 20], and those arising from IoT cloud architecture. This paper surveys security issues that are specific to IoT cloud paradigm, and to our knowledge, it is the first paper of its kind.

The paper is organized as follows, Section 2 presents the background of the research, Section 3 discusses the security challenges related to the IoT cloud, and Section 4 concludes the paper.

2. BACKGROUND

This section discusses areas of intersection of the IoT and Cloud computing. Specifically, the drivers that make the integration of IoT technologies and the cloud more important, and the IoT cloud applications. Furthermore, some of the architectures proposed for IoT cloud are studied. Finally, challenges and issues related to the IoT Cloud are presented.

2.1 IoT Cloud Drivers

IoT and cloud computing are from two different worlds. However, their characteristics are complementary, and that is the main reason why in the literature their integration is seen beneficial for both. That is, IoT can benefit from some aspects of cloud, likewise, IoT can help cloud in some other aspects [21]. For instance, the virtually unlimited resources of cloud can compensate the IoT resource constrains and, IoT can extend cloud services in a more distributed manner and may bring about new real-world service [22]. The driving motivations towards the integration of cloud and IoT mainly lay on three categories including communication, storage, and computing. In communication, data and application sharing are the two main IoT Cloud drivers [23]. In regards to the storage, by definition IoT technologies normally produce large amounts of semi-structured or non-structured data that are generated frequently in large volumes and varieties. Hence, making use of the virtually unlimited storage capacity of the cloud such data can be stored in the cloud. On the other hand, in computing, IoT technologies normally suffer from limited processing and energy resources [24]. These do not allow IoT devices complex data processing. Using cloud computing resources, IoT devices will be able to process data on-site. These are the main motivations that are driving the integration of IoT and Cloud [25]. Table 1 shows aspects where cloud and IoT may complement each other.

Criteria	IoT	Cloud
Displacement	pervasive	centralized
Reachability	limited	ubiquitous
Components	real-world things	virtual resources
Computational Capabilities	limited	virtual unlimited
Storage	limited or none	virtual unlimited
Role of the internet	point of convergence	means of delivering services
Big data	source	means to manage data

TABLE 1: Complementary aspects of Cloud and IoT.

2.2 IoT Cloud Application

IoT cloud paradigm has come with its new sets of applications and smart services most of which were conventionally deployed as a machine to machine communications. This section, discusses, however, the set of applications that have been improved in order to be used in the IoT cloud paradigm. Figure 1 shows an abstract picture of the IoT cloud applications scenario.

2.2.1 Smart Cities

The adoption of the IoT cloud has generated services like smart city applications that can interact with the surrounding environment to create geographic awareness and contextualization opportunities. IoT cloud provides middleware for future-oriented smart city services by collecting information about the geographical location of different sensing technologies and exposing that information uniformly. Most of the current smart application frameworks consist of APIs of sensors and actuators that are directly connected to cloud platforms where they can get scalability, durable storage and processing resources for automatic management and control of large deployments of sensing devices. For example, some researchers have proposed crowdsourced and reputation based smart city frameworks that implement sensing as a service aimed at public safety [26, 27]. Likewise, mobile crowdsensing smart cities technology that uses cloud-based publish/subscribe middleware that collects data from mobile devices are proposed in [28, 29].

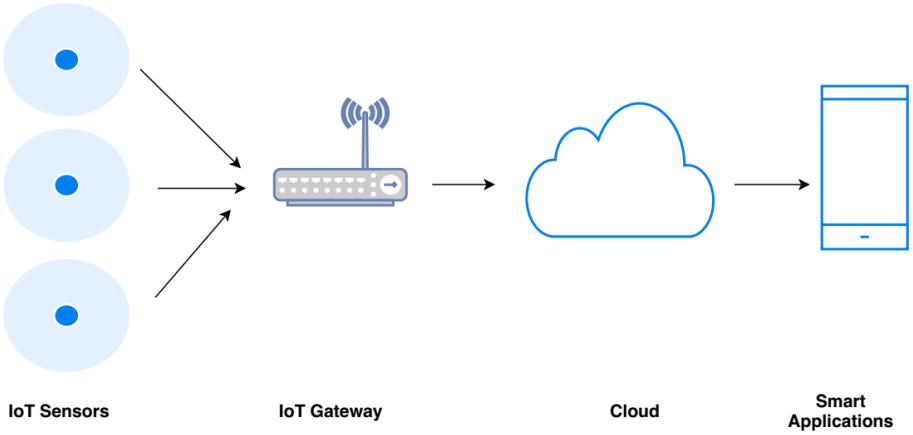


FIGURE 1: IoT Cloud Application Scenario.

2.2.2 Healthcare

In the healthcare industry, things like sensors and devices used for health monitoring are increasing and hugely impacting on patients and health professionals. According to IoT Forbes and Gartner, in 6 years' time from 2016-2020, the healthcare IoT market will be invested with $117 billion [30]. IoT cloud applications are immensely developed for this aspect. Some of the recent works of IoT cloud applications proposed for health care are discussed here. For example, Syed et al. have proposed an asthma patient health monitoring system that connects to the cloud using wireless body area networks [30]. For security, the researchers have watermarked the recorded signal before sending it to the cloud. Douglas et al. also proposed an efficient healthcare IoT cloud architecture for ambient assisted living environments [31]. The advantages of using IoT cloud in healthcare are discussed in [32].

2.2.3 Smart Home

In recent years, high development of smart home applications that use different sensors such as motion, light, and fire detector sensors, etc have been observed. Data collected from these sensors are used for decision making. Hence, like the preceding applications, the necessity of employing IoT cloud sensors in smart homes is becoming mandatory [33]. For instance, Yassine et al. proposed an IoT cloud platform that enables analytics on data captured from smart homes [34]. The proposed data-driven service uses fog nodes and cloud systems for online data

processing, storage, and classification. The researchers employ a policy-based access control mechanism to ensure trusted connectivity and security in their platform.

2.3 IoT Cloud Architecture

There are efforts made towards the definition of a reference architecture for IoT cloud. For example, Jenjira Jaimunk proposed Data Bank, an IoT cloud architecture that allows users to customize their data collection policies at the IoT device level and data sharing policies at the cloud level, as suited to their privacy needs [35]. Similarly, some researchers proposed IoT cloud architecture for different aspects such as for sustainability [36] where the architecture is focusing on low power consumption and environmental friendliness of the things. A generic IoT cloud architecture is provided by Araujo et al. [37], where data collected from a smart city can be stored, processed and managed.

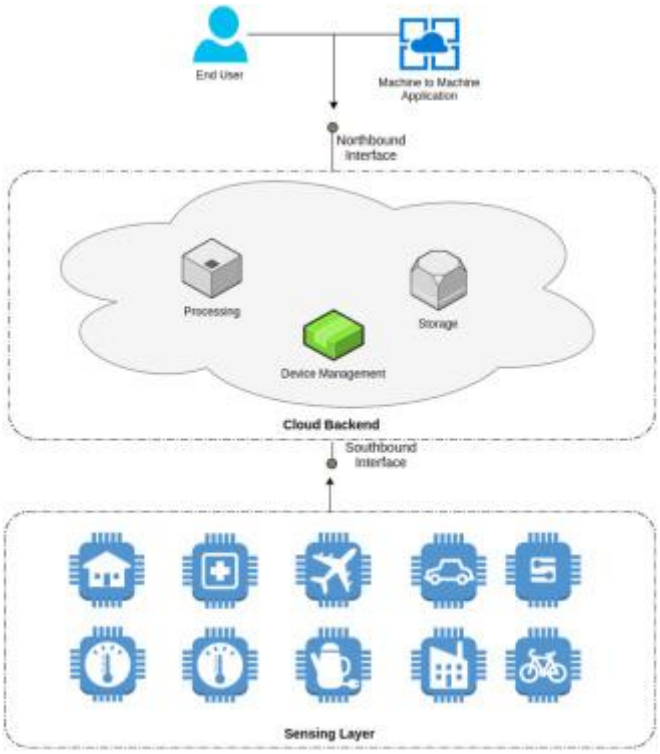


FIGURE 2: A Generic IoT Cloud Architecture [37].

As can be seen in Figure 2, the architecture provides a southbound interface where IoT technologies interact with the cloud and a northbound interface where higher-level services such as M2M applications and end consumers interact with the IoT cloud.

2.4 Security IoT Cloud Challenges

Even though the IoT cloud is advantageous for both consumers and providers, it is still facing some issues that threaten its usage. The heterogeneity of the IoT technologies, clouds, operating systems, network protocols from different vendors generates a more challenging environment that may result in a lack of interoperability and portability in IoT cloud [32, 38]. In addition, in IoT cloud, cloud elasticity and scalability is required. If for instance, IoT cloud provider resources do not meet the increased demand for IoT technologies, interruption or unavailability of the services may result in problem [39]. Security challenges pertaining to the IoT cloud environment are more volatile compared to the security issues in conventional cloud computing. For example, due to the limited resource of IoT technologies, it is not practical to run anti-virus on the IoT devices.

3. SECURITY CHALLENGES IN IOT CLOUDS

After having seen the basics of IoT clouds, this section discusses security challenges within IoT cloud. Such security issues may usually result from different parts of technologies constituting the IoT cloud.

3.1 Security Challenges Related To Data

The data security issues are mainly introduced as the consequence of when smart home owner data are transferred, stored, and processed at clouds that are not part of his network and belong to a third person. The data related security issues that may happen include data loss and data breach. The data loss refers to the data damage that may happen to consumer data. On the other hand, data breach means when the consumer data is taken by an unauthorized individual.

3.2 Offloading Security Challenge

During the transfer process of the data from smart devices to the IoT cloud, access to the cloud is accomplished through wireless networks. Since the consumer does not have access to the data or cannot have control over the data, then there is a risk of unauthorized access to the offloaded content, subsequently, processing of the loaded data is done at the cloud, then there may happen another incident where the integrity of the data is violated.

3.3 Virtualization Security Challenge

The IoT cloud service is provided by using some virtualization techniques. Hence, at the provider side of the IoT cloud, the consumer data is stored and processed on a virtual machine. However, in the cloud, there may be a number of virtual machines abstracted from the same physical server. Hence, a rogue user of a virtual machine may get unauthorized access to a neighboring virtual machine that stores the smart home consumer data.

3.4 IoT Cloud Applications Security Challenges

The security incidents in IoT cloud applications are about compromising the integrity, confidentiality, and availability of both data and applications. Security issues specific to the IoT cloud paradigm are hardly discussed in the literature. Nevertheless, the security challenges of IoT cloud applications may happen at IoT device level, and communication and networking level. Security issues associated with IoT cloud platforms for the smart home is thoroughly discussed in [40]. Likewise, security issues related to the IoT cloud-based healthcare systems can be found in [41].

Another main security issue arises due to a lack of trust in the service provider or the knowledge about service level agreement and knowledge about the physical location of data. Some other security challenges may include heterogeneity, performance, reliability, big data, and monitoring related. For example, the heterogeneity of devices involved in this integrated area may be focusing on the operating system, platform, and services availability [42, 43]. Likewise, those that may come with the performance are those threatening the availability of services such as communication, computation, and storage. Reliability issues may arise when mission-critical applications involving in IoT cloud may suffer from device failure due to a resource-constrained environment[44, 45]. Usually, thousands of smart devices (big data) networked with the cloud would create transportation, storage, access and processing of huge amounts of data that may scrutinize the limited resources of the IoT environment [46, 47]. Having no sophisticated authentication approaches also exacerbates the security associated with IoT cloud. Furthermore, intrusion-related security issues are of most importance for IoT Cloud. In the future as the adoption of cloud-connected IoT technologies increases, security concerns of this area are anticipated to be automatically added on top of currently known security issues.

3.5 IoT Cloud Security Solutions In The Literature

This section presents the current solutions proposed in the literature of IoT cloud. There are a couple of researches that have deliberated to get solutions to the security issues specific to the IoT cloud paradigm [48-64]. Moreover, the summary of the solutions is presented in Table 2. The solutions presented in Table 2 do not include those focusing on the IoT and cloud computing

differently, rather they are the solutions that consider IoT cloud as one area and, hence, trying to propose their solutions in that aspect. Table highlight the security feature in each solution as well as the target area of the paradigm.

3.6 Discussions and Future Directions

Based on the security challenges presented in this paper, it is obvious that security issues pertaining to IoT cloud entail a new set of security challenges from the emerging usage of the paradigm. This new set of security challenges are becoming more difficult to handle for the integration of IoT technologies and cloud. Despite the existence of some security solutions in the literature, there are still some open issues that deserve the attention of the security community. A first secure reference architecture is needed to coin most of the security requirements that IoT cloud need. The cost-effectiveness of the solutions proposed in the literature is not discussed in most cases, hence, the deployment of such solutions is not on the real horizon, thus not cost-effective. In addition, the IoT cloud architecture introduces communications between different technologies. Such communications tend to be secured as well. Here, lightweight secure communication protocols are recommended. There is also a need for algorithms that can create trust between IoT technologies and the cloud. More researches on lightweight solutions for securing virtual machines in the IoT cloud is an added value.

So far researchers have indicated that improving or integrating existing solutions may to some extent handle some of the discussed security issues. However, there is also a need for developing new and dedicated security solutions for the area. What makes different the cloud-connected IoT architecture is that two (cloud and IoT) broadly different areas of technologies are involved. Each has its security issues and challenges where researchers are striving to get solutions. Nevertheless, getting an end to end security solution that would protect personal data collected from IoT end devices and stored or processed in the cloud is alarming.

Converging towards a common security platform for providing APIs to auditing IoT clouds will enable new research efforts in the direction of standard rules and policies that would hold consumers and providers accountable. Similarly, security solutions of IoT cloud would benefit from efficient and flexible technologies that can create a network and virtual machine isolation. Solutions that detects manipulation of data based on IoT clouds will enable enhanced context-based security services.

