ELSEVIER

INSTRUCTIONS ON THE ANNOTATION OF PDF FILES

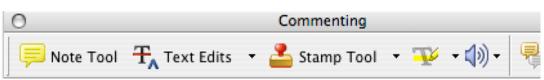
To view, print and annotate your content you will need Adobe Reader version 9 (or higher). This program is freely available for a whole series of platforms that include PC, Mac, and UNIX and can be downloaded from http://get.adobe.com/reader/. The exact system requirements are given at the Adobe site: http://www.adobe.com/products/reader/tech-specs.html.

*Note: **Please do NOT make direct edits to the PDF using the editing tools as doing so could lead us to overlook your desired changes.** Rather, please request corrections by using the tools in the Comment pane to annotate the PDF and call out the changes you are requesting. If you opt to annotate the file with software other than Adobe Reader then please also highlight the appropriate place in the PDF file.*

PDF ANNOTATIONS

Adobe Reader version 9

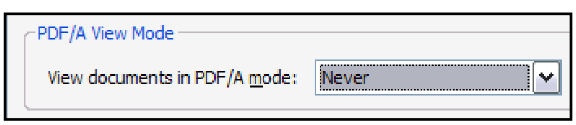
When you open the PDF file using Adobe Reader, the Commenting tool bar should be displayed automatically; if not, click on 'Tools', select 'Comment & Markup', then click on 'Show Comment & Markup tool bar' (or 'Show Commenting bar' on the Mac). If these options are not available in your Adobe Reader menus then it is possible that your Adobe Acrobat version is lower than 9 or the PDF has not been prepared properly.



(Mac)

PDF ANNOTATIONS (Adobe Reader version 9)

The default for the Commenting tool bar is set to 'off' in version 9. To change this setting select 'Edit | Preferences', then 'Documents' (at left under 'Categories'), then select the option 'Never' for 'PDF/A View Mode'.

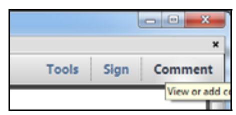


(Changing the default setting, Adobe version 9)

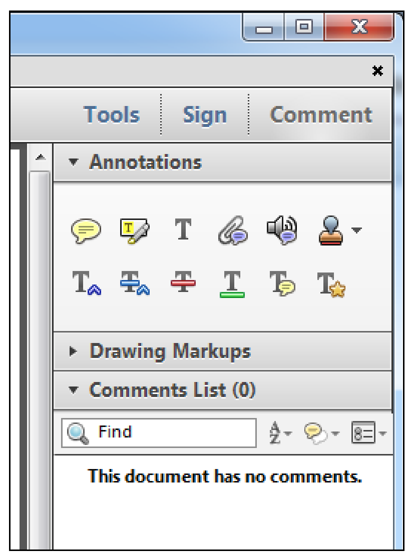
Adobe Reader version X and XI

To make annotations in the PDF file, open the PDF file using Adobe Reader XI, click on 'Comment'.

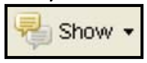
If this option is not available in your Adobe Reader menus then it is possible that your Adobe Acrobat version is lower than XI or the PDF has not been prepared properly.



This opens a task pane and, below that, a list of all Comments in the text. These comments initially show all the changes made by our copyeditor to your file.



HOW TO...		
Action	**Adobe Reader version 9**	**Adobe Reader version X and XI**
Insert text	Click the 'Text Edits' button on the Commenting tool bar. Click to set the cursor location in the text and simply start typing. The text will appear in a commenting box. You may also cut-and-paste text from another file into the commenting box. Close the box by clicking on 'x' in the top right-hand corner.	Click the 'Insert Text' icon on the Comment tool bar. Click to set the cursor location in the text and simply start typing. The text will appear in a commenting box. You may also cut-and-paste text from another file into the commenting box. Close the box by clicking on '_' in the top right-hand corner.
Replace text	Click the 'Text Edits' button on the Commenting tool bar. To highlight the text to be replaced, click and drag the cursor over the text. Then simply type in the replacement text. The replacement text will appear in a commenting box. You may also cut-and-paste text from another file into this box. To replace formatted text (an equation for example) please Attach a file (see below).	Click the 'Replace (Ins)' icon on the Comment tool bar. To highlight the text to be replaced, click and drag the cursor over the text. Then simply type in the replacement text. The replacement text will appear in a commenting box. You may also cut-and-paste text from another file into this box. To replace formatted text (an equation for example) please Attach a file (see below).
Remove text	Click the 'Text Edits' button on the Commenting tool bar. Click and drag over the text to be deleted. Then press the delete button on your keyboard. The text to be deleted will then be struck through.	Click the 'Strikethrough (Del)' icon on the Comment tool bar. Click and drag over the text to be deleted. Then press the delete button on your keyboard. The text to be deleted will then be struck through.
Highlight text/ make a comment	Click on the 'Highlight' button on the Commenting tool bar. Click and drag over the text. To make a comment, double click on the highlighted text and simply start typing.	Click on the 'Highlight Text' icon on the Comment tool bar. Click and drag over the text. To make a comment, double click on the highlighted text and simply start typing.
Attach a file	Click on the 'Attach a File' button on the Commenting tool bar. Click on the figure, table or formatted text to be replaced. A window will automatically open allowing you to attach the file. To make a comment, go to 'General' in the 'Properties' window, and then 'Description'. A graphic will appear in the PDF file indicating the insertion of a file.	Click on the 'Attach File' icon on the Comment tool bar. Click on the figure, table or formatted text to be replaced. A window will automatically open allowing you to attach the file. A graphic will appear indicating the insertion of a file.
Leave a note/ comment	Click on the 'Note Tool' button on the Commenting tool bar. Click to set the location of the note on the document and simply start typing. Do not use this feature to make text edits.	Click on the 'Add Sticky Note' icon on the Comment tool bar. Click to set the location of the note on the document and simply start typing. Do not use this feature to make text edits.

HOW TO...		
Action	**Adobe Reader version 9**	**Adobe Reader version X and XI**
Review	To review your changes, click on the 'Show' button  on the Commenting tool bar. Choose 'Show Comments List'. Navigate by clicking on a correction in the list. Alternatively, double click on any mark-up to open the commenting box.	Your changes will appear automatically in a list below the Comment tool bar. Navigate by clicking on a correction in the list. Alternatively, double click on any mark-up to open the commenting box.
Undo/delete change	To undo any changes made, use the right click button on your mouse (for PCs, Ctrl-Click for the Mac). Alternatively click on 'Edit' in the main Adobe menu and then 'Undo'. You can also delete edits using the right click (Ctrl-click on the Mac) and selecting 'Delete'.	To undo any changes made, use the right click button on your mouse (for PCs, Ctrl-Click for the Mac). Alternatively click on 'Edit' in the main Adobe menu and then 'Undo'. You can also delete edits using the right click (Ctrl-click on the Mac) and selecting 'Delete'.

SEND YOUR ANNOTATED PDF FILE BACK TO ELSEVIER

Save the annotations to your file and return as instructed by Elsevier. Before returning, please ensure you have answered any questions raised on the Query Form and that you have inserted all corrections: later inclusion of any subsequent corrections cannot be guaranteed.

FURTHER POINTS

- Any (grey) halftones (photographs, micrographs, etc.) are best viewed on screen, for which they are optimized, and your local printer may not be able to output the greys correctly.
- If the PDF files contain colour images, and if you do have a local colour printer available, then it will be likely that you will not be able to correctly reproduce the colours on it, as local variations can occur.
- If you print the PDF file attached, and notice some 'non-standard' output, please check if the problem is also present on screen. If the correct printer driver for your printer is not installed on your PC, the printed output will be distorted.

AUTHOR QUERY FORM

ELSEVIER	Journal: COSE Article Number: 1251	Please e-mail your responses and any corrections to: E-mail: corrections.esch@elsevier.toppanbestset.com

Dear Author,

Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list. To ensure fast publication of your paper please return your corrections within 48 hours.

For correction or revision of any artwork, please consult http://www.elsevier.com/artworkinstructions.

We were unable to process your file(s) fully electronically and have proceeded by

☐ Scanning (parts of) your article ☐ Rekeying (parts of) your article ☐ Scanning the artwork

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof. Click on the 'Q' link to go to the location in the proof.

Location in article	Query / Remark: click on the Q link to go Please insert your reply or correction at the corresponding line in the proof
Q1	Please confirm that given names and surnames have been identified correctly and are presented in the desired order and please carefully verify the spelling of all authors' names.
Q2	Your article is registered as belonging to the Special Issue/Collection entitled ″SP-PRO-SUS-DEV″. If this is NOT correct and your article is a regular item or belongs to a different Special Issue please contact s.sandacoumar@elsevier.com immediately prior to returning your corrections.
Q3	The number of keywords provided doesn't match the minimum (i.e., 5–10) allowed by this journal. Please provide more keywords.
Q4	"can better benefit of power scaling" changed to "better benefit from power scaling" correct? Please check or amend as necessary.
Q5	"w.r.t." has been changed to "with regard to" correct? Please check or amend as necessary.
Q6	"routers nominal capacity" changed to "router's nominal capacity" correct? Please check and confirm or amend as necessary.
Q7	Please note that the reference style has been changed from a Numbered style to a Name–Date style as per the journal specifications.
Q8	Please check all website addresses and confirm that they are correct. (Please note that it is the responsibility of the author(s) to ensure that all URLs given in this article are correct and useable.)
Q9	Please provide accessed date for Reference G. Group, 2017.

Thank you for your assistance.

Q10	Please provide year of publication for Reference Paganini.
Q11	Figure 1: The quality of Figure is too poor to be used. Please provide better quality figure.
Q12	Figure 2: The resolution of Figure is too low to be used. Please provide better quality figure[s] of 300 dpi.
	Please check this box or indicate your approval if you have no corrections to make to the PDF file

Thank you for your assistance.

Highlights

• Selective distributed routing and intrusion detection based on dynamic statistical analysis. • Adaptively organizes the intrusion detection activities. • Suppresses at the network ingress the undesired components of latency-insensitive traffic. • Distributes over multiple nodes the security check for latency sensitive flows. • Saves energy without affecting latency-sensitive traffic by introducing processing delays.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

COMPUTERS & SECURITY ■■ (2017) ■■–■■

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition

Sherenaz Al-Haj Baddar [a], Alessio Merlo [b], Mauro Migliardi [c], Francesco Palmieri [d,*]

[a] The University of Jordan, Amman, Jordan
[b] DIBRIS, Università degli Studi di Genova, Italy
[c] DEI, University of Padova, Italy
[d] Department of Computer Science, University of Salerno, Italy

ARTICLE INFO

Available online

Keywords:
Dynamic traffic classification
Network energy containment
Selective intrusion detection
Distributed intrusion detection

ABSTRACT

In an always connected world, cyber-attacks and computer security breaches can produce significant financial damages as well as introduce new risks and menaces in everyday's life. As a consequence, more and more sophisticated packet screening/filtering solutions are deployed everywhere, typically on network border devices, in order to sanitize Internet traffic. Despite the obvious benefits associated to the proactive detection of security threats, these devices, by performing deep packet inspection and inline analysis, may both affect latency-sensitive traffic introducing non-negligible delays, and increase the energy demand at the network element level. Starting from these considerations, we present a selective routing and intrusion detection technique based on dynamic statistical analysis. Our technique separates latency-sensitive traffic from latency-insensitive one and adaptively organizes the intrusion detection activities over multiple nodes. This allows suppressing directly at the network ingress, when possible, all the undesired components of latency-insensitive traffic and distributing on the innermost nodes the security check for latency sensitive flows, prioritizing routing activities over security scanning ones. Our final goal is demonstrating that selective intrusion detection can result in significant energy savings without adversely affecting latency-sensitive traffic by introducing unacceptable processing delays.

