Duplicate detection methodology
for IP network traffic analysis

Iñaki Ucar, Daniel Morato, Eduardo Magaña and Mikel Izal

Dept. of Automatics and Computer Science

Public University of Navarre

Campus Arrosadia, 31006 Pamplona, Spain

{i.ucar,daniel.morato,eduardo.magana,mikel.izal}@unavarra.es

Abstract—Network traffic monitoring systems have to deal with a challenging problem: the traffic capturing process almost invariably produces duplicate packets. In spite of this, and in contrast with other fields, there is no scientific literature addressing it. This paper establishes the theoretical background concerning data duplication in network traffic analysis: generating mechanisms, types of duplicates and their characteristics are described. On this basis, a duplicate detection and removal methodology is proposed. Moreover, an analytical and experimental study is presented, whose results provide a dimensioning rule for this methodology.

Index Terms—Network analysis, network monitoring, traffic capturing, packet duplication.

I. INTRODUCTION

Data duplication is a generic problem in the broad field of data engineering and information systems. There is a lot of affected disciplines that have worked very hard on this topic over the years like, for example, the fields of statistics, databases or artificial intelligence [1], [2]. In network traffic monitoring and analysis, data duplication appears as duplicate packets.

Most techniques belonging to this discipline rely on a first step of capturing network packets. There are several ways to capture traffic from Ethernet-based packet-switched networks, but the best suited to meet today's requirements is called port mirroring (*Switched Port Analyzer* or SPAN in Cisco Systems nomenclature). Network hardware, like switches and routers, usually implements this functionality. These devices have the ability to monitor multiple high speed links and send a copy of every packet to a mirror port that can be used to capture traffic.

Manufacturers like Cisco or Hewlett-Packard also allow monitoring Virtual LANs (VLANs) in their equipment. This feature makes monitoring easier and, therefore, it is the most common practice. This mechanism is simply a shortcut to
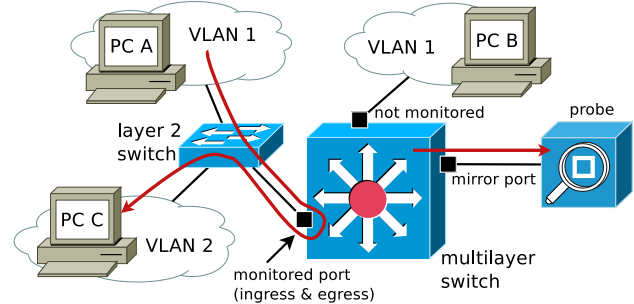
Fig. 1. A very simple port mirroring scheme.

monitor all the ports that belong to a particular VLAN in a single command. Therefore, hereafter we will use *monitored port* to refer indistinctly to both cases, port-based SPAN or VLAN-based SPAN, because in fact they represent the same problem.

Fig. 1 shows a very simple port mirroring scheme in which a single port is being monitored. PC A in VLAN 1 sends IP packets to PC C in VLAN 2. These packets are routed at the multilayer switch. At the mirror port of the switch, we obtain both the ingress and egress copies from every packet belonging to this stream, which constitute duplicate packets. It is obvious that monitoring only the ingress traffic to this port avoids data duplication, but in such a case, a packet coming from PC B to PC A or C would not be captured.

It's important to note that not all the traffic is duplicated, but only those packets coming to a monitored port and leaving from another monitored port (or the same one). This is the essence of duplicate packets. In Fig. 1, the traffic between PCs A-C will be duplicated, but the traffic between PCs B-C and A-B will not.

This simple example shows that, in a practical way, duplicate packets are unavoidable. Every packet traversing two monitored links will appear twice in the network trace. As a result, those duplicates constitute a noise signal that may mislead subsequent analysis in two main ways:

- **Perverting the volume of information**. Packet duplication implies throughput duplication, which impacts on Service Level Agreement (SLA) planning and threshold-based alerting. Moreover, this throughput duplication does not occur in a homogeneous fashion: as has al-
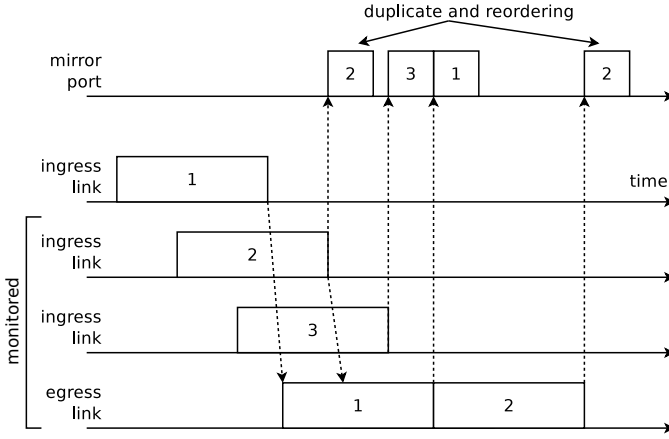
Fig. 2. Three packets crossing a device with some monitored ports. The horizontal axis represents time. The mirror port is supposed to be a faster port.

TABLE I
PACKET MODIFICATIONS FOR DIFFERENT TYPES OF DUPLICATES

Type	Layer	Change	May change
Switching	2		Trunking encap.
	3		DSCP value
			Checksum
Routing	2	Source address	Trunking encap.
		Destination address	
	3	TTL	DSCP value
		Checksum	Options
NAT routing	2	Source address	Trunking encap.
		Destination address	
	3	TTL	DSCP value
		Checksum	Options
			Source address
			Destination address
	4	TCP/UDP checksum	Source port
			Destination port
Proxying	2	Source address	Trunking encap.
		Destination address	
	3	Checksum	DSCP value
			Source address
			Destination address
	4	TCP/UDP checksum	Sequence number
			ACK number

ready been mentioned, some traffic may be affected and other may not. This point turns into estimation errors in traffic characterization through the traffic matrix, the identification of heavy hitters or the analysis of packet size distributions, protocol/application mix, etc.