Source	Title	Focus Area
[48]	Intrusion detection in Cloud Internet of Things Environment	Network security
[49]	A software defined network-based security assessment framework for cloudIoT	Network security
[50]	Secure Self-Destruction of Shared Data in Multi-CloudIoT	Data security
[51]	Secure and Parallel Expressive Search over Encrypted Data with Access Control in Multi-CloudIoT	Data security
[52]	PRTA: A Proxy Re-encryption based Trusted Authorization scheme for nodes on CloudIoT	Access control
[53]	A design of secure communication protocol using RLWE-based homomorphic encryption in IoT convergence cloud environment.	Network security
[54]	A Data Security Storage Method for IoT Under Hadoop Cloud Computing Platform	Data security
[55]	Advanced lightweight multi-factor remote user authentication scheme for cloud-IoT applications	Access control
[56]	Enhancing Cloud-Based IoT Security Through Trustworthy Cloud Service: An Integration of Security and Reputation Approach	Access control
[57]	Ontology-Based Security Context Reasoning for Power IoT-Cloud Security Service	Management
[58]	Privacy-aware IoT cloud survivability for future connected home ecosystem	Privacy
[59]	Security in Lightweight Network Function Virtualisation for Federated Cloud and IoT	Network security
[60]	A Brain-Inspired Trust Management Model to Assure Security in a Cloud Based IoT Framework for Neuroscience Applications	Trust
[61]	A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres	Digital forensics
[62]	A Lightweight User Authentication Scheme for Cloud-IoT Based Healthcare Services	Access control
[63]	IoT–Cloud collaboration to establish a secure connection for lightweight devices	Network security
[64]	Identity-based encryption with authorized equivalence test for cloud-assisted IoT	Access control

TABLE 2: Solutions in Te Literature.

4. CONCLUSION

In this paper, we review the security challenges pertaining to the IoT cloud. The basics of the IoT cloud are reviewed, followed by the security challenges that a consumer may encounter when using smart devices that are connected to the cloud are discussed. Moreover, solutions in the literature are studied and presented. Open security research issues that need immediate attention from the research community are discussed and some prospect solutions that may work conveniently with the IoT cloud paradigm are recommended. Finally, we hope that this paper will be a good entry in enabling a secure integration of IoT technologies and cloud computing.

5. REFERENCES

[1] Alabady, S.A., F. Al-Turjman, and S. Din, *A novel security model for cooperative virtual networks in the IoT era.* International Journal of Parallel Programming, 2018: p. 1-16.

[2] Adhikary, T., et al. *The Internet of Things (IoT) Augmentation in Healthcare: An Application Analytics*. in *International Conference on Intelligent Computing and Communication Technologies*. 2019. Springer.

[3] da Silveira, F., et al., *Analysis of Industry 4.0 Technologies Applied to the Health Sector: Systematic Literature Review*, in *Occupational and Environmental Safety and Health*. 2019, Springer. p. 701-709.

[4] Arasteh, H., et al. *Iot-based smart cities: a survey*. in *2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC)*. 2016. IEEE.

[5] Park, D.-M., S.-K. Kim, and Y.-S. Seo, *S-mote: SMART Home Framework for Common Household Appliances in IoT Network.* Journal of Information Processing Systems, 2019. 15(2).

[6] Mahmud, S., S. Ahmed, and K. Shikder. *A Smart Home Automation and Metering System using Internet of Things (IoT)*. in *2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*. 2019. IEEE.

[7] Mohamed, K.S., *IoT Cloud Computing, Storage, and Data Analytics*, in *The Era of Internet of Things*. 2019, Springer. p. 71-91.

[8] Zamora-Izquierdo, M.A., et al., *Smart farming IoT platform based on edge and cloud computing.* Biosystems engineering, 2019. 177: p. 4-17.

[9] Bhawiyuga, A., et al., *Architectural design of IoT-cloud computing integration platform.* Telkomnika, 2019. 17(3).

[10] Mircea, M., M. Stoica, and B. Ghilic-Micu, *Using Cloud Computing to Address Challenges Raised by the Internet of Things*, in *Connected Environments for the Internet of Things*. 2017, Springer. p. 63-82.

[11] Ali, Z.H., H.A. Ali, and M.M. Badawy, *A new proposed the internet of things (IoT) virtualization framework based on sensor-as-a-service concept.* Wireless Personal Communications, 2017. 97(1): p. 1419-1443.

[12] Nikolov, N. and O. Nakov. *Creating Architecture and Software of Embedded Systems with Constrained Resources and Their Communication to the IoT Cloud*. in *2019 X National Conference with International Participation (ELECTRONICA)*. 2019. IEEE.

[13] Aazam, M., et al. *Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved*. in *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014*. 2014. IEEE.

[14] Serrano, D., et al. *Towards qos-oriented sla guarantees for online cloud services*. in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 2013. IEEE.

[15] Sha, K., et al., *A survey of edge computing based designs for IoT security.* Digital Communications and Networks, 2019.

[16] Sharma, S., et al., *Cloud and IoT-based emerging services systems.* Cluster Computing, 2019. 22(1): p. 71-91.

[17] Hassan, W.H., *Current research on Internet of Things (IoT) security: A survey.* Computer Networks, 2019. 148: p. 283-294.

[18] Khan, M.A. and K. Salah, *IoT security: Review, blockchain solutions, and open challenges.* Future Generation Computer Systems, 2018. 82: p. 395-411.

[19] Kumar, R. and R. Goyal, *On cloud security requirements, threats, vulnerabilities and countermeasures: A survey.* Computer Science Review, 2019. 33: p. 1-48.

[20] Wang, Z., et al., *An empirical study on business analytics affordances enhancing the management of cloud computing data security.* International Journal of Information Management, 2019.

[21] Gomes, M.M., R.d.R. Righi, and C.A. da Costa. *Future directions for providing better IoT infrastructure*. in *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: Adjunct Publication*. 2014. ACM.

[22] Alhakbani, N., et al. *A framework of adaptive interaction support in cloud-based internet of things (iot) environment*. in *International conference on internet and distributed computing systems*. 2014. Springer.

[23] Aitken, R., et al. *Device and technology implications of the Internet of Things*. in *2014 Symposium on VLSI Technology (VLSI-Technology): Digest of Technical Papers*. 2014. IEEE.

[24] Botta, A., et al., *Integration of cloud computing and internet of things: a survey.* Future generation computer systems, 2016. 56: p. 684-700.

[25] Ray, P.P., *A survey of IoT cloud platforms.* Future Computing and Informatics Journal, 2016. 1(1-2): p. 35-46.

[26] Kantarci, B. and H.T. Mouftah. *Mobility-aware trustworthy crowdsourcing in cloud-centric Internet of Things*. in *2014 IEEE Symposium on Computers and Communications (ISCC)*. 2014. IEEE.

[27] Kantarci, B. and H.T. Mouftah, *Trustworthy sensing for public safety in cloud-centric internet of things.* IEEE Internet of Things Journal, 2014. 1(4): p. 360-368.

[28] Podnar Zarko, I., A. Antonic, and K. Pripužic. *Publish/subscribe middleware for energy-efficient mobile crowdsensing*. in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. 2013. ACM.

[29] Antonic, A., et al. *A mobile crowdsensing ecosystem enabled by a cloud-based publish/subscribe middleware*. in *2014 International Conference on Future Internet of Things and Cloud*. 2014. IEEE.

[30] Shah, S.T.U., et al., *Cloud-Assisted IoT-Based Smart Respiratory Monitoring System for Asthma Patients*, in *Applications of Intelligent Technologies in Healthcare*. 2019, Springer. p. 77-86.

[31] de Macedo, D.D.J., et al., *Toward an efficient healthcare CloudIoT architecture by using a game theory approach.* Concurrent Engineering, 2019: p. 1063293X19844548.

[32] Darwish, A., et al., *The impact of the hybrid platform of internet of things and cloud computing on healthcare systems: Opportunities, challenges, and open problems.* Journal of Ambient Intelligence and Humanized Computing, 2019. 10(10): p. 4151-4166.

[33] Madhu, B., et al., *IoT Based Home Automation System over Cloud.* 2019.

[34] Yassine, A., et al., *IoT big data analytics for smart homes with fog and cloud computing.* Future Generation Computer Systems, 2019. 91: p. 563-573.

[35] Jaimunk, J. *Privacy-preserving cloud-IoT architecture*. in *Proceedings of the 6th International Conference on Mobile Software Engineering and Systems*. 2019. IEEE Press.

[36] Singh, A., U. Sinha, and D.K. Sharma, *Cloud-Based IoT Architecture in Green Buildings*, in *Green Building Management and Smart Automation*. 2020, IGI Global. p. 164-183.

[37] Araujo, V., et al., *Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities.* Journal of Parallel and Distributed Computing, 2019.

[38] Kanchi, R.R., V.P. Sreeramula, and D.V. Palle. *Implementation of Smart Agriculture using CloudIoT and its Geotagging on Android Platform*. in *International Conference on Intelligent Computing and Communication Technologies*. 2019. Springer.

[39] Malik, A. and H. Om, *Cloud computing and internet of things integration: Architecture, applications, issues, and challenges*, in *Sustainable Cloud and Energy Services*. 2018, Springer. p. 1-24.

[40] Zhou, W., et al. *Discovering and understanding the security hazards in the interactions between IoT devices, mobile apps, and clouds on smart home platforms*. in *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 2019.

[41] Ahmed, A., et al., *Malicious insiders attack in IoT based multi-cloud e-healthcare environment: a systematic literature review.* Multimedia Tools and Applications, 2018. 77(17): p. 21947-21965.

[42] Grozev, N. and R. Buyya, *Inter-Cloud architectures and application brokering: taxonomy and survey.* Software: Practice and Experience, 2014. 44(3): p. 369-390.

[43] Moussa, A.N., et al. *A Consumer-Oriented Cloud Forensic Process Model*. in *2019 IEEE 10th Control and System Graduate Research Colloquium (ICSGRC)*. 2019. IEEE.

[44] Rao, B.P., et al. *Cloud computing for Internet of Things & sensing based applications*. in *2012 Sixth International Conference on Sensing Technology (ICST)*. 2012. IEEE.

[45] Moussa, A.N., N. Ithnin, and A. Zainal, *CFaaS: bilaterally agreed evidence collection.* Journal of Cloud Computing, 2018. 7(1): p. 1.

[46] Dobre, C. and F. Xhafa, *Intelligent services for big data science.* Future Generation Computer Systems, 2014. 37: p. 267-281.

[47] Moussa, A.N., N.B. Ithnin, and O.A. Miaikil. *Conceptual forensic readiness framework for infrastructure as a service consumers*. in *2014 IEEE Conference on Systems, Process and Control (ICSPC 2014)*. 2014. IEEE.

[48] Rebbah, M., D.E.H. Rebbah, and O. Smail. *Intrusion detection in Cloud Internet of Things environment*. in *2017 International Conference on Mathematics and Information Technology (ICMIT)*. 2017. IEEE.

[49] Han, Z., et al., *A software defined network-based security assessment framework for cloudIoT.* IEEE Internet of Things Journal, 2018. 5(3): p. 1424-1434.

[50] Guechi, F.A. and R. Maamri. *Secure Self-Destruction of Shared Data in Multi-CloudIoT*. in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017. IEEE.

[51] Guechi, F.A. and R. Maamri. *Secure and Parallel Expressive Search over Encrypted Data with Access Control in Multi-CloudIoT*. in *2018 3rd Cloudification of the Internet of Things (CIoT)*. 2018. IEEE.

[52] Su, M., et al., *PRTA: A Proxy Re-encryption based Trusted Authorization scheme for nodes on CloudIoT.* Information Sciences, 2019.

[53] Jin, B.-W., J.-O. Park, and H.-J. Mun, *A design of secure communication protocol using RLWE-based homomorphic encryption in IoT convergence cloud environment.* Wireless Personal Communications, 2019. 105(2): p. 599-618.

[54] Mo, Y., *A Data Security Storage Method for IoT Under Hadoop Cloud Computing Platform.* International Journal of Wireless Information Networks, 2019: p. 1-6.

[55] Sharma, G. and S. Kalra, *Advanced lightweight multi-factor remote user authentication scheme for cloud-IoT applications.* Journal of Ambient Intelligence and Humanized Computing, 2019: p. 1-24.

[56] Li, X., et al., *Enhancing cloud-based IoT security through trustworthy cloud service: An integration of security and reputation approach.* IEEE Access, 2019. 7: p. 9368-9383.

[57] Choi, C. and J. Choi, *Ontology-Based Security Context Reasoning for Power IoT-Cloud Security Service.* IEEE Access, 2019. 7: p. 110510-110517.

[58] Arabo, A. *Privacy-aware IoT cloud survivability for future connected home ecosystem*. in *2014 IEEE/ACS 11th International Conference on Computer Systems and Applications (AICCSA)*. 2014. IEEE.

[59] Massonet, P., et al. *Security in lightweight network function virtualisation for federated cloud and IoT*. in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017. IEEE.

[60] Mahmud, M., et al., *A brain-inspired trust management model to assure security in a cloud based IoT framework for neuroscience applications.* Cognitive Computation, 2018. 10(5): p. 864-873.

[61] Gupta, P.K., B.T. Maharaj, and R. Malekian, *A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres.* Multimedia Tools and Applications, 2017. 76(18): p. 18489-18512.

[62] Sharma, G. and S. Kalra, *A Lightweight User Authentication Scheme for Cloud-IoT Based Healthcare Services.* Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 2019. 43(1): p. 619-636.

[63] Park, J., H. Kwon, and N. Kang, *IoT–Cloud collaboration to establish a secure connection for lightweight devices.* Wireless Networks, 2017. 23(3): p. 681-692.

[64] Elhabob, R., et al., *Identity-based encryption with authorized equivalence test for cloud-assisted IoT.* Cluster Computing, 2019: p. 1-17.