1. Introduction

The number of devices interconnected and always on is growing rapidly and, according to a study from Gartner (G. Group, 2017), will outnumber humans on earth during 2017. Unfortunately, as it has been shown by the Mirai botnet[1], the resilience to intrusion and hacking of these devices has often not grown at the same pace, hence, the thorough sanitization of Internet traffic is a real necessity. Consequently, while in the past network breaches Denial of Service (DoS) attacks and cyber-security threats were considered just as an inconvenience, they have now been recognized as a major cause of financial losses and could morph into deadly threats together with the massive

* Corresponding author.
E-mail addresses: s.baddar@ju.edu.jo (S. Al-Haj Baddar), alessio@dibris.unige.it (A. Merlo), mauro.migliardi@unipd.it (M. Migliardi), fpalmieri@unisa.it (F. Palmieri).
https://doi.org/10.1016/j.cose.2017.12.003
0167-4048/© 2017 Elsevier Ltd. All rights reserved.

[1] https://www.cyber.nj.gov/threat-profiles/botnet-variants/mirai-botnet.

diffusion of ubiquitous connectivity, smart control/sensing devices and Internet of Things technologies. Indeed, many recent experiences focusing on network threats (see, e.g., Labs, 2016; Leder et al., 2009; Paganini) indicate that the worst state-of-the-art menaces are now empowered by botnet infrastructures and most of the malicious activities of these botnets are based on the availability of a huge number of "zombie" agent nodes that produce significant amounts of malicious traffic.

Several experiences prove that such malicious traffic can adversely affect network activity and operations (i.e., Lan et al., 2003; Mallikarjunan et al., 2016), even when the perceived effect does not assume catastrophic proportions as it happened in the case of the widespread Nimda/RedCode infection.

Furthermore, the evidence that malicious traffic is potentially able to change the behavior of networks is the basis on top of which all the modern anomaly detection systems build their work (see Baddar et al., 2014 for some references); in fact, if malicious traffic were completely negligible compared to normal aggregate traffic, no anomalies would be ever generated.

As an additional side effect, the aforementioned processing overhead will increase the energy demand at the network element level, due to the energy-proportional behavior of modern electronic processing devices.

The combination of all the previous factors, namely the need of detecting anomalous network behaviors within the context of an Internet traffic sanitization policy aiming at reducing the amount of undesired traffic loading the network infrastructure, represents an opportunity to leverage the need for security for energy saving purposes. This becomes particularly important in modern mobile and ubiquitous wireless networking scenarios, where battery-powered (and hence energy-constrained) devices assume both the role of routers and traffic control and screening devices in ad-hoc communication infrastructures.

In past works (i.e., Merlo et al., 2016; Migliardi and Merlo, 2011, 2013a, 2013b) we have proved that significant energy savings may be obtained through early suppression of undesired traffic by adopting an *aggressive Distributed Intrusion Detection*. In this context, Intrusion Detection is considered *aggressive* since it continuously performs scavenging for all the resources that are not dedicated to actual routing in order to maximize its effectiveness, and it is *distributed* because each flow is not checked in a single node but is, on the contrary, analyzed along its whole trajectory. It is important to notice that the two characteristics cannot be separated, in fact, adopting an aggressive approach to intrusion detection implies the need to distribute the burden of the analysis to avoid introducing congestion.

However, besides significant energy savings, our studies also showed that the misprediction of the amount of incoming traffic introduces a certain risk of burdening some legitimate flows with unwanted delays. In this work we extend the results presented in Al Haj Baddar et al. (2017), by trying to face this latter challenge by introducing a novel technique capable of dynamically identifying *Latency Sensitive Traffic* (LST) as opposed to *Latency Insensitive Traffic* (LIT). LST comprises two-way traffic flows where packets need to be delivered in almost real-time fashion. In LIT, delivering packets in real-time is not a mandate; delay in delivering packets is tolerable compared to LST applications. Table 1 lists examples of LST and LIT applications. It is worth noticing that Live Audio/Video streaming applications like Internet Radio, fall somewhere in between, as they are more sensitive to regularity in latency, than to latency itself.

Table 1 – LST and LIT examples.

LST examples	LIT examples
• Audio-video conferencing	• Media streaming
• Network/Internet gaming	• Mailing
• Remote control/tele-control	• File sharing
	• Upload/download of contents
	• Web browsing
	• Instant messaging

Such a separation allows focusing on the sanitization of LIT first, thereby guaranteeing undelayed forwarding of LST. At the same time, the suppression of undesired LIT results in freeing a significant amount of router resources that can reveal to be extremely useful later along the following steps of LST traffic flow processing activity, in order to complete the sanitization of the whole traffic without adversely affecting the timing features of LST. This results in a distributed intrusion detection framework where routing is privileged over security screening for LST packets, so that the latter activity, that is computationally heavier, is moved toward the routers that have enough resources over the traffic flows paths, in order to avoid affecting the traffic timing constraints. Even if LIT can be seamlessly delayed in order to perform Intrusion Detection as soon and effectively as possible, the amount of buffer space required for such activity would be unbounded (or bounded only by the maximum possible amount of incoming traffic). Consequently, an effective and more realistic approach implies the distribution of the Intrusion Detection activity over the nodes along the traffic path also for LIT in presence of buffer resources shortage. The estimation of the amount of energy saved considers the fact that, while the cost of sanitization is always required by safety and security reasons and hence cannot be considered a by-product of our scheme, the energy burden associated to the classification of the Internet traffic into LIT and LST is indeed instrumental only to our proposed scheme and hence it must be subtracted from the total quantity of energy that can be potentially saved by relying on it. In order to assess its performance and effectiveness, we apply our scheme to different scenarios based on real world data.

The work is structured as follows: in Section 2 we describe past relevant studies in the field, whereas in Section 3 we describe the proposed selective intrusion detection framework based on separation between LST and LIT. In Section 4 we present the experimental evaluation of the proposed adaptive intrusion detection solution on real world traffic, based on data collected at the University of Padua, by discussing the achieved results and finally, in Section 5 we draw our conclusions from the presented experience and try to sketch some future work perspectives.

2. Backgrounds and related literature

Despite the numerous Network Intrusion Detection Systems (NIDS) whether proposed or implemented, effective solutions have not been fully realized yet. Some well-known solutions adopted the signature-based approach were patterns of malicious behaviors are fed to the NIDS, and hence only intrusions within the scope of such signatures can be detected, while others opted for the behavioral approach. Such systems, instead aim at modeling the behavior of the monitored network to understand its behavior. Upon accomplishing such a far from trivial task, identifying malicious activities that do not comply with the designated normal behaviors would be rather feasible, even if clearly defining the concept of normal network behavior can be extremely challenging. Several signature-based solutions have been used extensively recently, and include Snort[2], Bro[3], and Suricata[4], to mention some of them. Snort is a well-known single-threaded signature-based NIDS that, despite its diffusion and huge users' community, fails at distinguishing application-level protocols. Compared to Snort, Bro better defines malicious signatures, yet, it is limited to Unix-based platforms. Suricata, on the other hand, stands out compared to Snort and Bro, as it uses multi-threading. Recent examples on behavioral approaches are depicted in Ashfaq et al. (2017), Ji et al. (2016), and Lin et al. (2015). There are several types of behavioral systems, some based on a statistical approach and others rely on machine learning (Aburomman and Reaz, 2017). In addition, a NIDS may also adopt strategies based on information theory such as the one presented in Weller-Fahy et al. (2015), as well as use a streaming approach (Desale et al., 2015; Noorbehbahani et al., 2017; Wang et al., 2014). However, as a common feature, all these behavioral approaches require significant amount of computation for their operations, essentially related to deep packet inspection, stateful flow analysis and high level protocol recognition, that in turn are able to heavily tax networking devices, both from the data processing capability and energy consumption perspectives. Several studies targeted the implementation of lightweight NIDS solutions (Li et al., 2009).

New emerging challenges surface and render developing effective NIDS solutions even harder. One of these challenges is energy-awareness: it is now essential for many types of heavy data processing operations, such as packet screening and flow inspection, to behave in a more energy-efficient way depending on the specific application context. This problem becomes even more pressing for Internet of Things (IoT) and wireless devices that are inherently energy-limited. Thus, several recently developed NIDS architectures address energy-awareness through different approaches as illustrated in the systems proposed in Hassanzadeh et al. (2014), Şen et al. (2010), and Tsikoudis et al. (2016). An example of effective distributed intrusion detection technique has been introduced in Migliardi and Merlo (2013b), where energy savings are analyzed for early and later discovery of intrusions. Another related experience has been presented in Viegas et al. (2017) describing novel energy-efficient feature selection and extraction approaches. This study also compared the energy consumption profiles of proposed hardware and software implementations with other machine-learning based approaches and showed that their approach saved more energy. Several recent studies addressed developing energy-aware NIDS solutions for IoT. For instance, the study presented in Khan and Herrmann (2017) depicts a NIDS that allows devices, in healthcare context, to manage reputation information about their neighbors. This approach enables identifying malicious units in an energy-aware way. To evaluate this solution, three variants of the reference scenario have been simulated by comparing their results. Another example is introduced in Sedjelmaci et al. (2016), where a game-theoretic approach and Nash equilibrium were used to implement intrusion detection while saving energy. Simulation results that compared this approach to other recent solutions showed that it is able to achieve comparable detection accuracy using less energy. The study in Muradore and Quaglia (2015) presented an energy-efficient intrusion detection and mitigation architecture for wireless control systems. The proposed architecture relied on selective encryption in order to save energy while attacks are being detected. It also adapted packet transmission rate during attacks according to instantaneous control performance. The simulation results in Muradore and Quaglia (2015) showed that the proposed architecture promptly reacted to attacks with energy saving compared to a default setup in which no intrusion detection was deployed.

Some recent NIDS approaches addressed energy-awareness with low latency; the study in (Tsikoudis et al., 2016), for example, introduced LEoNIDS, a low latency and energy-efficient NIDS. LEoNIDS balanced energy-awareness and low latency goals by identifying the packets that were more likely malicious and gave them higher priority, and thus, LEoNIDS achieved low attack detection latency. Simulation results show that LEoNIDS detected attacks faster by an order of magnitude compared to state-of-the-art solutions while consuming a comparable amount of energy. Other solutions addressed intrusion detection in delay/disruption tolerant networks (DTNs). More precisely, the work in Zhu et al. (2014) discussed a probabilistic strategy for detecting misbehaviors for secure routing in DTNs. The resulting scheme, referred to as iTrust, used a Trusted Authority to assess at regular intervals the behavior of network nodes by collecting routing evidences and performing probabilistic checking. It has been shown that, by choosing the appropriate investigation probability, iTrust is able to enforce the security of routing in DTN scenarios at a reduced cost.