Web Based Access Control of Smart Home Security System

Khalid Aloufi *koufi@taibahu.edu.sa*
College of Computer Science & Engineering
Taibah University
Madinah, Saudi Arabia

Ahmed Alharbi *amadsas2@taibahu.edu.sa*
College of Computer Science & Engineering
Taibah University
Madinah, Saudi Arabia

Anwar Redwan *anwar_nory@hotmail.com*
College of Computer Science & Engineering
Taibah University
Madinah, Saudi Arabia

Yousif AbuTarboush *gh.yousif@gmail.com*
College of Computer Science & Engineering
Taibah University
Madinah, Saudi Arabia

Abstract

Smart Home is an essential feature for future homes to provide smart automatic services in home daily activities. Smart Home system Framework will require different units to work, such as security system, maintenance system, kitchen system, living room system and other main part of the home. These unit should complement each other's. This work is proposing a solution to the design and implementation of Smart home security system (SHSS) unit. With the Internet of Things (IoT), SHSS has become simpler to design and implement. The control and management of such system will require an interface. The web is an excellent option as an interface with secure access. Integrating the system with the web increases the operation of the system and the interaction with other security department for the future Smart City. In this work, the design and implementation of SHSS is shown. The model has successfully been built and ready for the integration with more units of the smart home and smart city.

Keywords: IoT, Smart Home, MQTT, Web Services, Home Security Systems.

1. INTRODUCTION

With the growth of cities and population size, regular systems and services are not enough. Smart city framework is required to fulfill this almost currently and surely a future demanded service. Table 1 shows the Smart city framework units, one of which is the smart home. Smart home has also different units, such as security system that is should be part of the smart city in the future. Smart Home will provide services, that performs decisions without reference to human, which could be enhanced by an Artificial System (AI) that observe the habits of persons and take actions accordingly. This paper presents a Smart home security system (SHSS) design and implementation. SHSS is using embedded systems, Internet of Things (IoT) and web interface to monitor and control the system. SSHS and any smart city application should increase response performance for services. Time is a main factor in terms of home security, such as smoke alarm or motion detector. SHSS helps in smoke or toxic gases detection, which could be caused by a careless actions or faulty Electrical Equipment. This work present smart solutions to in short time,

discover problems, verify them and act without human intervention. Which helps us to prevent or reduce damage and speed of rescue.

IoT services can be classifieds in static or dynamic services. Static services start with user request only, such as door entry using mobile application or using finger barometric input device or by voice command. In general, IoT devices should work with more than one method to avoid errors in any possible case. After the introduction, a section about Previous studies and System Design and Implementation is presented. Then a section about results and discussion to discuss the system implementation. The paper closes by Conclusion and Future Work and the references referred to in this paper.

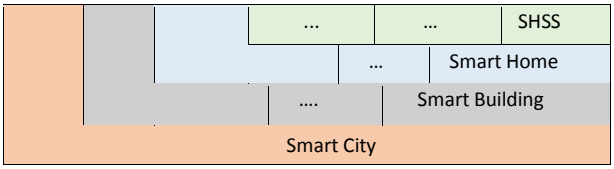


TABLE 1: Smart City Framework.

2.1 Previous Studies

IoT application uses different application protocols, mainly MQTT and Constrained Application Protocol (CoAP) with different enhanced security and operational schema over cloud and edge computing [6] [4]. There are great range of research projects and commercial Smart Home projects, using different types of sensors and actuators, such as camera microphones and Wireless Smoke Detector [19]. Intelligent Door Lock project connects the door of the house to mobile application with security to open a remote door for those trusted [12]. Home motion detector project for motion detection within a household [11]. In Smart doorbell project, the bell can be heard from anywhere connected [8]. One of the IoT application uses IoT for engagement of elderly people in daily activities [18] Also other examples are light bulbs and temperature Dynamic services starts without user request, such as opening Garage, Automatic entry door, light bulbs to operate according to time intervals [17].

Smart system has many operational and functional details and components with support of Artificial intelligence [20]. Embedded system" is not related to hardware alone or software but related to all of them. It is a computer made to do specific function. Therefore, it is compiled system of hardware and software designed to make function [16]. Embedded System Must be Reliable, maintainable and efficient [13]. Embedded systems naturally contain of diverse components, Sensors, Elements transmit, Command-and-control units, Actuators [14]. Different kinds and features of Embedded System can be used in IoT application, such as Raspberry pi and Arduino [1] [10] [5]. Smart homes have some Challenges, such implementation cost, technical awareness and insufficient technical support to maintain, operate and work with these systems [7]. The Internet of Things Technologies has been tested as a use Case Study of Smart Malls [3]. One of the projects presented a Scalable Monitoring System (WiSe-SMS) using IoT without using an application protocol [2].

Layer	OSI Layers	TCP/IP	IoT
7	Application	Application	MQTT
6	Presentation		
5	Session		
4	Transport	Transport	TCP
3	Network	Internet	IP
2	Data Link	Network Access	IEEE 802.15.4 MAC
1	Physical		IEEE 802.15.4 PYS

TABLE 2: Internet Of Things Stack.

2. SYSTEM DESIGN AND IMPLEMENTATION

SHSS is designed using Raspberry Pi 3 Model B and Arduino. The programming language used is Python, is the official programming language for raspberry pi. Message Queue Telemetry Transport (MQTT) is publish/subscribe and lightweight messaging protocol. Table 2 shows the MQTT stack model. A client receives messages from a sensor by subscribing to the topic on the same sensor. In this model, there is no straight connection between a publisher (sensor) and subscriber (client or actuator). The system use cases is shown in Figure 1 and 2. For instance, two cases that almost follow the same actions, but has a different reactions as shown in figure 2 and 3. Following the smart procedure, when there is an intruder, the system takes a reaction without human interference. When smoke is detected by a smoke detector, the reaction will be sent to the Civil Defense to take the message and send a rescue team. Home design is miniature design for home as shown in Figure 3.

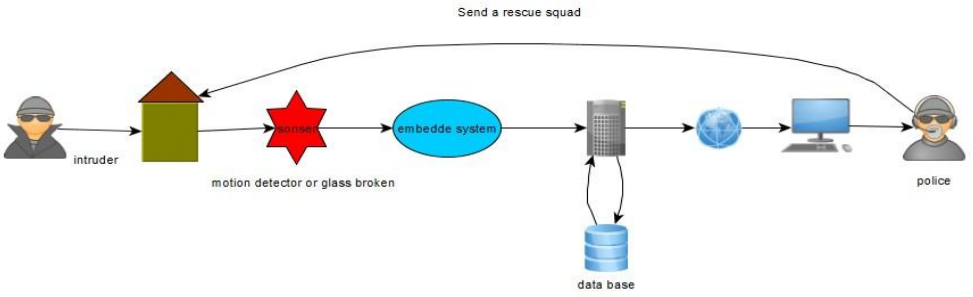


FIGURE 1: Case 1" Intruder".

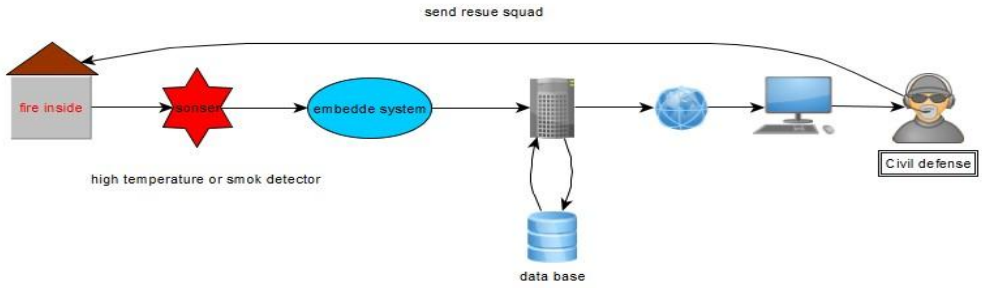


FIGURE 2: Case 2" Intruder".

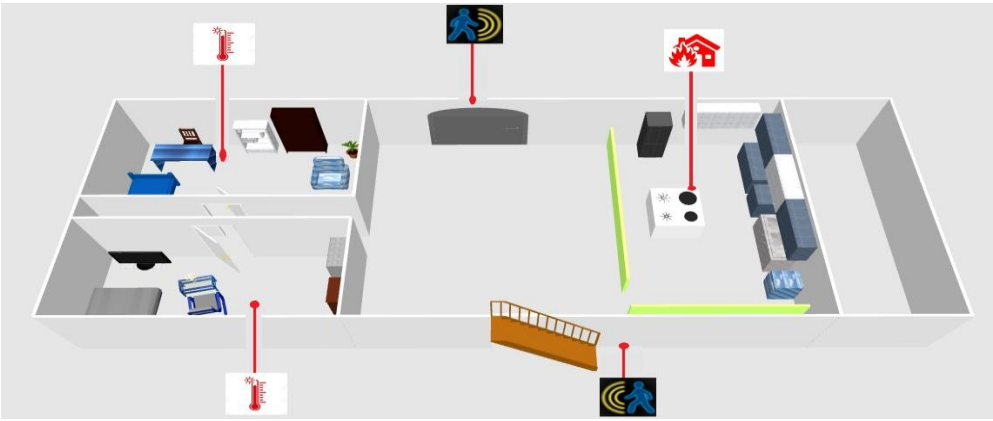
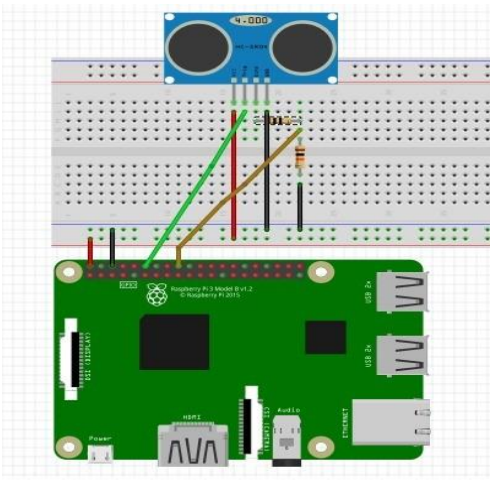


FIGURE 3: Home Design.

The system consists of Ultrasonic sensor, temperature and humidity sensor, Gas sensor, Raspberry pi ,Raspbian OS, Power Supply, 2A microB USB power supply, HDMI cable to Connect Raspberry Pi to a Screen, Monitor as a display for the raspberry pi, Keyboard, Mouse

and SD card Reader .SD Card and microSD card are used to store files and software to use on your Raspberry Pi, such as NOOBS and Raspbian. ”Mosquitto” distribution is used as MQTT broker and client with access control by a username and password [15]. Python web framework called Flask to shows up the Raspberry Pi into a dynamic web server [9]. Ultrasonic is a sensor that measure the distance to an object by using sound waves. Connection is shown in figure 4(a) and output is shown in figure 4(b). Connection of the temperature and humidity sensor is shown in figure 5(a) and output is shown in figure 5(b). Connecting the Gas sensor as shown in figure **6(a)** and the output is shown in figure **6**(b).

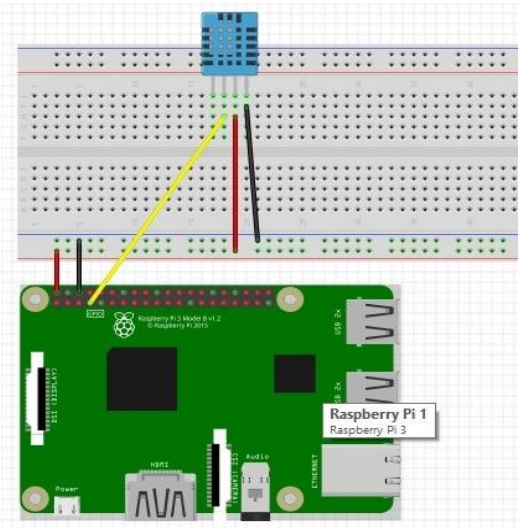


(a): Connection of Ultrasonic.

```
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for
>>>
================== RESTART: /home/pi/ultrasonic
Measured Distance = 3.6187 cm
Measured Distance = 3.6023 cm
Measured Distance = 3.5819 cm
Measured Distance = 3.6187 cm
Measured Distance = 3.6595 cm
Measured Distance = 3.6187 cm
Measured Distance = 3.6391 cm
Measured Distance = 3.6555 cm
Measured Distance = 3.5982 cm
Measured Distance = 3.6800 cm
Measured Distance = 3.6595 cm
Measured Distance = 3.6514 cm
Measured Distance = 3.5083 cm
```

(b): Output of Ultrasonic.

FIGURE 4: Ultrasonic Sensor.

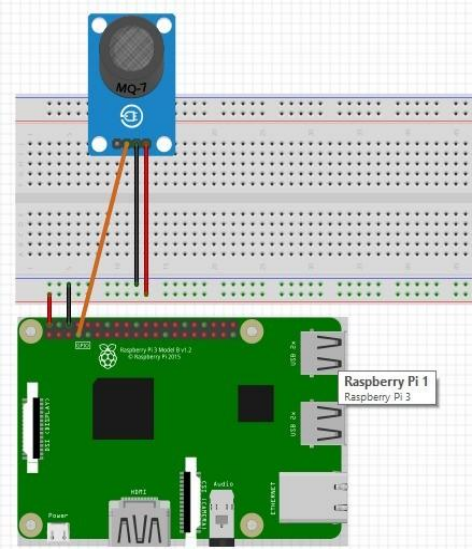


(a): Connection of Temperature Sensor.

```
Python 3.5.3 (default, Jan 19 2017, 14:11
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()
>>>
====================== RESTART: /home/pi/
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
Humidity = 16.0 %; Temperature = 23.0 C
```

(b): Output of Temperature Sensor.

FIGURE 5: Temperature Sensor.



(a): Connection of Gas Sensor.

```
alive
Sensor detected action!
Sensor detected action!
alive
alive
alive
Sensor detected action!
Sensor detected action!
alive
alive
alive
alive
alive
alive
alive
alive
alive
alive
alive
```

(b): Output of Gas Sensor.

FIGURE 6: Gas Sensor.

3. RESULTS AND DISCUSSION

We have three parts of SHSS system, home, police and the other for civil defense. The home requires watching for two entrances and two rooms and one kitchen. If there is a warning it will show that to the police or civil defense, which should be using MQTT. The main Home Interface will look like the page shown in figure 7.