3. Traffic prioritization and energy saving through selective intrusion detection

The basic concept behind the proposed selective intrusion detection strategy is that while LST cannot be delayed by packet inspection and screening activities without adversely affecting its behavior, LIT can. Hence, the proposed distributed intrusion detection architecture in presence of LST privileges routing activities over any kind of heavy security analysis, while for LIT, it gives priority to security analysis over routing. This results in a hierarchy of nodes, where the outermost ones (on top of the hierarchy), typically located on the network border, have the role of separating latency sensitive flows from latency insensitive ones and perform intrusion detection on LIT packets

[2] http://snort.org.
[3] http://bro.org.
[4] http://suricata-ids.org.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

only, by immediately forwarding, and hence distributing, LST to innermost nodes that in turn will perform their security screening activity on significantly reduced amounts of packets, thus without affecting the flows latency. This results in an adaptive approach where the task of identifying malicious traffic is distributed on multiple nodes, where some border nodes, if they have enough residual energy available, can immediately perform security screening on LIT only, by dropping unwanted packets and hence drastically reducing the load toward innermost nodes, whose screening and inspection will be limited to LST so that they can directly route LIT, without any additional computational burden. Such behavior, by distributing the load on multiple nodes, can lower their average processing activity, so that they can better benefit from power scaling behavior of modern processing devices, with significant energy savings on the overall network. As a further optimization, innermost devices perform traffic inspection only if they have enough residual energy to do so, otherwise they limit their activity to routing alone, by delegating the inspection activity to the next step nodes on the flow path.

In the following we present in detail the whole architectural framework by illustrating how distributed adaptive intrusion detection is implemented with F-Sketure, a new version of Sketure, the sketch-based packet analysis tool introduced in Baddar et al. (2016), specifically developed to operate on a per-flow basis. We also show how the F-Sketure system is able to effectively perform traffic classification, by separating the LST and LIT classes, and try to quantify the energy savings that can be achieved through a better load distribution.

3.1. *Separating LST from LIT traffic through F-Sketure*

First of all, the basic step needed for implementing the above adaptive distributed intrusion detection approach is discriminating LST from LIT on outermost nodes, in order to allow further processing depending on the traffic class and available energy/processing capabilities, as previously described. Packets have to be classified and screened, characterized by specific *source* and *destination* addresses, and tagged according to their latency sensitivity. For this purpose, we developed F-Sketure, a per-flow version of Sketure, a traffic analyzer that must run on outermost (border) nodes and classifies packets in flows as LST or LIT, without jeopardizing users' privacy. The abstract architecture of this tool is illustrated in Fig. 1.

As depicted in Fig. 1, F-Sketure aims at summarizing the behavior of each flow, alongside its inverse, in the monitored

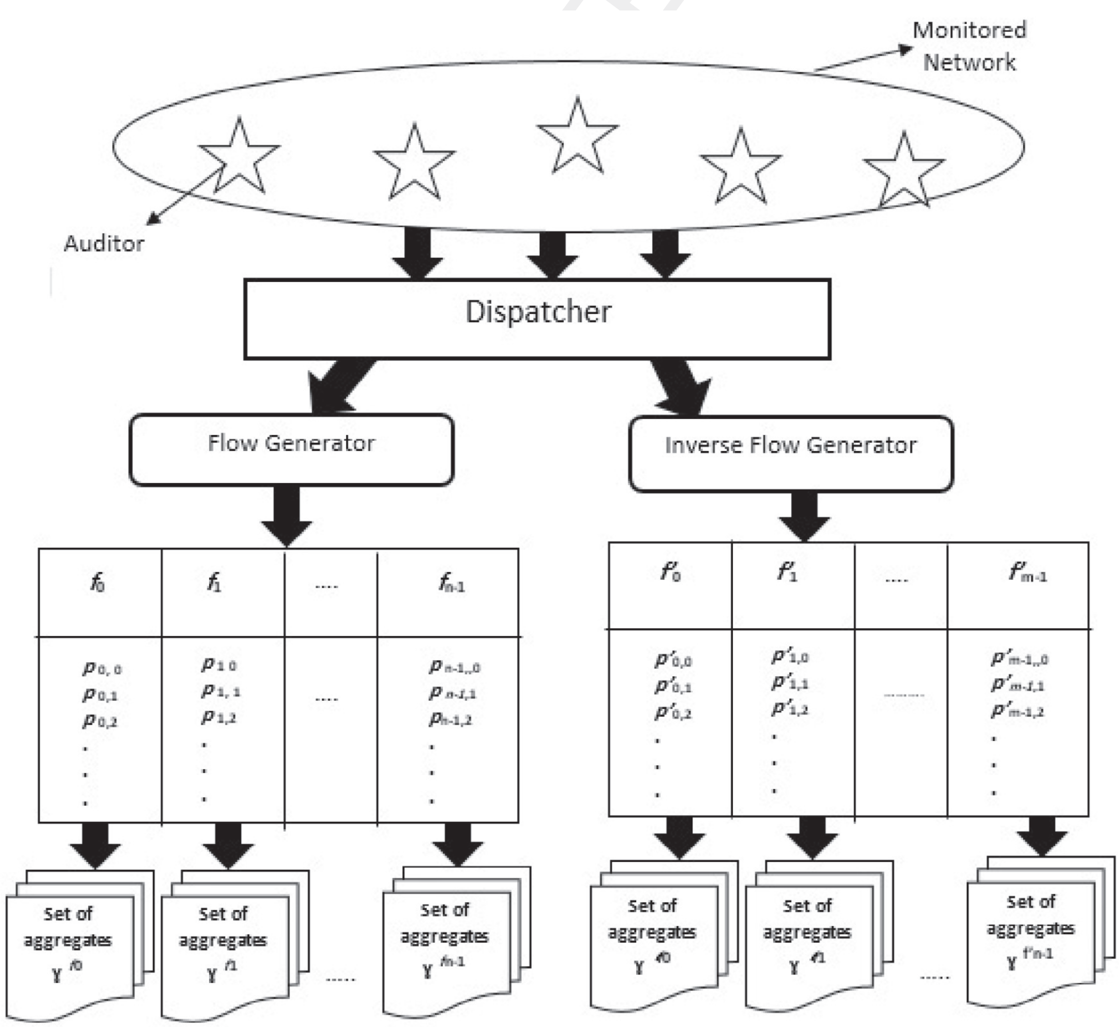


Fig. 1 – The F-Sketure abstract architecture.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

network. To achieve its goal, F-Sketure has to properly identify traffic flows, and then for each detected flow it needs to summarize some of the most discriminating features of the flows' packet headers in order to reliably discriminate LST from LIT. F-Sketure inspects each individual packet sent or received without violating users' privacy through auditors. Auditors would passively sniff exchanged packets and communicate only a subset of the their headers information, i.e., features, to the dispatcher process, after obfuscating the source and destination IP addresses. Here a packet p would be of the format (*source*, *destination*, t_i, $v^0(t_i)$, $v^1(t_i)$, . . ., $v^{|\mathcal{F}|-1}(t_i)$), where *source* and *destination* designate the packet's sender and receiver obfuscated addresses respectively, t_i denotes the packet timestamp, and $v^k(t_i)$ denotes the value of the k^{th} feature at time t_i, and $i \in \{0, 1, 2, \ldots\}$. The role of the dispatcher process is to forward obfuscated packet headers to the Flow Generator process, or, alternatively, to the Inverse Flow Generator process if the obfuscated packet is in the format (*destination*, *source*, t_j, $v^0(t_j)$, $v^1(t_j)$, . . ., $v^{|\mathcal{F}|-1}(t_j)$). The Flow Generator summarizes information conveyed by the set of packets that comprise flow f over equally-spaced time intervals, each of which is denoted by g. The summarized features considered in this case span the average packet size s, the average packet count c, and a tag field T, that denotes the traffic class (LST or LIT) to which the packets in f belong. An aggregate denoted by γ_g^f, depicts the statistical summary produced by the Flow Generator during time interval g for flow f. Each aggregate is of the form

$$\gamma_g^f = \langle g, s, c, T \rangle \tag{1}$$

After G time intervals, the Flow Generator compiles the set of aggregates γ_G^f that describe the behavior of flow f over G, where,

$$\gamma_G^f = \{\gamma_g^f, g \in \{0, \ldots, G-1\}\} \tag{2}$$

and based on the set of aggregates γ_G^f, flow f, will be denoted by the tuple

$$f = (source, destination, T^f, \gamma_G^f) \tag{3}$$

where T^f denotes the tag of flow f over G and likewise the tag field of a packet p, it can be set to either LST or LIT. When the flow generator designates flow f as LST or LIT, all packets comprising this flow over G will be labeled accordingly. A similar process is carried out by the Inverse Flow Generator where flow f', the inverse of flow f, designates the traffic flowing in the opposite direction, to be represented in the form

$$f' = (destination, source, T^{f'}, \gamma_G^{f'}) \tag{4}$$

We also denote a given aggregate in f' by $\gamma_g^{f'}$ where

$$\gamma_g^{f'} = \langle g, s', c', T \rangle \tag{5}$$

Obviously, both flow f and its inverse flow f' have the same tag value, so either both flows are LST or both are LIT over a given period of time G. Initially, the value of the tag T in a given packet p is set to undefined. Moreover, in order to avoid any security processing overhead, a packet has to be checked exactly once during its life span. Thus, each packet header comprises a binary status flag *Checked* indicating whether or not the packet has been already checked for security purposes. On the other hand, a flow may be re-tagged several times if its dynamic features change significantly.

Tag and checked flag values in packet headers can be properly conveyed within the label field in Multi-Protocol Label Switching (MPLS)-enabled routing devices; where only 2 bits out of 20 are necessary, one for the checked flag and one for the tag. Alternatively, these flags can be inserted directly in some unused fields of the IP packet header. More precisely, the high-order bit of the IP fragment offset field (the so called evil bit) can be used for the checked flag. This bit, officially defined as unused in the IP header, has been mentioned in RFC 3514 (released on 1st April of 2003, the April Fools' Day), with humorous intents, for flagging packets that have malicious intent, by recommending security enforcement devices to drop inbound packets with this bit set. In our specific case, the evil bit will explicitly flag packets that are yet to be checked. Similarly, for conveying the tag information we can use the original Type of Service (TOS) bits (or the co-located DSCP ones, obsoleting TOS in RFC 2474), by using a zero pattern for LIT traffic and a 111000 (CS7 for DSCP or Network Control for TOS) for LST. If these bits were already previously assigned on their origin, and F-Sketure confirms that a given flow f can be classified as LST, then the existing value (nonzero) can be left unaffected to represent a packet belonging to the LST traffic class. In order to identify the tag value for a given flow f, F-Sketure performs two tests on values s, and c within a given aggregate γ_g^f in flow f, and on their corresponding values in its inverse f', as LST flows are typically two-way, compared to, for example, streaming flows, which are not latency sensitive, according to our definition. More precisely, F-Sketure considers an aggregate γ_g^f to be latency sensitive if it meets either one of the two following conditions:

1. If s and $s' \in [\delta_1 \bar{S}_{voip}, \delta_2 \bar{S}_{voip}]$ and c and $c' \in [\delta_1 \bar{C}_{voip}, \delta_2 \bar{C}_{voip}]$, where $\bar{S}_{voip}$ and $\bar{C}_{voip}$ are, respectively, the average packet size and average packet count of the VOIP classes defined in industrial VOIP standards[5], while δ_1 and δ_2 denote the error margins.
2. If s and $s' \in [\delta_1 \bar{S}_{data}, \delta_2 \bar{S}_{data}]$ and c and $c' \in [\delta_1 \bar{C}_{data}, \delta_2 \bar{C}_{data}]$, where $\bar{S}_{data}$ and $\bar{C}_{data}$ are the average packet size and average packet count from the previous three aggregates respectively, while δ_1 and δ_2 denote the error margins.