In each icon there is a sensor that can detect something like gas, motion or high temperature. For Entrance 1 and 2, a limiting distance is set for the window sensor to alert the user and the Police Department as shown in figure 8(a) and 8(b) for entrance 1 and 2, respectively. If the sensor detects a motion for less than 15 cm, a warning is sent to the homeowner and the police. A pop-up massage is displayed for entrance 1 as shown in figure 9 and the same applied for entrance 2.

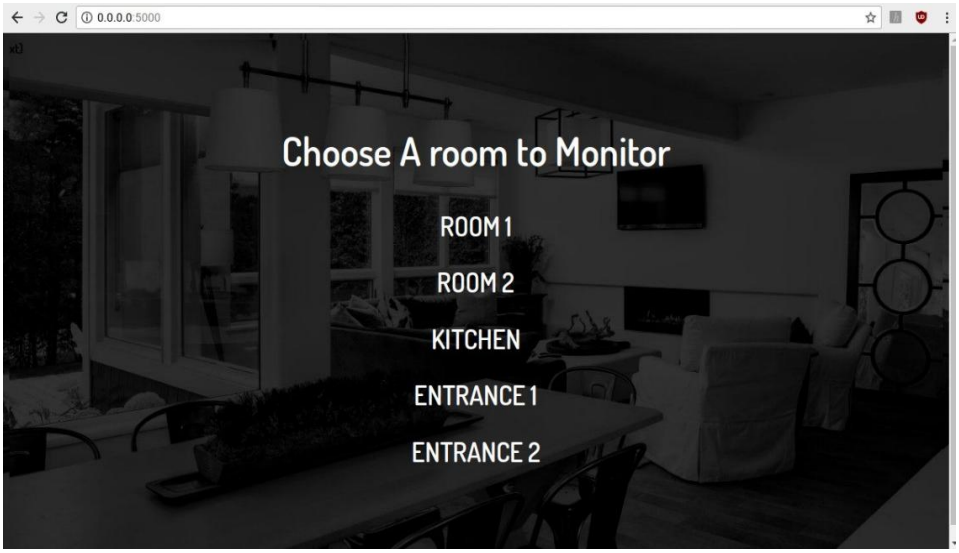
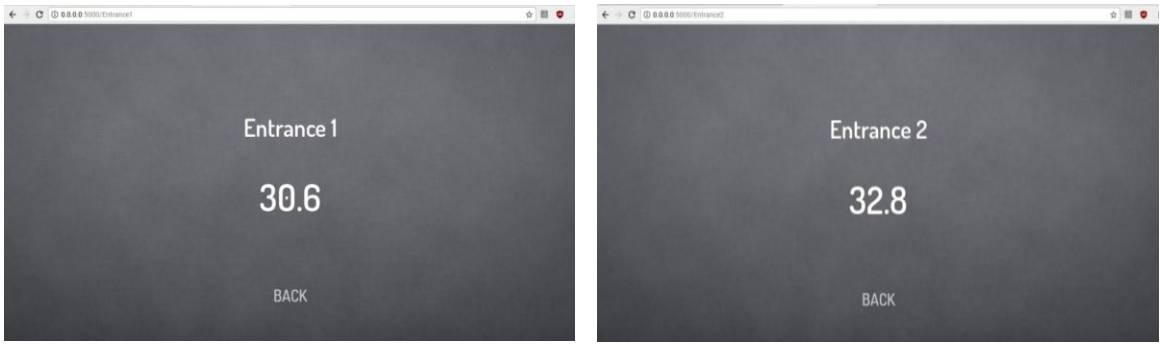


FIGURE 7: The Main Home Interface.



(a): Entrance 1 Interface. **(b):** Entrance 2 Interface.

FIGURE 8: Entrance Interface.

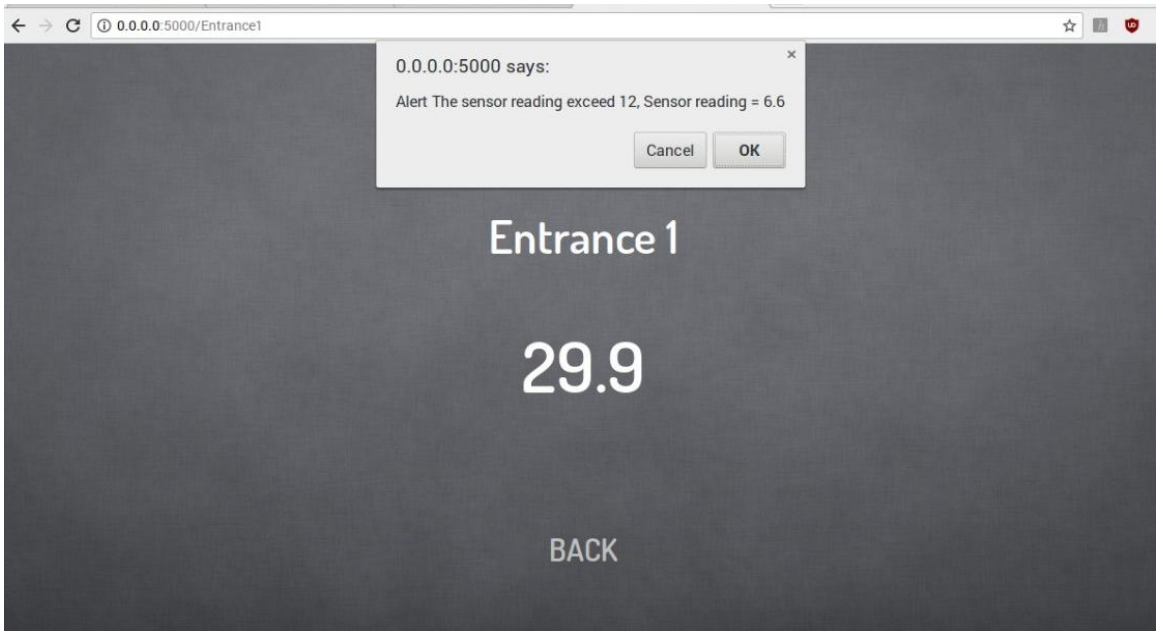
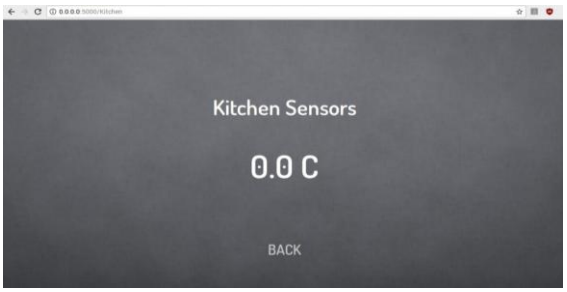


FIGURE 9: Entrance 1 with Pop Up Window.



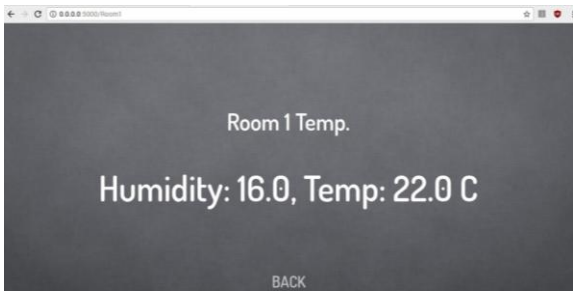


FIGURE 10: Kitchen Interface. **FIGURE 11:** Room 1 Interface.

For the Kitchen, there is a gas sensor connect that detect the Gas Leak. It will open a pop-up window in case of warning. Also, the Temperature and humidity sensor shows pop up in case of warning as shown in figure 10. If the sensor detect gas, it will turn to 1. For Room 1, there is a Temperature and humidity sensor. We have shown pop up the Temperature and humidity of the

room to show the information to the homeowner as shown in figure 11. For Room 2, the same as the Room 1.

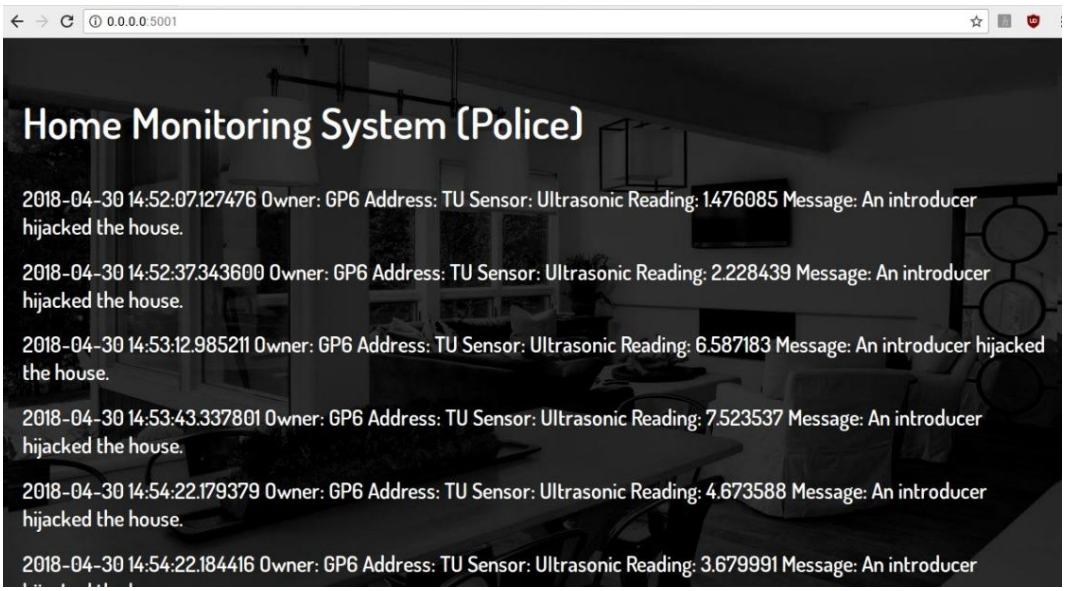


FIGURE 12: Police Page Interface.

The system will contact the police by warning message as shown in figure 12. It will display the time and the date and the location address for the home for every warning. For the civil defense, the same as the police warning messages but with different sensors. The main goals of this work are achieved by running the web page and python code using raspberry pi and reading the sensor and display it in the web page and send the warning to the police and civil defense.

4. CONCLUSION AND FUTURE WORK

Home security solutions is essential in the future of smart city. With the IoT popularity, Home security systems are expected to decrease in cost and increase in availability and performance. IoT enhanced by the cloud computing and web services, such system is expected to get simpler to install and implement making such studies are very important. The integration between IoT and application protocols, such as MQTT, and Web servers open wide range of applications and open the doors for using web technologies based on what so called Web of Things (WoT). One of the main challenges of the smart city and smart home application is the structured data integration between separate system from different vendor. Such System can be integrated with other systems using the Semantic Web (SW) with the support with an ontology to help the integration with other system as one system.

Connect the sensors to the Web with dynamic web page, and pop up message, help in increasing the dynamic interactions between machine and human. As well as the connection between different systems, such as the police and other departments to provide easy secure systems integration. Such research opens great range of research and commercial challenges for such systems. No one corporation with one solution will be able to provide complete solution for smart city application.

Therefore, it is very clear that the only solution is to agree on messages transaction between units of the smart city framework. Implementation is not a problem if different from system to system. To complete such mission, the governance of such mega projects need to specify the machine to machine and machine to human interaction and the technologies associated such as SW.

5. REFERENCES

[1] AbdAllah, A.A.: Simple raspberry pi,easy learning (2014).

[2] Aboelela, E., Aloufi, K., Atta, R.: Design and implementation of a wireless sensor network based scalable monitoring system (wise-sms). Journal of Computers 13, 244–261 (2018). DOI 10.17706/jcp.13.3.244-261.

[3] Algarni, F., Ullah, A., Aloufi, K.: Enhancing the Linguistic Landscape with the Proper Deployment of the Internet of Things Technologies: A Case Study of Smart Malls, pp. 13–39 (2020). DOI 10.1007/978-3-030-32523-7 2.

[4] Alhazmi, O.H., Aloufi, K.S.: Fog-based internet of things: A security scheme. In: 2019 2nd International Conference on Computer Applications Information Security (ICCAIS), pp. 1–6 (2019). DOI 10.1109/CAIS.2019.8769506.

[5] Aloufi, K.: 6lowpan stack model configuration for iot streaming data transmission over coap. International Journal of Communication Networks and Information Security 11, 304–3012 (2019).

[6] Amairah, A., Al-tamimi, B.N., Anbar, M., Aloufi, K.: Cloud computing and internet of things integration systems: A review. In: F. Saeed, N. Gazem, F. Mohammed, A. Busalim (eds.) Recent Trends in Data Science and Soft Computing, pp. 406–414. Springer International Publishing, Cham (2019).

[7] carbontrack: The challenges of security for iot and home automation. http://carbontrack.com.au/blog/challenges-security-iot-home-automation/ (2017).

[8] Caroline El Fiorenza S. Madhumita, S.B.M.G.A.A.: Iot smart doorbell surveillance. Internation Journal Of Advance Research And Innovative Ideas In Education 4(2), 2701–2706 (2018).

[9] DuPlain, R.: Flask Web Development (2013).

[10] Gus: What is a raspberry pi computer. https://pimylifeup.com/what-is-raspberry-pi/ (2019).

[11] hackster: Home motion detector. https://www.hackster.io/65860/home-motiondetector-28f965?ref=tag&ref id=home securit (2017).

[12] hackster: Smartphone connected home door lock. https://www.hackster.io/hackershack/smartphone-connected-home-door-lock-69944f (2017).

[13] Kopetz, H.: Real-time systems - design principles for distributed embedded applications. In: The Kluwer international series in engineering and computer science (1997).

[14] Lee, Seshia: Introduction to embedded systems, a cyber-physical systems approach, second edition, mit press, isbn 978-0-262-53381-2 (2017).

[15] Light, R.: Mosquitto: server and client implementation of the mqtt protocol. The Journal of Open Source Software 2 (2017). DOI 10.21105/joss.00265.

[16] Nasir, S.Z.: Real life examples of embedded systems. https://www.theengineeringprojects.com/2016/11/examples-of-embeddedsystems.html (2016).

[17] safewise team: What is a security system and how does it work? https://www.safewise.com/home-security-faq/how-do-security-systems-work/ (2019).