As clearly shown from the two previous conditions, an aggregate is considered latency sensitive if both itself and its corresponding instance for the inverse flow exhibits a VOIP typical behavior, and/or if they exhibit a temporal regularity pattern. Otherwise, the aggregate is considered latency insensitive. Thus, the aggregate tag field T, is set to LST if one of the aforementioned conditions holds, and set to LIT otherwise. When a given aggregate is tagged as either LST or LIT, its corresponding flow and inverse flow alongside all the packets that comprise the corresponding aggregates are tagged

[5] http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/7934-bwidth-consume.html.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

ARTICLE IN PRESS

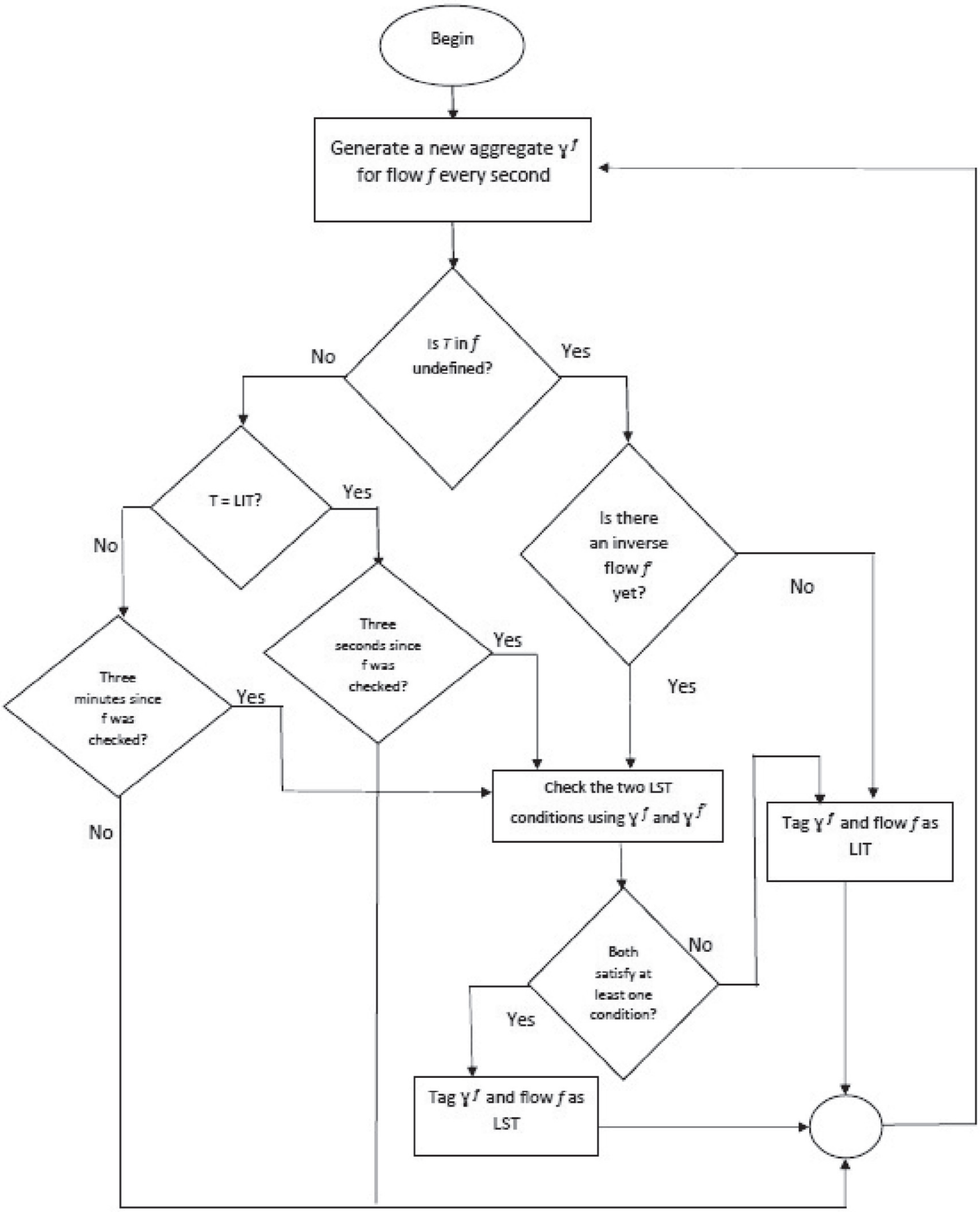


Fig. 2 – The F-Sketure flowchart.

accordingly. After tagging the very first aggregate, all subsequent aggregates together with their comprising packets retain the same tag, until the next classification window arrives after a given number of time units; at that time, the two LST conditions are checked again using the current aggregate, then the aggregate tag, together with the flow and its current and upcoming packets either retain the previous value or obtain a new one. Fig. 2 depicts the process via which F-Sketure tags the flows it identifies.

3.2. *The selective distributed intrusion detection architectural framework*

F-Sketure uses a multi-resolution time window to implement packet tagging; it trades off accuracy for cost and tries to prioritize LST without overlooking the dynamic nature of flows. The first aggregate in a flow gets examined alongside its counterpart in the inverse flow to see whether it meets either one of the aforementioned LST conditions; if that is the case,

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

its T field is set to "LST", otherwise it gets set to "LIT". Then, every 3 seconds, the LST conditions get re-evaluated using the current aggregate in *f*. However, if a flow gets tagged as "LIT" at any given instance of time, it gets re-evaluated once every 3 minutes. This re-evaluation mechanism saves tagging time and energy as it refrains the ingress router from tagging frequently, while prioritizing LST flows. Reducing the tagging time window would render F-Sketure tagging more accurate, however, it would imply consuming more processing time and energy at the ingress node. The proposed selective distributed intrusion detection architecture requires the presence of at least a head router, which is a router endowed with a packet inspection unit running the F-Sketure packet analysis engine, followed by a set of Intrusion Prevention Routers (IPR) that are only able to perform routing and security analysis. To clarify the operations of our approach, we assume time is split into slices, denoted by *t*.

The role of head routers is performing preliminary classification of each incoming packet as either LST or LIT, by properly tagging and then forwarding it to the next IPR node. The head router can also inspect LIT traffic for malicious packets if it has enough residual energy resources and hence computational capabilities. Clearly, each packet already analyzed for security, is marked as checked by using the proper flag in its header. On the other hand, a typical IPR node routes, and possibly, analyzes unchecked incoming packets by screening for security threats and discarding dangerous packets/flows when needed.

As illustrated in Merlo et al. (2016), an IPR routing node *i* has not only the task of routing incoming packets, but is also capable of identifying malicious incoming packets. The fundamental role of its pre-processing unit, is determining the maximum amount of packets that can be analyzed at a given time slice *t*, whereas its Intrusion Prevention System (IPS) unit is responsible for identifying malicious incoming packets when needed. Since its latency will be not affected by the security analysis process, LIT traffic will be assigned a higher priority for being considered for analysis. Consequently, LST traffic will be analyzed before being routed, only after all LIT traffic within the same slice has been analyzed.

Each node *i* has an amount of energy that is equally split among time slices *t*. Thus, the amount of residual energy that a router has available at *t* is denoted by $E_i(t)$ and represents the power *P* that can be drawn at a specific time *t*. As each packet gets routed and/or analyzed, the available amount of energy $E_i(t)$ reduces accordingly. Clearly, the main operating priority of an IPR node is routing packets rather than analyzing them. Hence, it is not typically possible to analyze all incoming packets within a slice at router *i*, Therefore, different packets will be analyzed by different routers along the path to the destination, which adaptively distributes the intrusion detection process throughout the network.

3.3. The energy consumption model

In modeling the energy consumption characterizing the forwarding or traffic inspection activity of IPR or head nodes we considered the energy-proportional behavior of modern electronic devices, whose energy demand is strongly influenced by the actual load, which often influences its operating frequency and voltage level.

Typically, a linear model can be used to describe how the router requires power at different loads. Starting from real world measurements (Chabarek et al., 2008), we can see that, when a router is turned on but is still *idle*, it consumes about half of its total power. When the incoming packet load grows, power consumption linearly increases together with it, up to an upper limit value that is reached when the router is fully loaded. The slope describing the growth of power consumption function with regard to the incoming traffic load is defined by a *scaling factor SF*, that can be measured in *W/Gbps*.

As a general criterion, the power consumption P_r of a router, associated to its packet forwarding activity, can be expressed as a function of its load (*d*) by using the following equation:

$$P_r = SF \cdot d + P_{idle}, \tag{6}$$

where P_{idle} is the fixed power consumption of the router when idle, accounting for a significant part of the total power required to keep the device on, mainly depending on the switching matrix and control circuitry. Clearly, the greater the node, the more complex its circuitry is, and hence, the higher its static power demand. On the other side, the dynamic component depends on the traffic load and on the interfaces speed, characterizing the scaling factor, where faster interfaces require less power per bit than slower ones (Ricciardi et al., 2011, 2013).

On the other hand, when considering the power P_a required by the packet inspection/analysis activity we have to consider that it is essentially due to the associated processing work, that in turn is due to state switching of electronic gates and is proportional to their operating frequency *f* as well as to the square of their supply voltage *V*:

$$P_a = \approx \frac{1}{2} V^2 fC \tag{7}$$

where *C* denotes the average capacitance and the 1/2 factor is a constant value (often indicated with γ) that depends on the switching activity. Operating at a lower frequency can significantly reduce the energy consumption by also allowing the use of dynamic voltage scaling (DVS) for reducing the operating voltage. This allows energy consumption to scale quadratically with operating frequency *f*.

In a typical circuitry performing deep packet inspection, γ will depend on the number of gates flipping at each new packet and hence on the capability of the processing system to match the flow structure with specific templates, by adapting to them through comparison operations. That is, the lower the degree of organization in the packet stream, the higher will be the switching effort required by the hardware devices. It is straightforward to consider that P_a is typically much greater than P_r.

3.4. The IPR-router model

Let $M_i(t)$ be the maximum number of packets that can be analyzed by the IPR *i* at a specific time slice *t*. Let also $C_i(t)$ be the processing capacity of IPR *i* at the time *t*, and $X_i(t)$ be the expected number of incoming packets at the time *t*. In addition, let us assume that routing a single packet requires *R* energy

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

units. According to the considerations presented in the previous section and in Merlo et al. (2016), the energy needed for performing security checks on a packet is higher than the one needed for routing it. That is, the energy needed for analyzing a packet is α times the energy needed for routing it, with $\alpha > 1$. If we also indicate as $B_i(t)$ the number of incoming packets buffered by IPR i at the start of time slice t, then we can determine $M_i(t)$ as follows

$$M_i(t) = \frac{C_i(t) - (X_i(t) + B_i(t))}{\alpha} \tag{8}$$

Let us define $A_i(t)$ as the number of packets analyzed during the time slice t, $K_i(t)$ as the number of already checked packets observed during t, $N_i(t)$ as the number of incoming packets observed during t, and r_{LIT} as the ratio of LIT packets observed so far. Upon receiving a new packet p with the proper tag T and already screened flag *Checked* in its header, one of the following scenarios will happen:

1. If *Checked* = 0 or 1, T = LST or LIT, and $E_i(t) < R$, then the packet p is buffered until the next time slice, and $N_i(t)$ is incremented by 1.
2. If *Checked* = 1, T = LST or LIT, and $E_i(t) \geq R$, then packet p is routed. The residual energy is $E_i(t) = E_i(t) - R$, while $N_i(t)$ and $K_i(t)$ are each incremented by 1.
3. If *Checked* = 0, T = LST or LIT, $A_i(t) = M_i(t)$, and $E_i(t) \geq R$, then packet p is routed without being analyzed, $E_i(t) = E_i(t) - R$, while $N_i(t)$ is incremented by 1.
4. If *Checked* = 0, T = LIT, $A_i(t) < M_i(t)$, and $E_i(t) \geq (1 + \alpha)R$, then packet p is analyzed, if it is malicious router i drops it, otherwise, its *Checked* flag is set to 1, and the packet is routed. The residual energy is $E_i(t) = E_i(t) - (1 + \alpha)R$, while $A_i(t)$, $N_i(t)$, and $K_i(t)$ are each incremented by 1.
5. If *Checked* = 0, T = LST, $A_i(t) < M_i(t)$, and $E_i(t) \geq (1 + \alpha)R$, and $K_i(t) < r_{LIT}N_i(t)$, then packet p is routed without analysis, as $K_i(t) < r_{LIT} * N_i(t)$ implies that not all LIT traffic has been already analyzed. The residual energy is $E_i(t) = E_i(t) - R$, while $N_i(t)$ is incremented by 1.
6. If *Checked* = 0, T = LST, $A_i(t) < M_i(t)$, and $E_i(t) \geq (1 + \alpha)R$, and $K_i(t) \geq r_{LIT}N_i(t)$ then packet p is analyzed, as $K_i(t) \geq r_{LIT}N_i(t)$ implies that all LIT traffic has been already analyzed. If packet p is found to be malicious, router i drops it, otherwise, its *Checked* flag is set to 1, and the packet is routed. The residual energy is $E_i(t) = E_i(t) - (1 + \alpha)R$, while $A_i(t)$, $N_i(t)$, and $K_i(t)$ are each incremented by 1.

In summary, the behavior of a given IPR node depends on the incoming traffic compared to the node's own capacity. If incoming packets are less than the node's maximum routing capacity, then it immediately routes incoming LST packets, and then applies an intrusion detection technique to as much as it can handle from remaining LIT packets. However, when the router falls short of energy, it limits its actions to only routing incoming packets without analysis. This implies that malicious packets get dropped as soon as they are discovered which is expected to reduce the overall energy consumption. The expected amount of traffic $X_i(t)$ at a given time slice t, can be estimated by using the average of the actual incoming packets from the N most recent $N_i(t)$ values, so that

$$X_i(t) = \frac{N_i(t-1) + \ldots + N_i(t-N)}{N}$$

Initially N is set to 1, then 2 and it grows up to the desired number. In our experiments we set N to the value of 5.

3.5. The head router model

We can observe that the head router in our architecture can use a similar model. Essentially, it estimates the maximum number of packets $M_h(t)$ it can tag at the beginning of each time slice t, and behaves accordingly. The value of $M_h(t)$ can be determined as follows

$$M_h(t) = \frac{C_h(t) - (X_h(t) + B_h(t))}{\beta}$$

where $C_h(t)$ represents the head node's capacity, $X_h(t)$ is the expected number of packets at time slice t, and $B_h(t)$ is the number of packets buffered at the head router at the beginning of t. Moreover, the value β can be used to represent the ratio of the energy needed for tagging one packet to the energy required for routing it. Yet, the behavior of the head router is a bit different, since it needs to tag all incoming packets before they get routed. The head router is presumably the first node, typically located on the network border, that observes the packets, so that, all its incoming packets arrive initially untagged. Since we need all packets to be tagged and use IPR nodes with capacity $C_i(t)$ for further processing them for security analysis purposes, we have to use more capable header nodes in order to be able to deliver tagged packets at the underlying IPR maximum capacity. To achieve this goal, the following inequality must be satisfied

$$C_h(t) > (1 + \beta)C_i(t)$$

As the reduction of energy consumption on the overall network is pursued, it is recommended to choose head routers characterized by a capacity $C_h(t)$ equal to $(1 + \beta)C_i(t)$. We can also consider that the smaller the value of β is, the higher amount of energy will be saved.

4. Experimental evaluation and results analysis

The validation of our scheme has been performed through several experiments based on data extracted from real traffic. Our experiments allowed us to evaluate the performance of our scheme both in terms of energy savings and in terms of latency imposed to the packets flowing through the system. First, we now describe the dataset we have adopted to evaluate the traffic mix in terms of LST vs. LIT, then we illustrate the F-Sketure tool by both providing some details about its implementation and depicting its performance. Second, we illustrate how we used the information obtained from the real traffic dataset to define extended simulations capable of probing the performance of our scheme in different scenarios. Finally, we illustrate and discuss the results of our simulations.

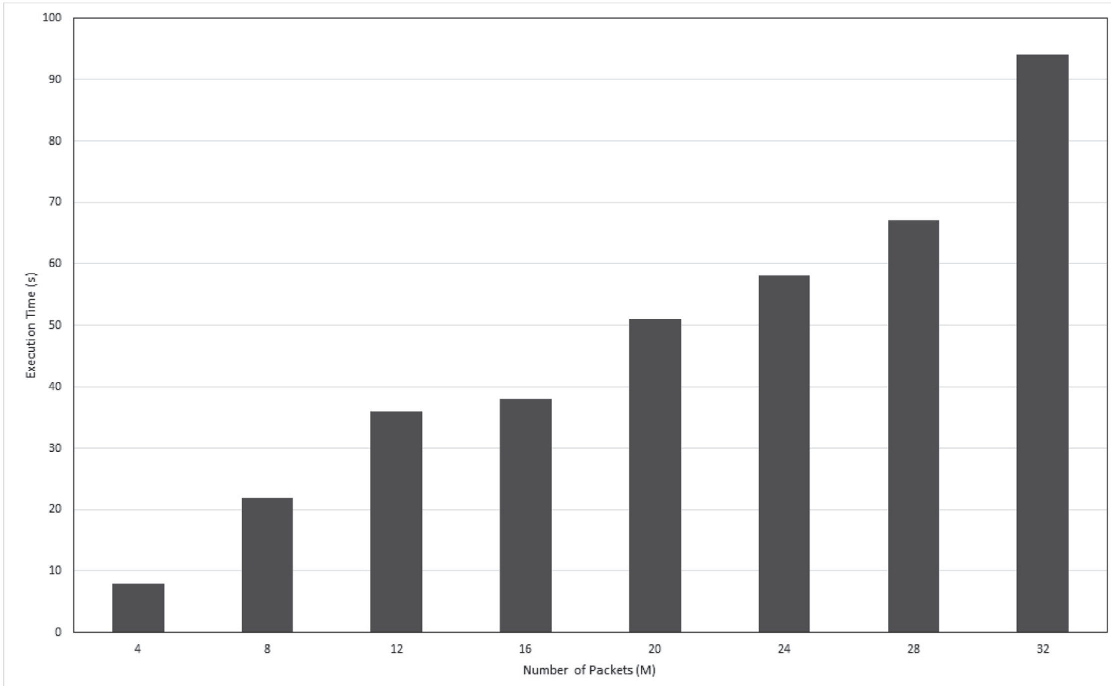


Fig. 3 – F-Sketure execution time.

4.1. *The padua dataset sample*

In order to base our assessment of the performance capabilities of F-Sketure on real traffic data we selected a 10-hour packet trace from the Padua dataset (Baddar et al., 2016). In this dataset, for each packet we explicitly collected:

- A timestamp in the form *hh:mm:ss*;
- source and destination addresses, properly obfuscated for privacy reasons;
- the packet size

This subset of the whole dataset contains 1.48 GB of the above listed packet information captured uninterruptedly in the time window comprised from 1:00 am until 10:00 am on Friday April 10th 2015.

4.2. *The F-Sketure implementation*

We implemented the The F-Sketure tool using the Java programming language. F-Sketure, when processing the previously mentioned traffic data, produces aggregates every second, and re-evaluates their tags every 3 seconds. In our experiments, the error margin parameters δ_1 and δ_2 have been set to 0.95 and 1.05, respectively.

Our experiments show that the traffic in our dataset can be classified as 3% LST and 97% LIT. At the same time, our experiments show that packet tagging once the aggregations have been produced requires only 0.5 ns; re-evaluating all the aggregation classes requires 0.273 ms, more than 500 times the time needed to tag a single packet. This level of performance is compatible with our experiments. Furthermore, it is important to notice that, with the exception of the first evaluation, all the subsequent re-evaluation can be done in parallel with the tagging of packets using the previous set of classes. Hence, the re-evaluation does not represent a significant performance bottleneck in the flow of packets. Nonetheless, for energy saving purposes, in future work we will assess the feasibility of reducing the frequency at which the aggregation classes are re-evaluated.

In Fig. 3 and Fig. 4 we show the time and memory consumption required for running F-Sketure on the above mentioned dataset. As depicted in Fig. 3, the time it took F-Sketure to read packet headers, identify flows, generate aggregates, and tag packets ranged from 8 seconds to handle 4M packets up to 94 seconds to handle 32M packets. With regard to F-Sketure memory consumption, Fig. 4 illustrates that the amount of memory consumed to process packets ranged from 2.5 GB to 4.5 GB.