[18] Thakur, N., Han, C.: Framework for an intelligent affect aware smart home environment for elderly people. pp. 23 – 43. International Journal of Recent Trends in Human Computer Interaction (IJHCI) (2019).

[19] Vivint: Vivint doorbell camera. https://www.vivint.com/products/doorbell-camera (2017).

[20] Wikipedia: Smart system. https://en.wikipedia.org/wiki/Smart system (2017).

A Comparison of Queueing Algorithms Over TCP Protocol

Mahmud Mansour *mm.mansour@gmail.com*
Faculty of Information Technology
Department of Network Engineering
Tripoli University
Tripoli, Libya

Ahmed Hmeed *A.Hmeed@hotmail.com*
Faculty of Information Technology
Department of Network Engineering
Tripoli University
Tripoli, Libya

Abstract

Network congestion control is one of the most key problems in network study. With the expansion of network size, the continuous increase of network bandwidth and increasing diversification of networking forms, congestion control has encountered some new problems that require a solution.

If a packet number that reaches the network is greater than the processing capacity of network, network performance would drop dramatically, resulting in an inevitable congestion. In order to avoid congestion, people use congestion control algorithm in the network.

This paper studies different versions of TCP source algorithms, such as Reno and Vegas, and investigate the impact of various Queuing management algorithms on the self-similarity of network traffic. We compare the performance of Reno and Vegas using various queue management algorithms, namely Droptail, Fair Queueing (FQ), Deficit Round Robin (DRR) and Random Early Detection (RED) using NS-2 network simulators. The characteristics of different algorithms are also discussed and compared based on the basis of packet loss, fairness and throughput metric.

Keywords- Congestion Control, TCP Reno, TCP New Reno, TCP Vegas, Transmission Control Protocol /Internet Protocol (TCP/IP).

1. INTRODUCTION

Internet has experienced exploding growth since its emergence. Internet traffic keeps growing at an exponential rate of almost doubling itself each year; and this trend is expected to continue [1]. Various and vast amount of Internet-based applications and services emerge with this growth and surveys reveal trend towards them. More and more people come to depend on them, and all kinds of business processes are built around them. And along with the emergence of Next Generation Network (NGN) services Internet is entering every home and business groups, and Internet-based applications and services are pervading everyday life.

In return, people and business groups unceasingly bring forth all kinds of new demands. They not only ask for diversified applications and services to satisfy their needs, but also demand for better quality of service (QoS). These different applications and services have varying requirements for goodput, packet loss ratio and end-to-end latency [2]. As surveys reveal, time-critical and mission-critical applications are rapidly growing to be a significant portion of Internet-based applications. In pursuing of greater profit, delay as one of the important performance indices of

quality of service is becoming more and more recognized and emphasized by Internet service providers.

The transport layer providers duplex, end-to-end data transport services between applications. Data sent from the application layer ware divided into segment appropriate in size for the network technology. Transmission control protocol (TCP) and user datagram protocol (UDP) are the protocols used at this layer [1].

Transmission control protocol (TCP) provides reliable service by being connection-oriented and including error detection and correction. The connected nature of TCP used only for two end-points to communicate with each other. The connection must be established before a data transfer can occur, and transfers are acknowledged throughout the process. Acknowledgments assure that data received properly [6]. The acknowledgment process provides robustness in the face of network congestion or communication unreliability. It also determines when the transfer ends and closes the connection, thus freeing up resources on the systems. Checksums assure that the data not accidentally modified during transit [3].

Traffic management in TCP examines the reality of two autonomous methods: Delivery control regulated by the recipient using the window specification and Congestion control regulated by the sender for employed the congestion window and slow begin method. The first method oversees the recipient input buffer and the second method registers the channel congestion, hence it helps to decrease the level of traffic. The Congestion Window (CWND) and slow start method gives resolve the full loading of the virtual connection and decreasing the packet loss in case of overloading in the network [4].

2. TCP CONGESTION CONTROL ALGORITHM

The basis of TCP congestion control lies in Additive Increase Multiplicative Decrease (AIMD), halving the congestion window for every window containing a packet loss, and increasing the congestion window by roughly one segment per Round Trip Time (RTT) otherwise.

The second component of TCP congestion control is the Retransmit Timer, including the exponential bakeoffs of the retransmit timer when a retransmitted packet is itself dropped. The third fundamental component is the Slow-Start mechanism for the initial probing for available bandwidth. The fourth TCP congestion control mechanism is ACK-clocking, where the arrival of acknowledgements at the sender is used to clock out the transmission of new data.

There are two windows in TCP: receiver advertised window (rwnd) and congestion window (cwnd). The TCP receiver advertised window (rwnd) is added in each sent ACK packet, which sets the size for the sender's sliding window. The sender's transmission rate is then adjusted by the rwnd value, so that the maximum number of allowed outstanding packets is equal to the size of the receiver advertised window at any given time instant, i.e .(rate = rwnd/RTT). Congestion window, which is used by the sender to estimate the network capability, is the key window in the congestion control mechanism. At any given time, instant, the maximum amount of outstanding bytes is equal to MIN (cwnd, rwnd) [5].

Lost packets in the Internet are generally due to data collision and network congestion. Data collision occurs, when the shared media used such as switch and Hub, and this happen in LAN. Congestion refers to a state of the network, where one or more routers receive packets faster than they can forward them. After the queues of one of those routers fill up, it starts to drop packets.

TCP retransmits lost packets, which introduces an overhead in bandwidth utilization. The purpose of congestion control is to try to minimize congestion and, consequently, the need for retransmission by adjusting the transmission rate of TCP. The main concept of congestion control is the congestion window (cwnd), which controls, with the receiver advertised window

(rwnd), the size of sender's sliding window and, thus, the transmission rate. At any given time, instant, the maximum amount of outstanding bytes is equal to min (cwnd, rwnd). Different flavors of TCP have different strategies to react to a loss event, i.e. they resize the cwnd differently.

2.1 Slow Start

Slow start and congestion avoidance are essentially different strategies to grow the cwnd. In the beginning of a connection, TCP sender is in slow start mode. The size of the cwnd is initialized to one Maximum segment size (MSS) and is increased by one each time a new ACK arrives.

The slow start mechanism was a direct attempt to avoid congestion collapse by increasing the packet rate of a connection in a controlled fashion – slowly at first, faster later on - until congestion is detected (i.e. packets are dropped), and ultimately arrive at a steady packet rate, or equilibrium. To achieve this goal, the designers of the slow start mechanism chose an exponential ramp-up function to successively increase the window size. Slow start introduces a new parameter called the congestion window or cwnd, which specifies the number of packets that can be sent without needing a response from the server. TCP starts off slowly (hence the name “slow start”) by sending just one packet, then waits for a response (an ACK) from the receiver. These ACKs confirm that it is safe to send more packets into the network (i.e. double the window size), rather than wait for a response packet by packet. The window size grows exponentially until a router in between the sender and the receiver discards (drops) some packets, effectively telling TCP through a time-out event that its window size has grown too large [10].

2.2 Congestion Avoidance

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion [12].

When a network is congested, queue lengths start to increase exponentially. Congestion avoidance was devised as a technique to signal packet loss via time-outs and make TCP throttle back quickly (more quickly than queue lengths are growing), with the objective of stabilizing the whole system. Near the point of congestion, overly aggressive increases in connection bandwidth can drive the network into saturation, causing the “rush-hour effect”. To avoid the rush-hour phenomenon, the congestion avoidance mechanism increases bandwidth additively rather than exponentially. When congestion is detected via a timeout, bandwidth is scaled back aggressively by setting cwnd to half the current window size. Congestion avoidance is sometimes characterized as being an additive-increase, multiplicative-decrease (AIMD) technique [8].

While slow start and congestion avoidance were devised separately and for different purposes, they are almost always implemented together. The combined algorithm introduces a new variable called ssthresh (slow-start threshold), that effectively determines which mechanism is used.

2.3 Fast Retransmit

The TCP receiver can only acknowledge, the last packet received in sequence. Thus, if packets arrive out of sequence (e.g. one packet was lost but packets sent later arrive correctly), the receiver sends the same ACK more than once [11].

Fast retransmit algorithm makes usage of identical acknowledgement to discover packet loss. In Fast retransmit, during an acknowledgement packet is received a congestion window is fixed to three, TCP sender is adequately assured that the TCP packet is lost and will retransmit the packet beyond waiting for retransmission clock. Fast recovery is approximately connected to retransmit the packet [9].

In Fast recovery algorithm, TCP sender will not arrive in the slow start phase rather it will exactly decrease the congestion window by halve and boost the congestion window by estimating the convenient congestion window. When an acknowledgement of current data is received, it restore to congestion avoidance phase. This appropriate case may cause fast buffer uniform with low use determinant. Suddenly the queues on the maximum loaded lines will build endlessly and, in the end, exceed the width of the buffers at the equivalent nodes. This leads to the known fact that the packets retransmit to the nodes with complete buffers will be reboot and therefore are to be re-entering and that in change effect in wasting of network resources.

2.4 Fast Recovery

TCP Reno introduced a new mechanism called fast Recovery that, changes the congestion control behaviour, after retransmit: When three duplicate ACKs received, TCP sets the slow start threshold (ssthresh) to half of cwnd and cwnd to ssthresh plus three packets [15].

Fast recovery works hand in hand with fast retransmit to improve the responsiveness of TCP once congestion has occurred. Fast recovery introduces the concept of partial acknowledgements, which are used to notify that the next in-sequence packet has been lost so it can be re-transmitted immediately. Furthermore, instead of going back into slow-start mode, fast recovery jumps TCP into congestion avoidance mode. The congestion window is also reduced to the slow start threshold (ssthresh) instead of dropping all the way back to a window size of one packet [11].

3. TCP IMPLEMENTATIONS AND EXTENSIONS

In today's Internet, many different versions of TCP implementation coexist and communicate with each other, most common types of these versions are Reno, Vegas, Tahoe and New Reno, In this section, we review two different TCP implementations Reno and Vegas [17].

3.1 TCP Reno

TCP Reno introduced major improvements by changing the way in which, it reacts to detecting a loss through duplicate acknowledgements. The Reno TCP implementation retained the enhancements incorporated into Tahoe TCP but modified the Fast-Retransmit operation to include Fast Recovery [7]. The new algorithm prevents the communication channel from going empty after Fast Retransmit, thereby avoiding the need to Slow-Start to re-fill it after a single packet loss.

The idea is that the only way for a loss to be detected via a timeout and not via the receipt of a duplicate acknowledgements (dup ACK), is when the flow of packets and ACKs has completely stopped this would be an indication of heavy congestion [21].

The response to packet loss events has been modified in order to maintain a high sending rate, in a mildly congested network. The so, called coarse-grained implementation of the TCP, timeout leads to long idle periods, while waiting for the timeout timer to expire. During this waiting period, packet sending is discontinued, which results in low throughput. In TCP-Reno, the lengthy loss recovery phase has been improved upon, via the introduction of the fast-retransmit loss recovery algorithm [13].

Fast retransmit is a mechanism that, sometimes results in a much faster retransmission, of a lost packet than, what would have been possible if only the expire of timeout timers, was used to detect packet loss. The idea behind the fast-retransmit algorithm, is intuitive and easy to grasp. Every time a packet arrives to the receiver, the receiver responds by returning an acknowledgment packet.

Fast recovery replaces slow-start after a packet loss event is discovered by triple duplicates. The effect of fast retransmit / fast recovery is in principle that, if a packet loss is discovered via triple duplicates, the first lost packet will be quickly, resent and the congestion window size halved. If

the resulting congestion window size allows it, linear increase during congestion avoidance follows directly. This results in a more aggressive, and more effective utilization of the available Network capacity, resulting in high throughput for TCP-Reno sender, when only a few packets are lost at each congestion event where the fast retransmit / fast recovery instance, will re-send, the first lost packet and quickly resume with congestion avoidance. If multiple packets are lost from a single window, the TCP-Reno fast retransmit / fast recovery, algorithm might, however, lead to multiple consecutive invocations, each invocation halving the congestion window size. in case of multiple packet losses, from a single window, the first re-sent packet will lead to, the receiver acknowledging, that it expects the second lost packet [14]. This ACK for a previously, sent and lost packet could be called a partial ACK. In the TCP-Reno implementation of fast retransmit / fast recovery, the arrival of partial ACKs will initiate a new fast retransmit / fast recovery followed by window halving.

These consecutive window halving, will decrease the congestion window so much that TCP-Reno, will ultimately not be able to send, any new packets, due to the congestion window size restriction, on the number of packets, it is allowed to have, un-acknowledged on the link. Hence, multiple packet losses, might finally lead to the sender having, to wait for a coarse timeout timer to expire, even if the re-sent packets, are being correctly received and acknowledged.

3.2 TCP Vegas

New TCP implementation, called Vegas is presented in 1994 by Brakmo. O'Malley and Petersen. TCP Vegas adopts a more sophisticated, bandwidth estimation scheme. Vegas algorithm estimates the buffering that does arise in reach the system and controls the rate affiliate with appropriate flow. This algorithm is absolutely capable to regulate and decrease the flow rate since the packet loss arise.

It uses the difference between expected and actual flows rates, to estimate the available bandwidth in the network. The idea is that when the network, is not congested, the actual flow rate, will be close to the expected flow rate [6]. Otherwise, the actual flow rate, will be smaller than the expected flow rate. TCP Vegas, using this difference in flow rates, estimates the congestion level, in the network and updates the window size accordingly. This difference in the flow rates can be easily translated into the difference between the window size and the number of acknowledged packets during the roundtrip time, using the equation:

$$\text{Diff} = (\text{Expected} - \text{Actual})\ \text{BaseRTT}$$

Where Expected is the expected rate, Actual is the actual rate, and BaseRTT is the minimum round trip time. The details of the algorithm are as follow:

1. First, the sender computes the expected flow rate:

$$Expected = \frac{CWND}{BaseRTT},$$

   where CWND is the current window size and BaseRTT is the minimum round trip time.