4.3. *The simulation setup*

In our simulations we considered the flow along a single path of length 10 routers. Even if the number of hops a packet makes to get to its destination is usually higher, our goal here is not to check if we can achieve full traffic sanitization in just ten hops, our goal is to assess that we have some effect on energy saving while reducing the amount of malicious traffic and without jeopardizing the QoS for LST; for this reason, the fact that we do not achieve full sanitization of the traffic in just 10 hops is not significant. At the ingress node, we have the packet tagging engine based on the F-Sketure tool. After that, the whole traffic flows through the adaptive IDS enabled IP routers. Fig. 5 depicts the arrangement of this scenario. While F-Sketure operates at the head of the routing chain to provide the next routers with tagged packets, the remaining IP routers

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

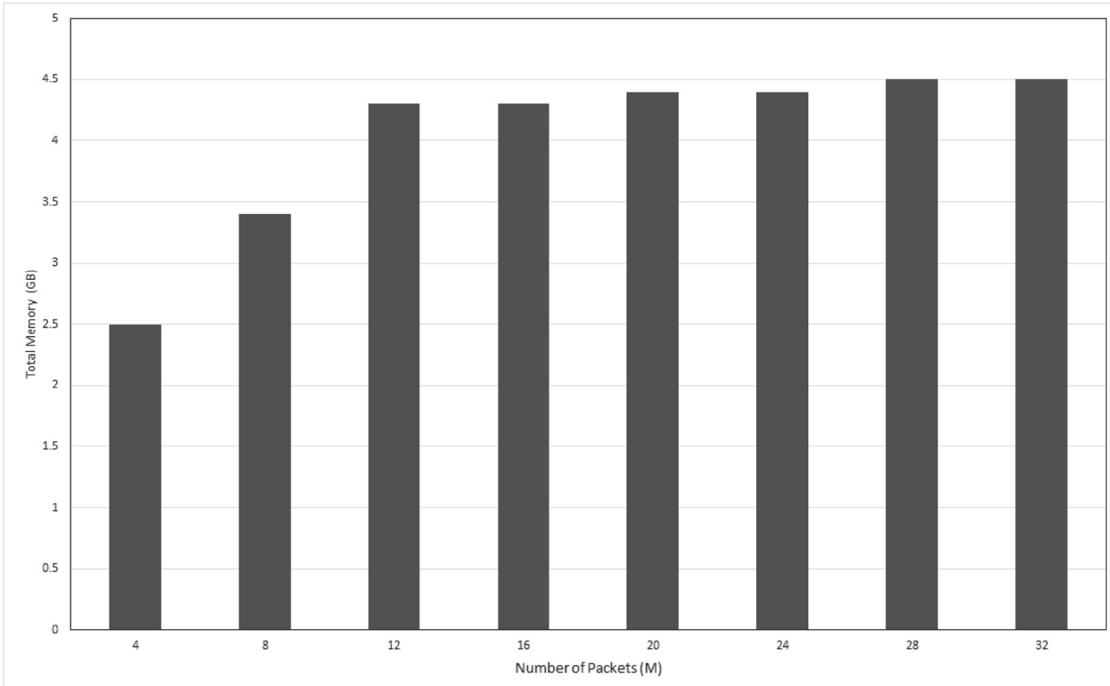


Fig. 4 – Memory consumed by F-Sketure.

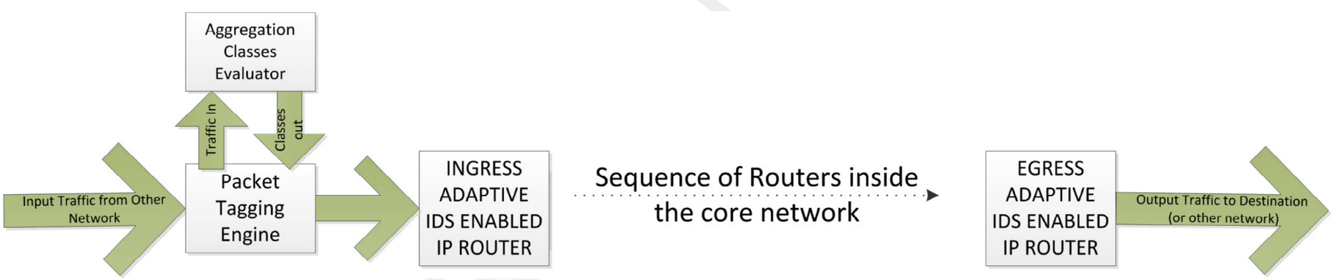


Fig. 5 – The network setup used in the simulation.

are fully dedicated to the task of routing packets, and, if possible, analyzing them according to the process depicted in section 3.4. In our simulations, we assumed the routers to be capable of processing 2 million packets per second and we adopted a 1 ms time slice for calculating the predicted load and the predicted energy available for security analysis. At first, we assumed to use the Padua dataset as the input traffic of our simulation, but we observed that:

1) the peak bandwidth of the Padua dataset was much lower than the one we expected to inject in the system;
2) the percentage of LST was very low, just a 3%, and showed no significant changes over time;
3) the traffic was very regular and not capable of injecting into our system significant misprediction errors.

For these reasons, we decided to adopt a synthetic trace for our simulations. We used as a traffic shaping function the shape of the traffic flowing through the Milan Internet eXchange (MIX)[6]; then, we added to that a Gaussian distributed burstiness to make it more difficult to predict. We adopted Suricata signature-based intrusion detection system as the intrusion detection technique deployed in simulated IP routers, and, according to Merlo et al. (2016), α is set to 4.5. As for β, according to the experiments executed using F-Sketure, it was set to 0.35.

We ran our simulation at steady state (i.e., after the flow of packets had reached the last router and without stopping the flow getting into the first router) for 10, 20, 30, 40, 50 and 60 seconds; As the different durations showed no effects on the observed variables, we will describe the results for the longest duration only. We used a fixed malicious packet ratio set to 6%. We varied LST ratio from 5% to 20%; we decided to stress our system with percentages of LST up to almost 7 times the ratio we have observed in the real world traffic from the

[6] http://www.mix-it.net/statistics/cgi-bin/14all-Totale_globale.cgi?log=totaltraffic_global.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

Padua dataset. We varied the amount of traffic entering the system from 50% of the nominal capacity of the routers (actually 1 million packets per second) up to the full nominal capacity of 2 million packets per second with an intermediate step of 1.5 million packets per second. Besides, we changed the percentage of time slices in which traffic was bursty (i.e., from 1/5 to 1/2 positively or negatively different from the traffic shape observed in MIX) to values of 50% and 75%. We introduce burstiness because a regular traffic will be correctly predicted and there is hence no risk to spend in analysis more resources than those actually available. Burstiness stresses the capabilities of our system forcing prediction errors. For each simulation, we calculated the average number of:

1) routed packets;
2) analyzed packets;
3) malicious packets dropped;
4) predicted incoming packets;
5) actual incoming packets;
6) LIT packets delayed to next time slice;
7) LST packets delayed to next time slice.

Furthermore, we added another variation to the scheme. Since the direct routing or analyzing decision is taken once per slice, it is possible that a concentration of unpredicted LST packets toward the ends of the slice leads to the delay of those same packets. To minimize that possibility we tried to add a *safeguard* to our scheme; to do so, in each time slice we tried to not use the whole amount of resources available once the routing of all LST packets have been taken into account, but to use that amount reduced by 10% of the available resources. This *safeguard* obviously reduced the level of aggressiveness in hunting down the bad packets and the amount of energy saved, yet, it also reduced the likelihood of delaying LST packets.

4.4. Discussion of the simulation results

We first analyze the effect of different percentage of LST traffic on the energy saving our methodology can generate in the 10 routers. Fig. 6 shows the percentage of energy saved with different percentages of LST traffic, with different amounts of input traffic, but with the level of burstiness fixed at 75%. Across different LST ratios, the energy saving percentage shows only minor fluctuations. On the contrary, the level of energy saved shows a significant dependency on the size of the input, varying almost an order of magnitude when the amount of traffic injected into the system ramps from 50% of the nominal routing capacity of the nodes up to 100% of that same capacity. For these reasons, we will now consider only the 5% LST case, actually the one that is most similar to what we observed in the Padua dataset, while we will focus on the effects of different input sizes and different level of input burstiness to study how our methodology impacts on the delay imposed to LST packets.

We now analyze the number of LST packets that incur in a delay with different amount of traffic and with different levels of burstiness. Fig. 7 shows that our methodology is capable of keeping the number of LST packets than incur a delay to less than 1 every 10,000, even when the input traffic may go occasionally over the router's nominal capacity (i.e., when input is 100% of nominal capacity and there are one half of the time slices that are bursty).

We now analyze the system behavior when we inject a traffic that is bursty in 75% of the time slices. Fig. 8 depicts the system behavior in this second case. It is possible to see that, when

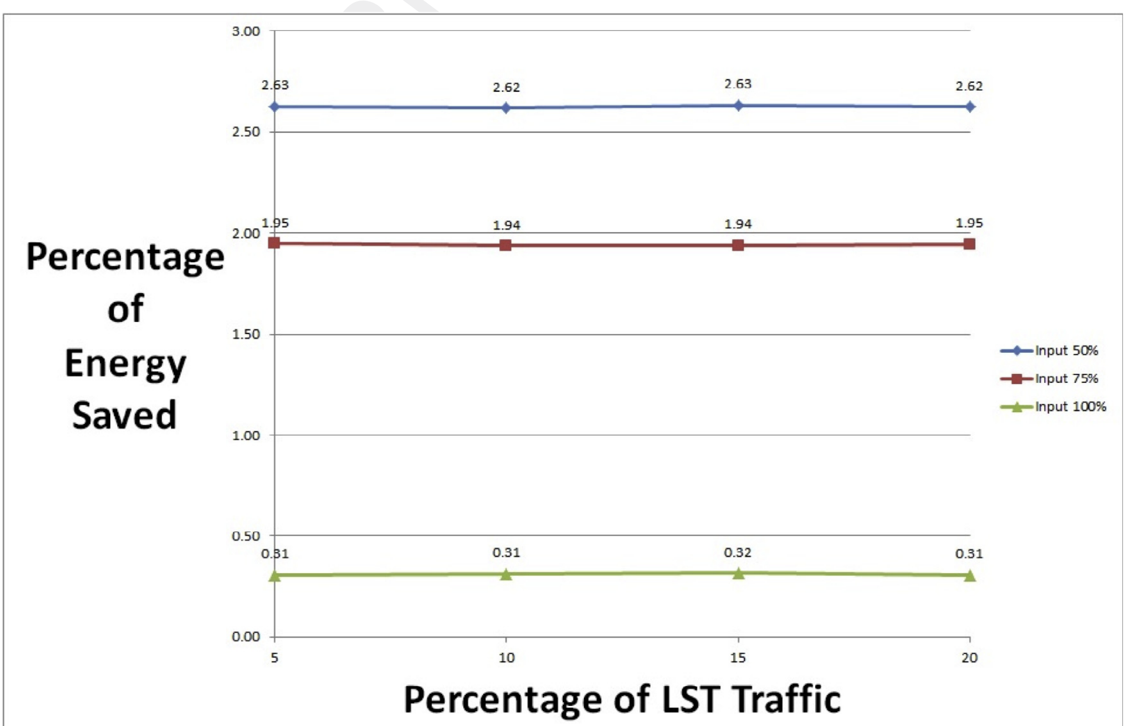


Fig. 6 – Percentage of energy savings with different LST percentages.