2. Second, the sender estimates the current flow rate by using the actual round trip time.

$$Actual = \frac{CWND}{RTT}$$

   where RTT is the actual round trip RTT time of a packet.

3. The sender, using the expected and actual flow rates, computes the estimated backlog in the queue from diff= (Expected – Actual) BaseRTT.

4. Based on diff, the sender updates its window size as follows:

$$CWND = \begin{cases} CWND + 1 & \text{if } diff < \alpha \\ CWND - 1 & \text{if } diff > \beta \\ CWND & otherwise \end{cases}$$

TCP Vegas tries to keep at least α packets but no more than β packets in the queues. The reason behind this is that TCP Vegas attempts to detect and utilize the extra bandwidth whenever it becomes available without congesting the network. This mechanism is fundamentally different from that used by TCP Reno. TCP Reno always updates its window size to guarantee full utilization of available bandwidth, leading to constant packet losses, whereas TCP Vegas does not cause any oscillation in window size once it converges to an equilibrium point

4. CONGESTION MANAGEMENT

Congestion can occur at any point in the network where there are points of speed mismatches, aggregation, or confluence Queuing manages congestion to provide bandwidth and delay guarantees.

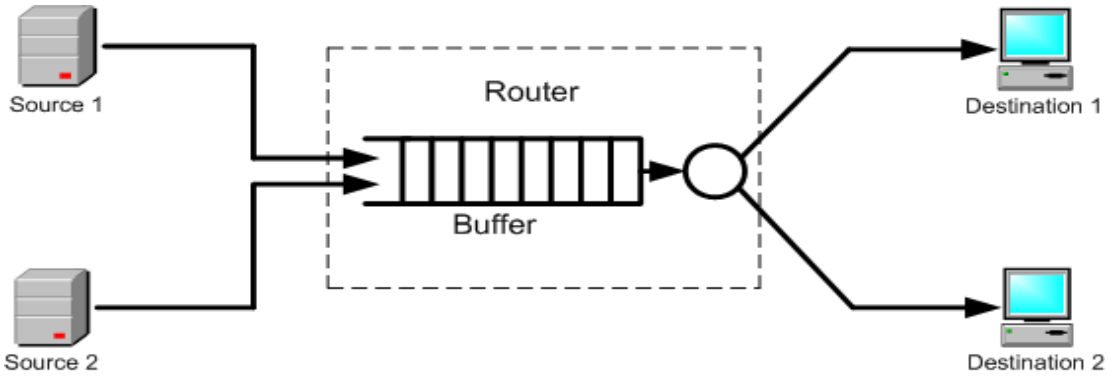


FIGURE 1: Queuing Theory **.**

The queuing theory work showing in Figure 1, when source 1 and source 2, send sum of packets at the same time the router starts to decide, where to direct first packet, and at the same time begin saving, the another packets into, the Buffer as seen in Figure 1. The way which the router (gateway), mange that packets in the buffer is called queuing theory [13].

During congestion, gateway influences, the fairness problem, because its queuing discipline, determines which packet to drop. This has effect on retransmissions used by TCP. Normal or default configuration, of gateways used the FIFO discipline, commonly referred to as Drop Tail (DT) gateway. Other possible gateway's, configurations of discipline include, the Random Drop (RD) Gateway and the Random Early Detection (RED) Gateway [16].

The TCP algorithm, at the sender, and the receiver is only part of the congestion control effort. An equally important part, is the congestion feedback from the network, which is the basis of any congestion control actions [20].

Congestion feedback, can be delivered in different ways. The most primitive feedback, is packet drops. In case of buffer overflow, the network simply drops incoming packets that cannot be accommodated in the router buffer. The source can detect the packet drop, either from a timeout, or from duplicate acknowledgments [22].

5. QUEUING MANAGEMENT ALGORITHMS

In this section we outline the most common queue management algorithms. Queue management is obviously about managing queues in forwarding devices such as routers and switches.

5.1 Drop Tail

The simplest queue management algorithm, where, when a queue becomes full, packets are simply dropped. Since packets can be dropped on all TCP connections simultaneously, many TCP connections on the link can be forced to go into slow-start mode. Figure 2 showing the Droptail queueing [20].

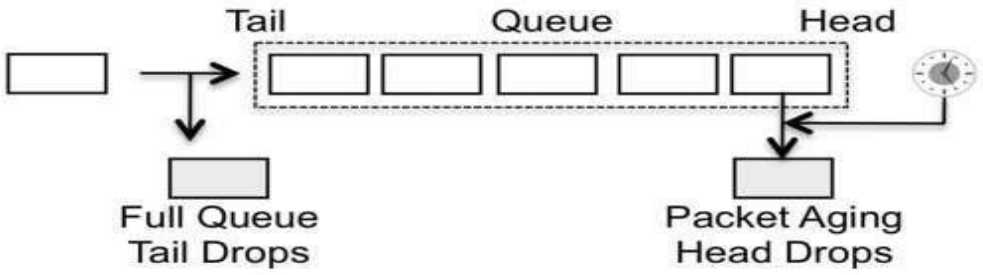


FIGURE 2: Drop Tail Queueing.

5.2 Fair Queuing

It is a queuing mechanism that is used to allow multiple packets flow to comparatively share the link capacity. "Fair Queuing" is an attempt to give the flows equal shares, at least within the limits of actual demand.

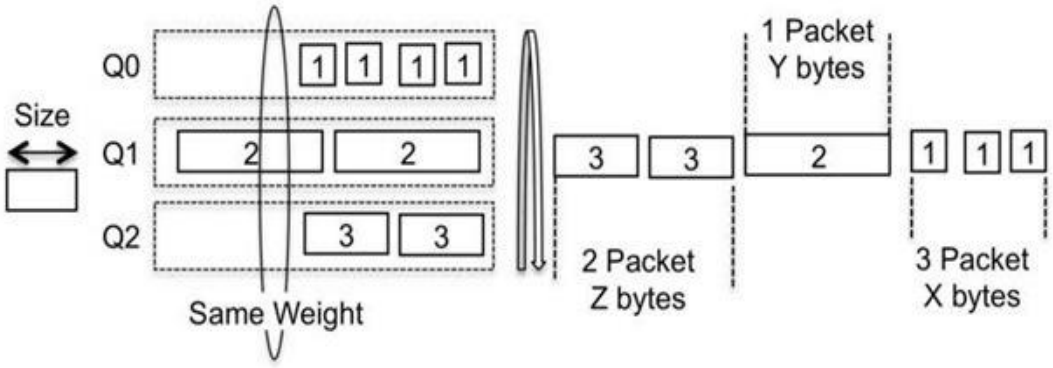


FIGURE 3: Round Robin with Different Size Packets.

The simplest algorithm for fair queuing is round-robin queue service, with all packets of equal size; this is sometimes called Nagle Fair Queuing, each nonempty queue gets to send an equal share of packets, Nagle fair queuing allows other flows to use more than their equal share, if some flows are underutilizing. Shares are divided equally among the active flows. As soon as a flow becomes active (that is, its queue becomes nonempty) it gets to start sharing in the bandwidth allocation; it does not have to wait for other flows to work through their backlogs, Round-robin works as fair queuing as long as all packets have the same size [18]. If packets have different sizes, then flows all get their fair share of packets per second, but this may not relate to bytes per second. FQ also ensure about the maximum throughput of the network. Figure 3 showing the Round Robin [19].

5.3 Deficit Round Robin

It is a modified weighted round robin scheduling mechanism. It can handle packets of different size without having knowledge of their mean size. Deficit Round Robin keeps track of credits for each flow. It derives ideas from Fair Queuing. It uses hashing to determine the queue to which a

flow has to be assigned and collisions automatically reduce the bandwidth guaranteed to the flow. Each queue is assigned a quantum and can send a packet of size that can fit in the available quantum. If not, the idle quantum gets added to this meticulous queue's deficit and the packet can be sent in the next round. The quantum size is a very vital parameter in the DRR scheme, determining the upper bound on the latency as well as the throughput as shown in Figure 4. [20]

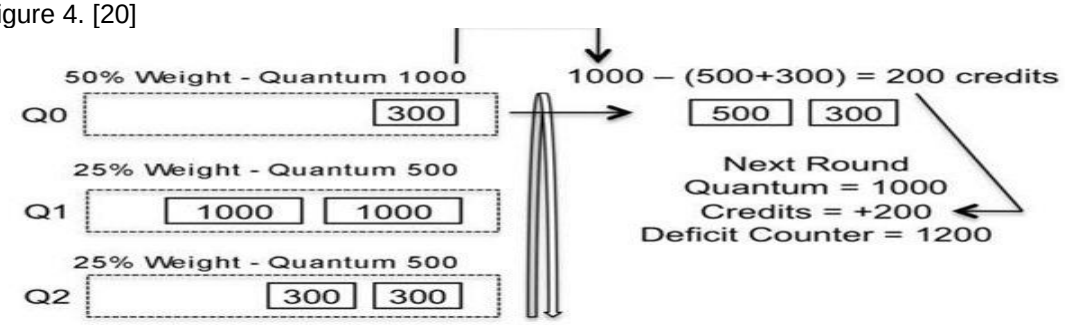


FIGURE 4: Deficit Round Robin.

5.4 RED

Random Early Detection (RED) is a congestion avoidance queuing mechanism. RED is a type of congestion control algorithm/mechanism that takes advantage of TCP's congestion control mechanisms and takes proactive approach to congestion.

It operates on the average queue size and drop packets on the basis of statistics information. If the buffer is empty all incoming packets are acknowledged. As the queue size increase the probability for randomly discarding a packet also increase. When buffer is full probability becomes equal to 1 and all incoming packets are dropped [21].

RED has three modes:

- No drop: When the average queue size is between 0 and the minimum threshold.
- Random drop: When the average queue size is between the minimum and the maximum Threshold.
- Full drop (tail drop): When the average queue size is at maximum threshold or above.

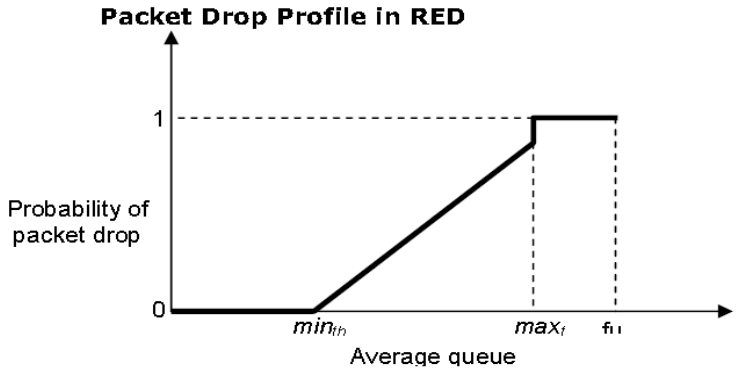


FIGURE 5: Packet Drop Profile In RED.

6. SIMULATION SCENARIO and CONFIGURATION

In this section, we evaluate the performance proposed for the different Queueing Algorithms over TCP Protocols. This network was examined using NS2 Simulator software.

Figure 6 show the proposed scheme of the network, covering the needs of an average networks with a data server that allows to send files and make voice / video calls and FTP Application. Although minor changes could be made to allow for the needs of different Networks. This network was designed to connect two branches; each branch has 4 nodes. In addition, 2 routers were used to connect network.

Here we will be using five different mechanisms which have different behavior for different network configuration and traffic pattern. Most importantly, the task in designing the simulation is to select parameters (bandwidth, queue limit, packet size, etc.) and a typical set of network topology. A simple topology is used in our simulation where different flows share a bottleneck between the two routers. The packets sent from sources queue to the queue of router and wait for transmitting. If the sender keeps sending and the queue overloaded, then congestion occurs.

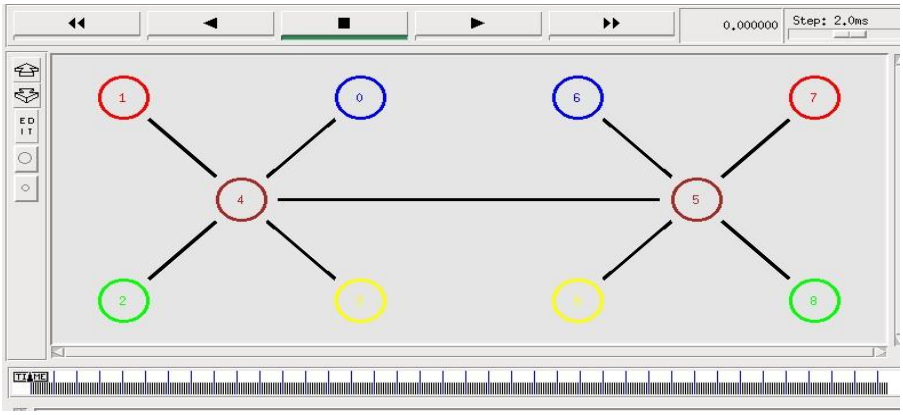


FIGURE 6: Network Topology of The Case Sudy.

There are four nodes at each side of the bottleneck link. Here four nodes are acting as a TCP source (Vegas, Reno) and four nodes are acting as a TCP sink so that both routers are applying the congestion control algorithm. We simulate this network on NS2 for different Queueing algorithms such as Drop tail, Fair Queueing, Random Early Detection and Deficit Round Robin over TCP protocols Vegas and Reno and compare the results for all proposal. This simulation has been observed over the period of 200 seconds.

TABLE 1: Simulation Parameters.

Simulation Parameter	V
Simulator	Ns-allinone-
Type of link	Duplex Link
Routers	2
Nodes	8
Network Applications	FTP, VOIP, Video conferencing
Queue	Drop Tail, FQ, RED, RDD
Speed	2MB
Delay	20ms
Transmission Protocol	TCP
Simulation start time	0s
Simulation finish time	200s

7. SIMULATION RESULTS

The performance of the congestion management system that consists of the queueing algorithm and the link congestion signal, algorithm has many facets, and these variables can impact the QoS, that applications experience from the network. Varieties of metrics that address different aspects of performance used to evaluate a congestion management system. At first, the results will be displayed algorithms, followed by an explanation of how the work of the network and right after, the results will be discussed. A final step and it will explain the main differences between the four algorithms.