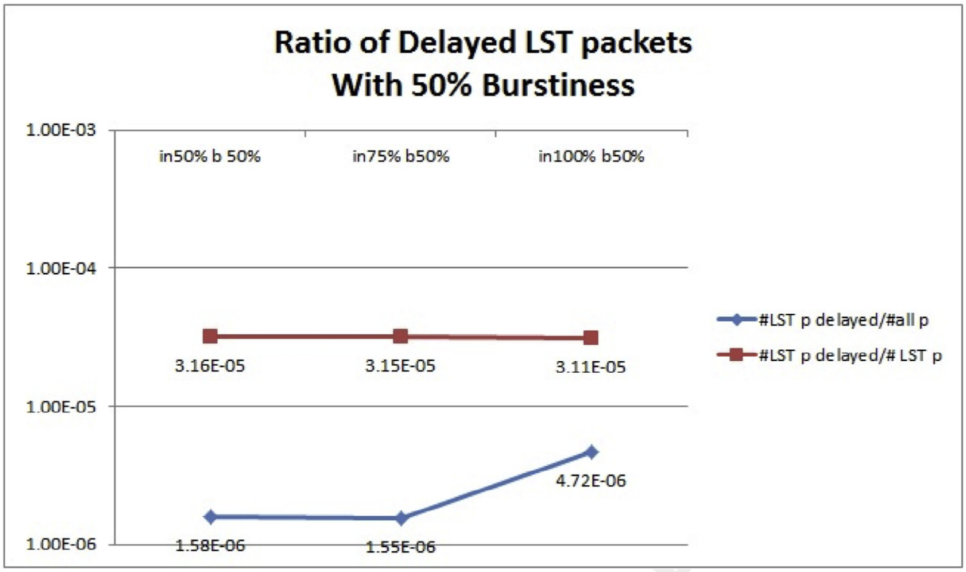


Fig. 7 – The ratio between LST packets that incur into a delay and all packets or LST packets. Burstiness is 50%.

the size of the input traffic goes beyond 50% of the nominal capacity of the routers, the increased level of burstiness causes an increase of the number of LST packets that incur a delay of almost three times. Even if we can still keep the number of delayed LST packets around 1 every 10,000 packets traveling through the network, we now introduce a variation of our scheme that allows overcoming the problems caused by the increased level of burstiness. Then, we evaluate the impact on the energy saving introduced by this variation in our methodology. Obviously, the increased level of burstiness causes an increase in the number of misprediction errors and makes the system use more resources for analysis than the ones that are actually available. When this occurs, some packets are not routed in time and, as we update the decision about how many LST packets to route before doing any analysis only at the beginning of each time slice, some of the delayed packets may be LST ones. To reduce this problem, we introduce a simple *safeguard*, that is, we systematically underestimate the amount of resources available for analysis by 10%. So, a router will analyze packets only if its estimation shows that its load will be less than 90% of its nominal capacity.

The amount of LST packets delayed with a 50% level of burstiness in the input traffic is depicted in Fig. 9. It is easy to see that, at this level of burstiness, the presence of the *safeguard* has no impact until the size of the input traffic reaches the nominal capacity of the routers. At that point, it is capable of introducing a minor reduction in the number of delayed LST packets in comparison to the situation that can be observed in Fig. 7. We now analyze the effects when the level of burstiness is set to 75%. In Fig. 10 it is possible to observe that the pres-

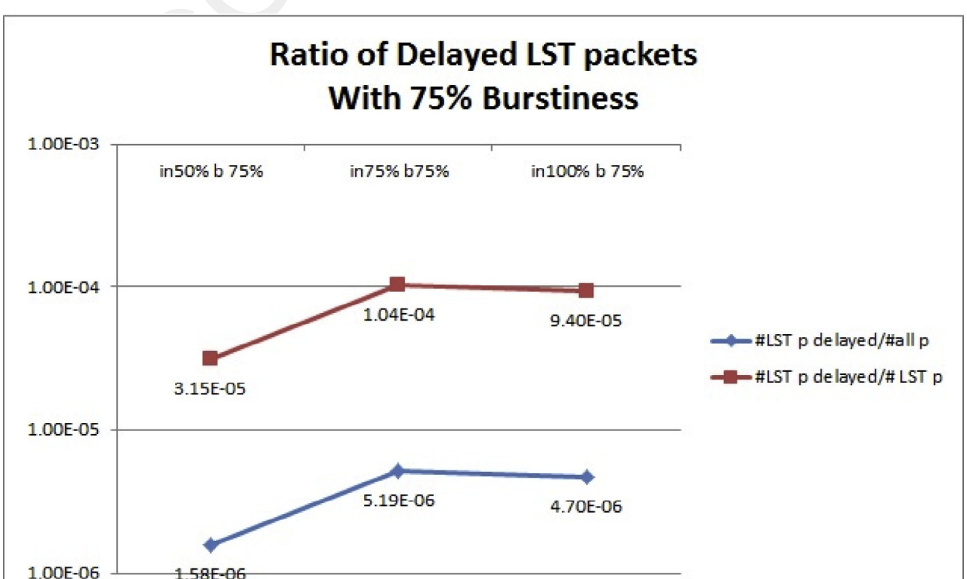


Fig. 8 – The ratio between LST packets that incur into a delay and all packets or LST packets. Burstiness is 75%.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

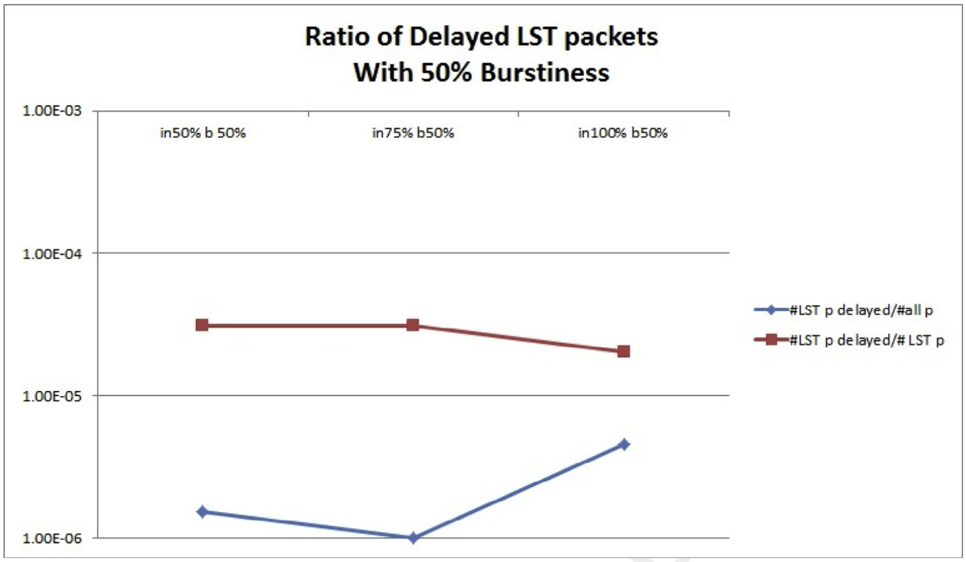


Fig. 9 – The ratio between LST packets that incur into a delay and all packets or LST packets. Safeguard set to 10%. Burstiness is 50%.

ence of the *safeguard* delays the increase in the number of delayed LST packets that could be observed in Fig. 8 when the traffic size reached 75% of the routes nominal capacity. On the contrary, it even reduces it for that traffic size. However, when the input traffic size goes to 100% of the router nominal capacity, the *safeguard* has no effect. We now analyze the effect of the safeguard on the energy savings that the system can achieve. As Tables 2 and 3 clearly show, the introduction of the *safeguard* has a steep price energy-wise. At best it halves the percentage of energy the system can save, while the worst case is almost three orders of magnitude. At the same time, it is important to notice that when the reduction in the energy savings is at its minimum, the effect of the safeguard on the amount of LST packets that are delayed is also negligible, but it is not true that the effect on the number of packets delayed increases as it increases the energy cost. In Table 4 we provide a synthesis of the effects of the *safeguard* both on the energy costs and on the number of delayed LST packets. It is obvious that the energy cost is always larger than the benefit related to the reduction of LST packets delayed, however, our experiments also show that there is a local maximum in the convenience to apply the *safeguard* when the traffic size is 75% of the router's nominal capacity and the traffic has a 75% level of burstiness. Hence, while a blanket introduction of a *safeguard* is not cost effective, it might be convenient to introduce it adaptively, only when the traffic shows specific characteristics. Furthermore, we adopted a single size, namely 10%, for the *safeguard*, while different sizes may have different

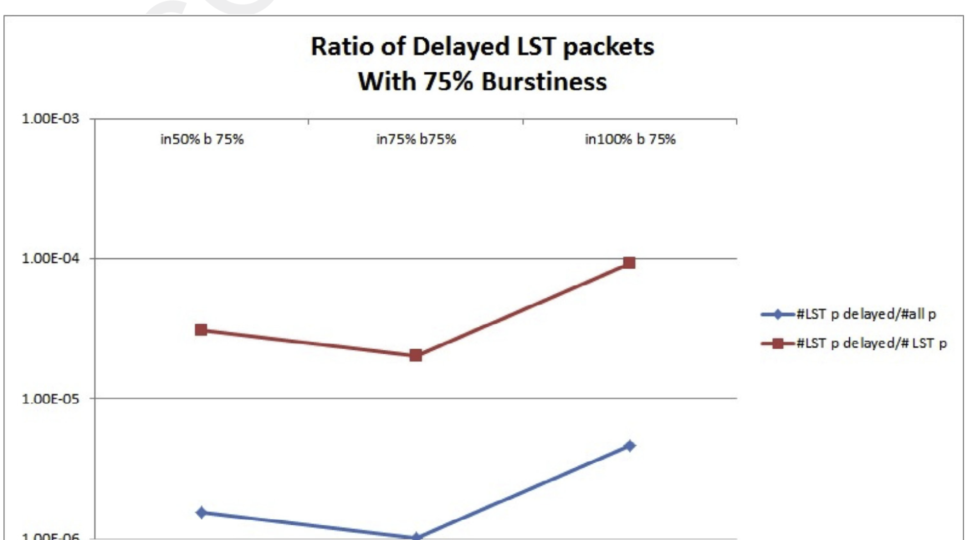


Fig. 10 – The ratio between LST packets that incur into a delay and all packets or LST packets. Safeguard set to 10%. Burstiness is 75%.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

Table 2 – Energy savings with 50% level of burstiness.

–	No safeguard	Safeguard	Safeguard/ no safeguard
in50%	2.37192	0.83397	2.84414
in75%	1.62984	0.09240	17.63902
in100%	0.15419	0.00021	734.56149

Table 3 – Energy savings with 75% level of burstiness.

–	No safeguard	Safeguard	Safeguard/ no safeguard
in50%	2.33335	0.90910	2.56667
in75%	1.68678	0.09632	17.51162
in100%	0.13384	0.00021	637.53426

Table 4 – Advantages and disadvantages of the *safeguard*.

Burstiness 50%	Energy cost increase	Packet delay reduction
in50%	2.84	1.02
in75%	17.64	1.02
in100%	734.56	1.52
Burstiness 75%	Energy cost increase	Packet delay reduction
in50%	2.57	1.02
in75%	17.51	5.09
in100%	637.53	1.02

characteristics. In future work we will analyze in further details both the most convenient traffic characteristics and the most effective *safeguard* size. Finally, we analyzed the size of delays in the worst case scenario, i.e., nominal load, maximum burstiness and no safeguard. As Fig. 11 shows, the size of the delays imposed to packets is extremely limited. In fact, even if LIT packets are almost always delayed, they are never delayed by more than two time slices; furthermore, LST packets are very seldom delayed and never by more than a single time slice.