7.1 Congestion Window in TCP Reno

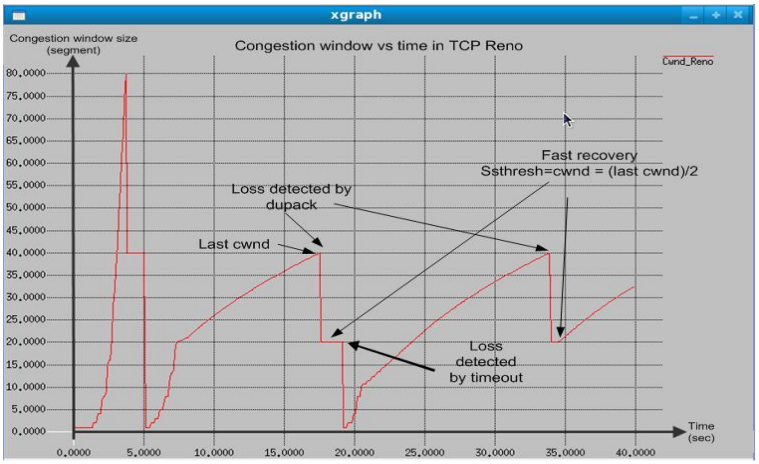


FIGURE 7: Congestion Window In TCP Reno.

7.2 Send, Dropped and ACK Packet In 40 Sec using TCP Reno

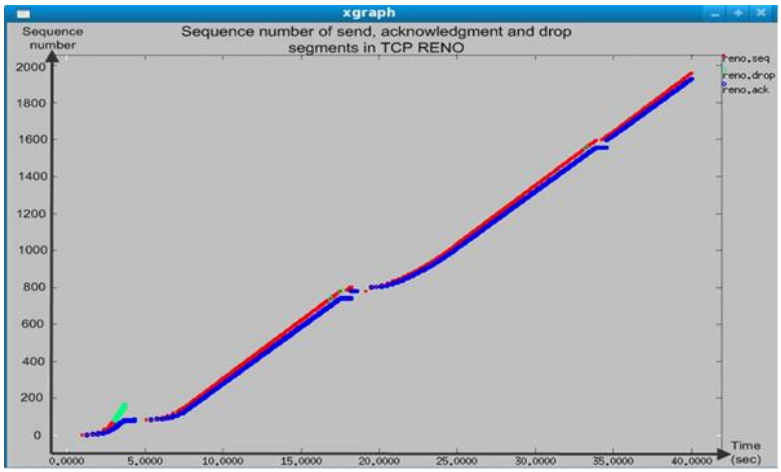


FIGURE 8: Send, Dropped and ACK Packet In 40 Sec using TCP Reno.

7.3 Queueing Algorithms (Droptail, FQ, RDD and RED) over TCP Reno

If all TCP Sources work as TCP Reno and the used queuing type is Drop Tail and Fair Queueing in a bottleneck link, G1-G2 and Queuing limit (Buffer size) is equal to 20. The results shown in Figure 9 Drop Tail and in Figure 10 FQ and tables are below.

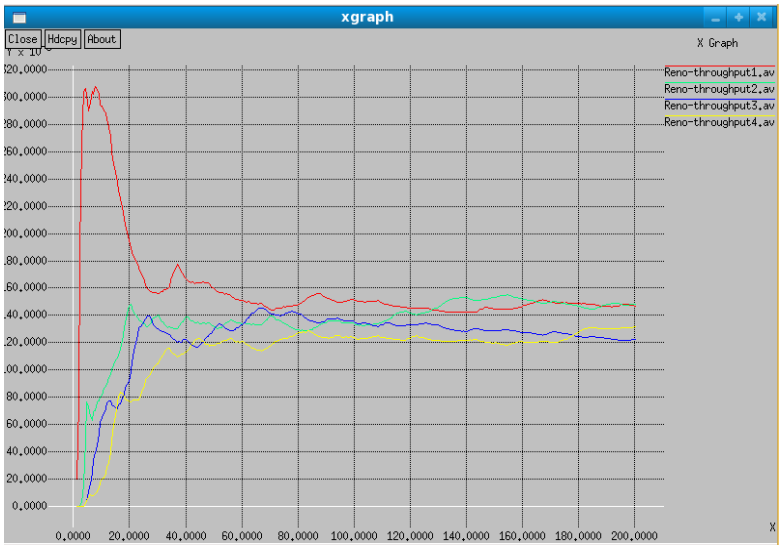


FIGURE 9: Throughput kbps to four sources work as TCP Reno when Queuing type is DropTail.

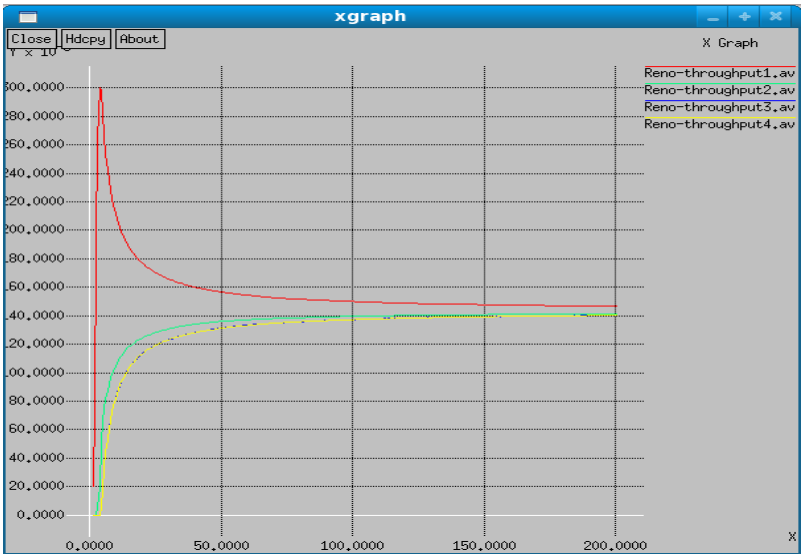


FIGURE 10: Throughput kbps to four sources work as TCP Reno when Queuing type is FQ.

If all TCP Sources work as TCP Reno and the used queuing type is DRR and RED in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 11 Deficit Round Robin and in Figure 12 RED and tables are below.

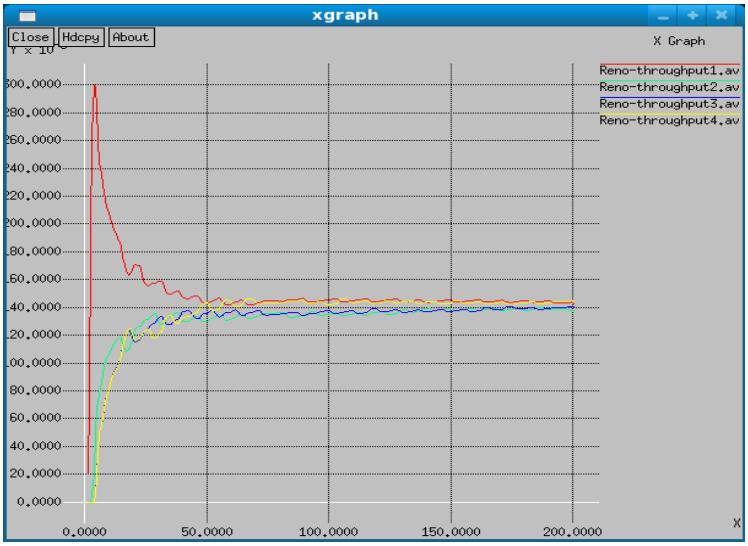


FIGURE 11: Throughput kbps to four sources work as TCP Reno and Queuing type is Deficit Round Robin.

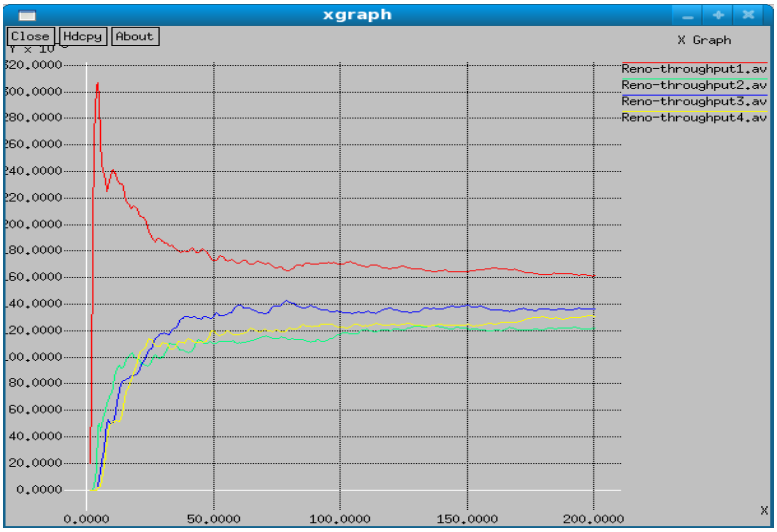


FIGURE 12: Throughput kbps to four sources work as TCP Reno and Queuing type is RED.

TABLE (7-1): Compare between Sent, Received and Dropped packets with different Queuing algorithms in TCP Reno**.**

	Sources	S1	S2	S3	S4
Droptail	**Sent Packet**	3775	3816	3155	3385
	Received Packet	3706	3750	3085	3320
	Dropped Packet	68	66	70	65
	Sent Packet	3719	3588	3559	3559
	Received Packet	3706	3576	3547	3547

Fair Queueing	**Dropped Packet**	0	0	0	0
DRR	**Sent Packet**	3675	3550	3627	3724
	Received Packet	3612	3480	3558	3660
	Dropped Packet	63	70	69	64
RED	**Sent Packet**	4182	3190	3551	3406
	Received Packet	4085	3077	3449	3306
	Dropped Packet	97	113	102	100

7.4 Congestion Window In TCP Vegas

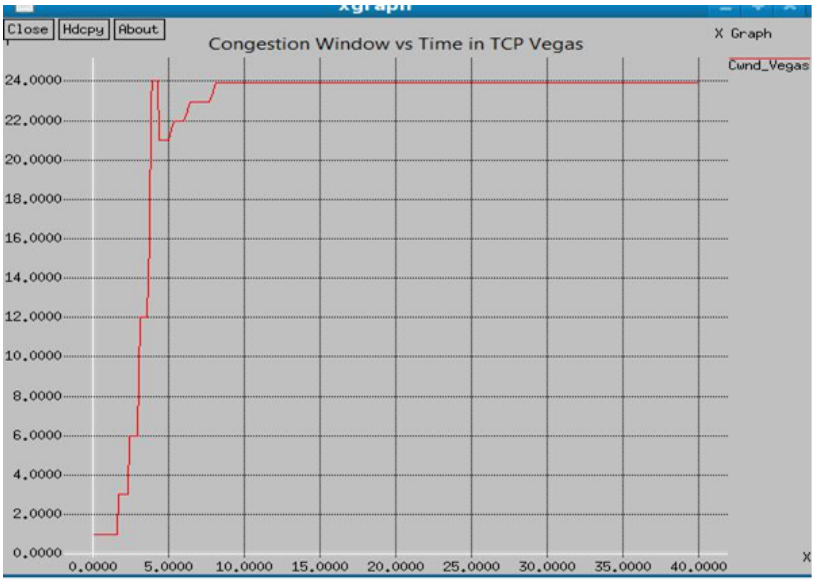


FIGURE 13: Congestion Window in TCP Vegas.

7.5 Send, dropped and ACK packet in 40 sec using TCP Vegas

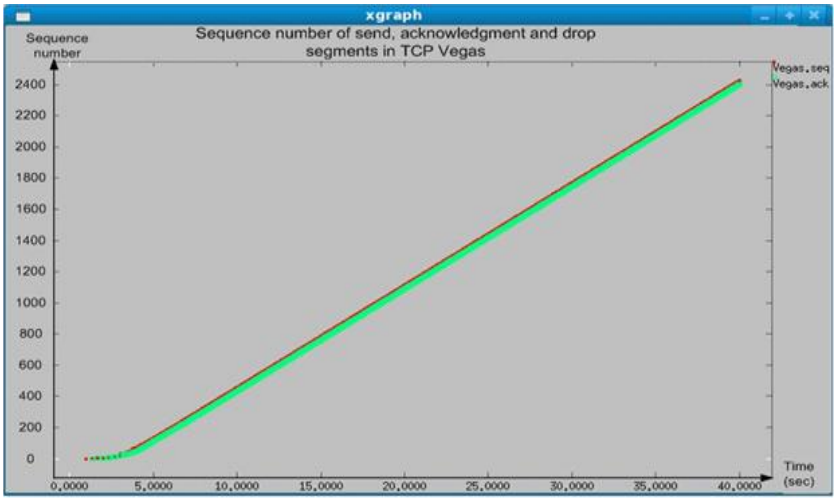


FIGURE 14: Send, Dropped and ACK Packet In 40 Sec using TCP Vegas.

7.6 Queueing Algorithms (Droptail, FQ, RDD and RED) over TCP Vegas

If all TCP Sources work as TCP Vegas and the used queuing type is Drop Tail and Fair Queueing in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 15 Drop Tail and in Figure 16 FQ and tables are below.

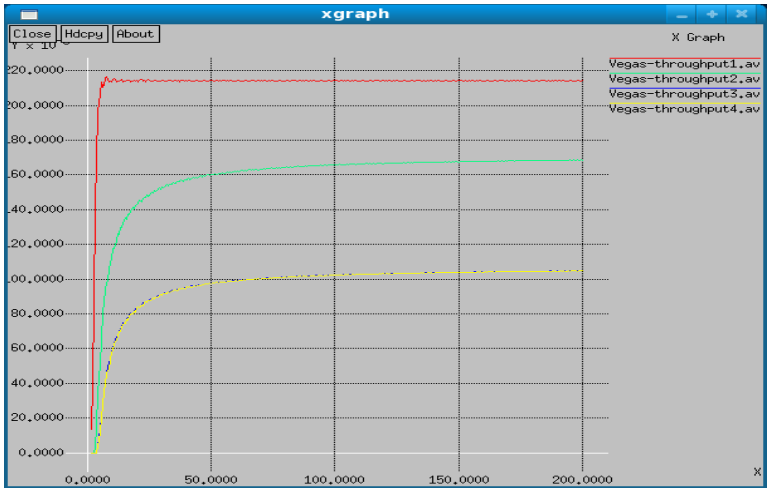


FIGURE 15: Throughput kbps to four-sources work as TCP Vegas when Queuing type is DropTail.

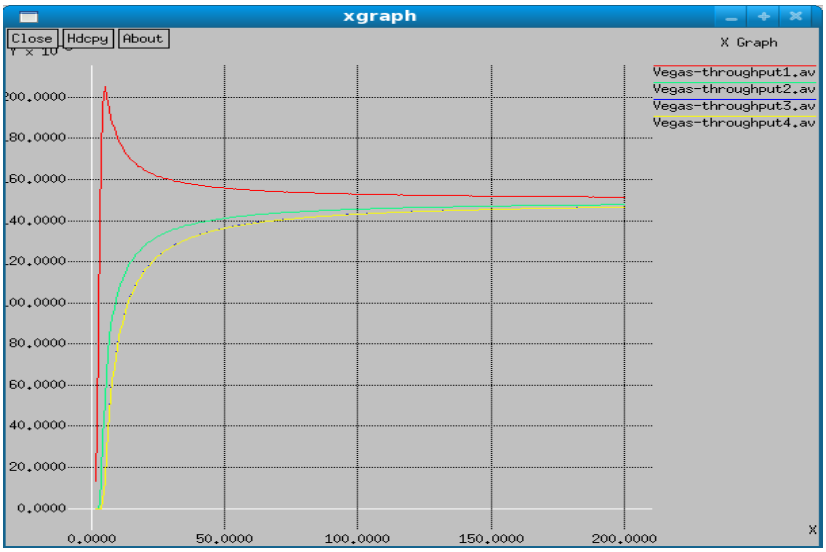


FIGURE 16: Throughput kbps to four-sources work as TCP Vegas when Queuing type is FairQueueing.

If all TCP Sources work as TCP Vegas and the used queuing type is DRR and RED in a bottleneck link, G1-G2 and Queuing limit (Buffer size) are equal to 20. The results shown in Figure 17 Drop Tail and in Figure 18 FQ and tables are below.

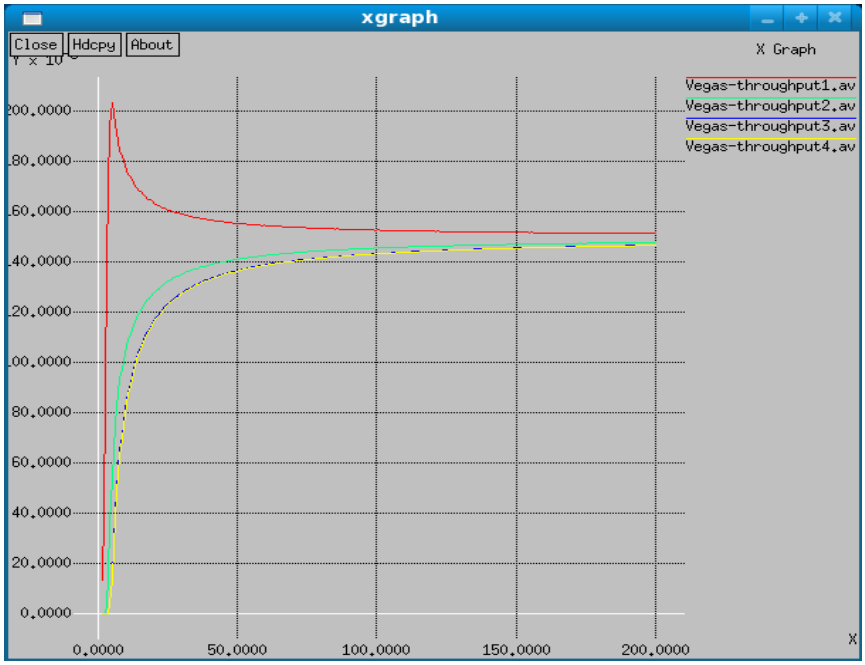


FIGURE 17: Throughput kbps to four sources work as TCP Vegas when Queuing type is DRR.

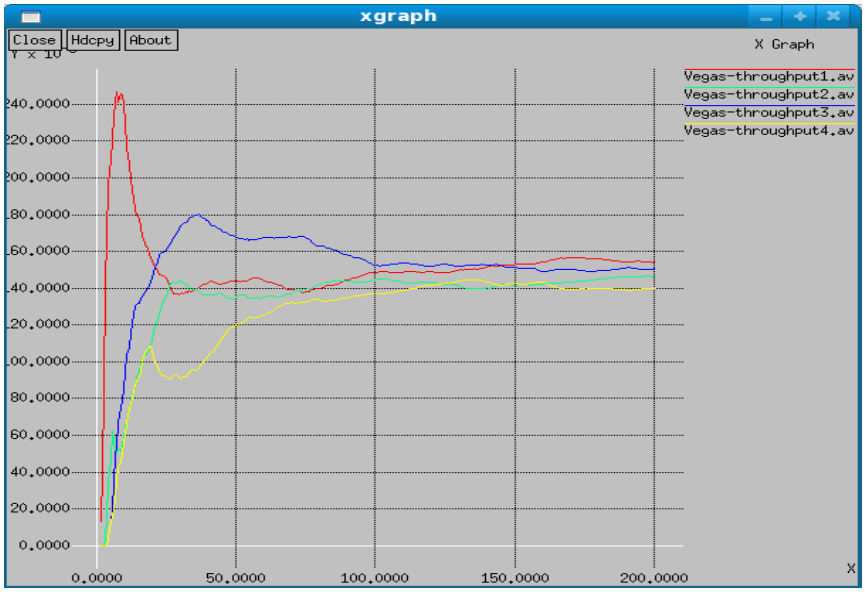


FIGURE 18: Throughput to four sources work as TCP Vegas when Queuing type is RED.

	Sources	S1	S2	S3	S4
Droptail	**Sent Packet**	5382	4234	2630	2630
	Received Packet	5382	4234	2630	2630
	Dropped Packet	0	0	0	0
Fair Queueing	**Sent Packet**	3803	3713	3681	3680
	Received Packet	3803	3713	3681	3680
	Dropped Packet	0	0	0	0
DRR	**Sent Packet**	3801	3712	3682	3681
	Received Packet	3801	3712	3682	3681
	Dropped Packet	0	0	0	0
RED	**Sent Packet**	3960	3761	3865	3625
	Received Packet	3869	3673	3779	3520
	Dropped Packet	91	88	86	105

TABLE (7-2): Compare between Sent, Received and Dropped packets with different Queuing algorithms in TCP Vegas**.**

7.8 Compare between sent and received packets with different Queuing Algorithms in (TCP Reno and Vegas):

Type	TCP Vegas	TCP Reno
Drop Tail	14876	13861
Fair Queueing	14877	14376
DRR	14876	14310
RED	14841	13917

TABLE (7-3): Compare Between Total of Sent Packets.

Type	TCP Vegas	TCP Reno
Drop Tail	1162.1785	276.0121
Fair Queueing	50.1416	65.7381
DRR	48.9540	66.8636
RED	1129.9141	374.0550

TABLE (7-4): Compare Standard Deviation.

8. CONCLUSION

In this paper, we analysed the proposed scheme of 'A Comparison of Queueing Algorithms over TCP Protocol'.

While looking at the performance metrics (fairness and throughput) for all four queueing algorithms, we find that The DRR algorithm in TCP Vegas is the best in fairness. Then we find FQ algorithm in both TCP (Vegas and Reno) is better than DRR in Reno and the other two algorithms RED and Droptail, so we can conclude that the overall performance of FQ algorithm in both TCP protocols Reno and Vegas is better than DRR. DRR in TCP Vegas is better than FQ only in fairness but in throughput it is the same with FQ and Droptail and better than RED.

Moreover, we find Droptail in Reno is better than RED in both TCP protocols (Reno and Vegas), but when it is in Vegas, it is less fairness than RED and better throughput, so that the overall performance of Droptail in both TCP protocols Reno and Vegas is better than RED except when it is in Vegas, then it is worse in fairness.

From this study we can classify these four Queueing algorithms in accordance to performance metrics as this order:

1-FQ
2-DRR
3-Droptail
4-RED

9. REFERENCES

[1] Lawrence G. Roberts, “Beyond Moore’s Law: Internet Growth Trends,” Computer, vol. 33, no. 1, pp. 117-119, January 2000.

[2] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, Analysis and Comparison of TCP Reno and Vegas.

[3] V. Jacobson. “Modified TCP Congestion Avoidance Algorithm”, Technical report, 30 Apr.1990

[4] M. Fomenkov, K. Keys, D. Moore, and K. Claffy, Longitudinal studyofInte rnettrafficin 1998-2003,WISICT’04:Proc.Winter Int. Symp. Info.Commun.Technol, 2004.

[5] K. Fall, and S. Floyd, “Simulation–Based Comparison of Tahoe, Reno and SACK

[6] TCP”, Computer Communications Review ACMSIGCOMM, Vol. 26, No. 3, July 1996 K. Fall, and S. Floyd, “Simulation–Based Comparison of Tahoe, Reno and

[7] SACK TCP”, Computer Communications Review ACMSIGCOMM, Vol. 26, No. 3, July 1996.

[8] M. Chiang, S. Low, A. Calderbank and J. Doyle, Layering as optimization decomposition: Amathematical theory of network architectures, Proc. of the IEEE,95(1): 255–312, 2007.

[9] UDP Protocol,http://www.erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html. 5 NOV 2015.

[10] UDP and TCP Segment Structure http ://searchnetworking.techtarget .com / definition/duplex http://www.ciscopress.com/store/cisco-ip-telephony-cipt -authorized-self-study-guide-9781587054099. , 22 Dec 2015.

[11] Nagle, J. RFC 896: Congestion control in IP/TCP internetworks (1984).

[12] TCP Protocol http://searchnetworking.techtarget.com/definition/TCP. , 5 DEC 2015.

[13] The Advantages and Disadvantages TCP and UDP http://smblog.iiitd .com /2010 /09 /advantages-and-disadvantages-of-tcp-and.html. , 31 DEC 2015.

[14] V. Jacobson, "Congestion Avoidance and Control", In Proceedings of ACM SIGCOMM' 88, PP. 314-329, Stanford, CA, August 1988.

[15] Chunlei Liu, B.Sc., M.S."Wireless Network Enhancements Using Congestion Coherence, Faster Congestion Feedback , Media Access Control and AAL2 Voice Trunking " 2001, The Ohio State University.

[16] B.B. et al."Recommendations on queue management and congestion avoidance in the internet". RFC 2309, April 1998.

[17] Romanow and S.Floyd. "The dynamics of TCP over ATM networks . In Proceedings,1994 SIGCOMM Conference , , London, 1994.

[18] N.Yin and M. G. Hluchyj."Implication of dropping packets from the front of a queue.proc".7th ITC seminar ,1990.

[19] T.Lakshman, A. Neidhardt, and T. Ott. The drop from front strategy in tcp and in tcp over atm,1996.

[20] S. Floyd and V.Jacobson. "Random early detection gateways for congestion avoidance". IEEE/ACM Transactions on Networking,Aug.1993.

[21] J. Postal."Internet control message protocol icmp",1981,ISI.

[22] Michael Welzl "Network Congestion Control Managing Internet Traffic" John Wiley & Sons Ltd,2005.

INSTRUCTIONS TO CONTRIBUTORS

The *International Journal of Computer Science and Security (IJCSS)* is a refereed online journal which is a forum for publication of current research in computer science and computer security technologies. It considers any material dealing primarily with the technological aspects of computer science and computer security. The journal is targeted to be read by academics, scholars, advanced students, practitioners, and those seeking an update on current experience and future prospects in relation to all aspects computer science in general but specific to computer security themes. Subjects covered include: access control, computer security, cryptography, communications and data security, databases, electronic commerce, multimedia, bioinformatics, signal processing and image processing etc.

To build its International reputation, we are disseminating the publication information through Google Books, Google Scholar, Open J Gate, ScientificCommons, Docstoc and many more. Our International Editors are working on establishing ISI listing and a good impact factor for IJCSS.

The initial efforts helped to shape the editorial policy and to sharpen the focus of the journal. Started with Volume 14, 2020, IJCSS is appearing with more focused issues. Besides normal publications, IJCSS intend to organized special issues on more focused topics. Each special issue will have a designated editor (editors) – either member of the editorial board or another recognized specialist in the respective field.

We are open to contributions, proposals for any topic as well as for editors and reviewers. We understand that it is through the effort of volunteers that CSC Journals continues to grow and flourish.

IJCSS LIST OF TOPICS

The realm of International Journal of Computer Science and Security (IJCSS) extends, but not limited, to the following:

- Authentication and authorization models
- Computer Engineering
- Computer Networks
- Cryptography
- Databases
- Image processing
- Operating systems
- Programming languages
- Signal processing
- Theory
- Communications and data security
- Bioinformatics
- Computer graphics
- Computer security
- Data mining
- Electronic commerce
- Object Orientation
- Parallel and distributed processing
- Robotics
- Software engineering

CALL FOR PAPERS

Volume: 14 - **Issue: 1**

i. Submission Deadline : December 31, 2019 **ii. Author Notification:** January 31, 2020

iii. Issue Publication: February 2020

CONTACT INFORMATION

Computer Science Journals Sdn BhD

B-5-8 Plaza Mont Kiara, Mont Kiara

50480, Kuala Lumpur, MALAYSIA

Phone: 006 03 6204 5627

Fax: 006 03 6204 5628

Email: cscpress@cscjournals.org

COMPUTER SCIENCE JOURNALS SDN BHD
B-5-8 PLAZA MONT KIARA
MONT KIARA
50480, KUALA LUMPUR
MALAYSIA

PHONE: 006 03 6204 5627
FAX: 006 03 6204 5628
EMAIL: cscpress@cscjournals.org