5. Conclusions and future work

The need for a full sanitization of Internet traffic is becoming more critical everyday as more types of potentially vulnerable devices get connected. In past works it has been proven that it is possible to leverage an aggressive, distributed intrusion detection to achieve both full traffic sanitization and a reduction of the energy costs of networking, however, the impact of this methodology on the delay imposed to packets had not been fully evaluated. In this work we have introduced and evaluated a methodology that, by prioritizing the type of traffic that gets examined first, is capable of reducing the disruption that delaying packets may cause. More in details, our approach is based on the dynamic classification of traffic into latency sensitive traffic (LST) and latency insensitive traffic (LIT) at the source node. The knowledge about which class a packet belongs allows routers along its path to modulate their resources so that while LIT packets might be delayed as their security analysis has priority over their forwarding, LST packets are analyzed only if there are enough resources to also guarantee their timely forwarding. To validate our methodology we have performed

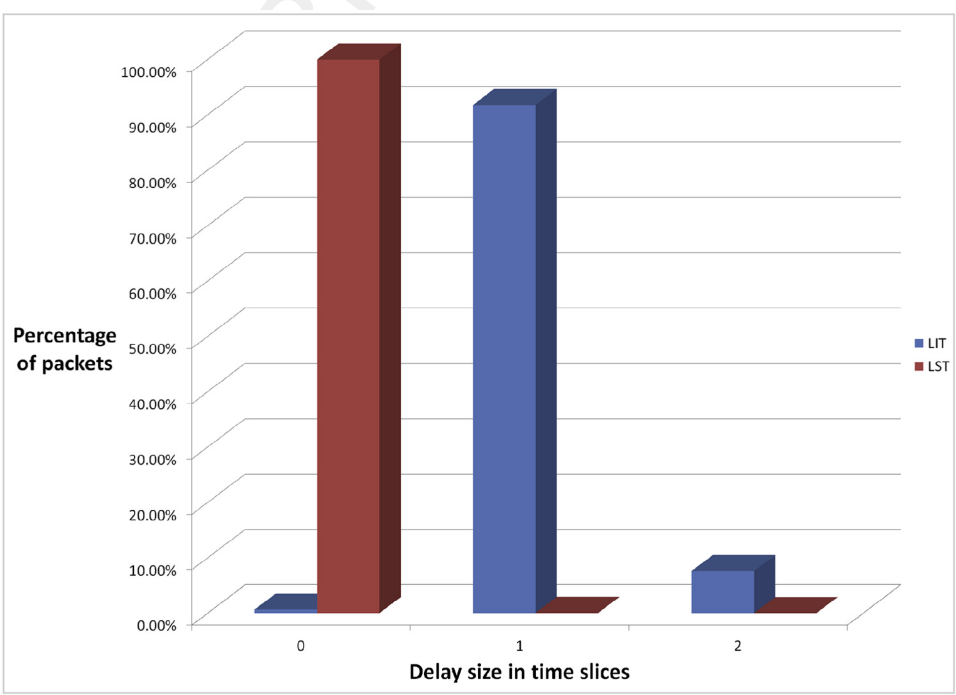


Fig. 11 – Distribution and size of delays of both LST and LIT packets in the worst case scenario (Burstiness 75%, nominal load).

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003

a simulation campaign adopting a synthetic traffic sample based on the characteristics of real world traffic captured at the University of Padua and at the Milan Internet eXchange. Our simulations prove that our methodology allows keeping the number of LST packets delayed to very low levels for network loads that are not close to the network nominal forwarding capacity. In order to get the same low number of delayed LST packets with higher network loads we have tested the introduction of a systematic underestimation of the resources available for security analysis as a *safeguard* against errors in resource availability estimation. The presence of the *safeguard* provides better results in terms of the number of delayed LST packets, yet its cost in terms of lost energy savings do not call for its general adoption.

In this work, to dynamically identify what traffic flows are sensitive to latency, we focused on a subset of commonly-used symmetric VOIP traffic classes; in future work, we plan to include additional categories of real-time LST traffic classes such as tele-control and Internet gaming. Furthermore, given the mixed results that the adoption of the *safeguard* provided in our simulations, we also plan to study how to dynamically adapt the size of the *safeguard* according to the QoS requirements of the LST traffic, the size of the traffic entering the network and the level of burstiness of the traffic itself. Finally, we also plan to study how to improve our approach by offloading the process of tagging packets to a co-processor unit so that packet tagging and analysis can happen simultaneously.

REFERENCES

Aburomman AA, Reaz MBI. A survey of intrusion detection systems based on ensemble and hybrid classifiers. Comput Secur 2017;65(Suppl. C):135–52. doi:10.1016/j.cose.2016.11.004.

Al Haj Baddar S, Merlo A, Migliardi M, Palmieri F. Dynamic latency sensitivity recognition: an application to energy saving. In: Green, pervasive, and cloud computing: 12th international conference, GPC 2017, Cetara, Italy, May 11–14, 2017, proceedings. Springer International Publishing; 2017. p. 138–51.

Ashfaq RAR, Wang X-Z, Huang JZ, Abbas H, He Y-L. Fuzziness based semi-supervised learning approach for intrusion detection system. Inf Sci (Ny) 2017;378:484–97. doi:10.1016/j.ins.2016.04.019.

Baddar S, Merlo A, Migliardi M. Anomaly detection in computer networks: a state-of-the-art review. J Wireless Mobile Netw Ubiquitous Comput Depend Appl 2014;5(4):29–64.

Baddar SWA, Merlo A, Migliardi M. Generating statistical insights into network behavior using SKETURE. J High Speed Netw 2016;22(1):65–76. doi:10.3233/JHS-160539.

Chabarek J, Sommers J, Barford P, Estan C, Tsiang D, Wright S. Power awareness in network design and routing. In: INFOCOM 2008. The 27th conference on computer communications. IEEE; 2008. p. 457–65. doi:10.1109/INFOCOM.2008.93.

Desale KS, Kumathekar CN, Chavan AP. Efficient intrusion detection system using stream data mining classification technique. In: 2015 international conference on computing communication control and automation. 2015. p. 469–73. doi:10.1109/ICCUBEA.2015.98.

G. Group. Gartner says 8.4 billion connected things will be in use in 2017, up 31 percent from 2016. Tech Rep. 2017. Available from: https://www.gartner.com/newsroom/id/3598917.

Hassanzadeh A, Altaweel A, Stoleru R. Traffic-and-resource-aware intrusion detection in wireless mesh networks. Ad Hoc Netw 2014;21:18–41. doi:10.1016/j.adhoc.2014.04.009.

Ji S-Y, Jeong B-K, Choi S, Jeong DH. A multi-level intrusion detection method for abnormal network behaviors. J Netw Comput Appl 2016;62:9–17. doi:10.1016/j.jnca.2015.12.004.

Khan ZA, Herrmann P. A trust based distributed intrusion detection mechanism for internet of things. In: 2017 IEEE 31st international conference on advanced information networking and applications (AINA). 2017. p. 1169–76. doi:10.1109/AINA.2017.161.

Labs M. McAfee labs threats report. Tech Rep. 2016. Available from: https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-dec-2016.pdf.

Lan K-C, Hussain A, Dutta D. Effect of malicious traffic on the network. In: Passive and active measurement workshop (PAM). 2003.

Leder F, Werner T, Martini P. Proactive botnet countermeasures – an offensive approach. In: In cooperative cyber defence centre of excellence. 2009.

Li Y, Wang J-L, Tian Z-H, Lu T-B, Young C. Building lightweight intrusion detection system using wrapper-based feature selection mechanisms. Comput Secur 2009;28(6):466–75. doi:10.1016/j.cose.2009.01.001.

Lin W-C, Ke S-W, Tsai C-F. CANN: an intrusion detection system based on combining cluster centers and nearest neighbors. Knowl Based Syst 2015;78:13–21. doi:10.1016/j.knosys.2015.01.009.

Mallikarjunan KN, Muthupriya K, Shalinie SM. A survey of distributed denial of service attack. In: 2016 10th international conference on intelligent systems and control (ISCO). 2016. p. 1–6. doi:10.1109/ISCO.2016.7727096.

Merlo A, Spadacini E, Migliardi M. IPS-based reduction of network energy consumption. Logic J IGPL 2016;24(6):982. doi:10.1093/jigpal/jzw053.

Migliardi M, Merlo A. Modeling the energy consumption of distributed ids: a step towards green security. In: MIPRO, 2011 proceedings of the 34th international convention. IEEE; 2011. p. 1452–7.

Migliardi M, Merlo A. Energy consumption simulation of different distributed intrusion detection approaches. In: 2013 27th international conference on advanced information networking and applications workshops (WAINA). 2013a. p. 1547–52. doi:10.1109/WAINA.2013.214.

Migliardi M, Merlo A. Improving energy efficiency in distributed intrusion detection systems. J High Speed Netw 2013b;19(3):251–64.

Muradore R, Quaglia D. Energy-efficient intrusion detection and mitigation for networked control systems security. IEEE Trans Industr Inform 2015;11(3):830–40. doi:10.1109/TII.2015.2425142.

Noorbehbahani F, Fanian A, Mousavi R, Hasannejad H. An incremental intrusion detection system using a new semi-supervised stream classification method. Int J Commun Syst 2017;30(4):e3002. doi:10.1002/dac.3002. e3002 IJCS-15-0106.R1.

Paganini P Botnet around us: are we nodes of the matrix. The Hacker News August.

Ricciardi S, Careglio D, Fiore U, Palmieri F, Santos-Boada G, Solé-Pareta J. Analyzing local strategies for energy-efficient networking. In: ICC 2011, networking workshops. Springer; 2011. p. 291–300.

Ricciardi S, Careglio D, Santos-Boada G, Solé-Pareta J, Fiore U, Palmieri F. Towards an energy-aware internet: modeling a cross-layer optimization approach. Telecommun Syst 2013;1–22.

Sedjelmaci H, Senouci SM, Al-Bahri M. A lightweight anomaly detection technique for low-resource IoT devices: a game-theoretic methodology. In: 2016 IEEE international conference on communications (ICC). 2016. p. 1–6. doi:10.1109/ICC.2016.7510811.

Şen S, Clark JA, Tapiador JE. Power-aware intrusion detection in mobile ad hoc networks. Berlin Heidelberg: Springer; 2010. p. 224–39.

Tsikoudis N, Papadogiannakis A, Markatos EP. LEoNIDS: a low-latency and energy-efficient network-level intrusion detection system. IEEE Trans Emerg Topics Comput 2016;4(1):142–55. doi:10.1109/TETC.2014.2369958.

Viegas E, Santin AO, França A, Jasinski R, Pedroni VA, Oliveira LS. Towards an energy-efficient anomaly-based intrusion detection engine for embedded systems. IEEE Trans Comput 2017;66(1):163–77. doi:10.1109/TC.2016.2560839.

Wang W, Guyet T, Quiniou R, Cordier M-O, Masseglia F, Zhang X. Autonomic intrusion detection: adaptively detecting anomalies over unlabeled audit data streams in computer networks. Knowl Based Syst 2014;70:103–17. doi:10.1016/j.knosys.2014.06.018.

Weller-Fahy DJ, Borghetti BJ, Sodemann AA. A survey of distance and similarity measures used within network intrusion anomaly detection. IEEE Commun Surv Tutor 2015;17(1):70–91. doi:10.1109/COMST.2014.2336610.

Zhu H, Du S, Gao Z, Dong M, Cao Z. A probabilistic misbehavior detection scheme toward efficient trust establishment in delay-tolerant networks. IEEE Trans Parallel Distrib Syst 2014;25(1):22–32. doi:10.1109/TPDS.2013.36.

Please cite this article in press as: Sherenaz Al-Haj Baddar, Alessio Merlo, Mauro Migliardi, Francesco Palmieri, Saving energy in aggressive intrusion detection through dynamic latency sensitivity recognition, Computers & Security (2017), doi: 10.1016/j.cose.2017.12.003