- **Complicating the tracking of stateful connections**. For example, duplicate sequence numbers within TCP connections can be mistaken for valid TCP retransmissions.

Nevertheless, in spite of its importance, too little attention has been paid to this packet duplication problem. This work is intended to fill this gap. The main contributions of this paper are the analysis of the problem, with a description and classification of the different types of duplicates; the development of a methodology for network packet duplicate detection and removal, and an analytical and experimental dimensioning of this methodology.

The remainder of the paper is organized as follows. Section II describes the problem of duplicate packets in network traffic monitoring. We discuss several mechanisms that produce distinct types of duplicates. Section III presents a methodology for off-line detection of duplicate packets in order to avoid them in subsequent analysis. Section IV discusses efficiency aspects. We present some analytical and experimental results and their implications on the fine tuning of our methodology. A dimensioning rule regarding our approach is provided. Finally, Section V summarizes the conclusions of this paper.

II. DUPLICATES IN NETWORK TRAFFIC MONITORING

The analysis presented in this section is referred but not limited to a switched Ethernet environment, and the layer 3 is supposed to be IPv4. The IPv6 case is analogous, with the caveat that the IP identification field disappears.

Fig. 2 shows a schematic of three packets crossing a network device (switch, router...) with some monitored ports:

- Packet 1 comes into a not monitored port and goes to a monitored one.

- Packet 2 comes into and leaves from a monitored port, therefore it becomes a duplicate at the mirror port.
- Packet 3 comes into a monitored port and goes to a not monitored one (not represented).

It is assumed that the packets are copied to the mirror after being received or transmitted. Those are the three possible behaviours at any monitored device, and only the second one produces duplicates. Determining which streams will be duplicated becomes challenging in general. Those duplicates consist of an ingress and an egress copy of the same packet. As can be seen in Fig. 2, there is a time lag between copies as a consequence of the switching time and the queueing delay, and a variable number of other packets may fall between them.

Moreover, the egress copy can undergo different switching mechanisms that cause distinct types of duplicates. While packet payloads in general remain unchanged, the switching processes (i.e., switching at layer 2, routing, etc.) could lead to several changes in packet headers at various levels of the protocol stack, even when switching at layer 2.

In summary, it is not enough to look for identical packets like some approaches do [3]. On the whole, several fields will change, others might or might not change, and others remain the same, like the packet payload. We are going to classify the duplicates depending on the generating mechanism. Table I summarizes the expected changes at different layers.

A. Switching duplicates

They are generated by a switching process at layer 2: the packets enter and leave the device within the same VLAN.

Even so, there are some header fields that could change. For example, the egress port could be a trunking port while the ingress not, which implies that egress packets incorporate a 802.1Q VLAN tag at the link layer. In the same manner, the device could be applying QoS policies and thereby the outgoing packets could have different 802.1p priority at Ethernet level or different DSCP value at IP level (wherewith the IP checksum also changes). Note that these changes due to classification and marking could happen for all types of duplicates.

These duplicates can represent up to 50 % of the traffic captured (even more in particular conditions of packet flooding). They distort MAC-to-MAC stream analysis, hence all the higher layers will be affected. Some equipment, like HP ProCurve switches, are able to automatically suppress these duplicates, but only when the ingress and the egress copies are identical.

B. Routing duplicates

They are generated by a switching process at layer 3 (or routing). This was the example shown in Fig. 1. In this process, some alterations must occur:

- The source MAC address is replaced with the device address.
- The destination MAC address is replaced with the next hop address.
- The Time To Live (TTL) value is decreased by one.
- The IP checksum is recalculated.
- The IP options may change.

C. NAT Routing duplicates

If the router from the previous case is also working as a Network Address Translation (NAT) device, additional changes are expected to occur [4]:

- The source IP address is replaced with the external address if the packet is leaving the private network. In the opposite direction, the destination IP address is replaced with the internal address.
- Both IP addresses are included in the pseudo-header used for IP and TCP/UDP checksum calculation. Therefore, they need to be recalculated.

This behaviour constitutes a basic NAT, but it can be extended with port translation:

- The source TCP/UDP port is replaced with a mapped port if the packet is leaving the private network. In the opposite direction, the destination TCP/UDP port is replaced with the original port.

D. Transparent proxying duplicates

Transparent proxies and reverse proxies (load balancers) are not uncommon. These devices serve as a basic NAT with transport layer rewriting ability, a technique called *delayed binding* or *TCP splicing* [5]:

- The TCP sequence number is rewritten if the packet is travelling from client to server. In the opposite direction, the TCP ACK number is rewritten.
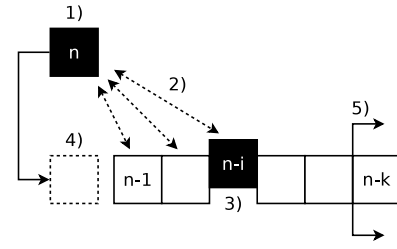

Fig. 3. Sliding window for duplicate detection.

E. Other rare cases

Today's network infrastructures are dotted with heterogeneous devices that may give rise to unexpected behaviours. Many of these behaviours may respond to, or be consequence of, security reasons. For example, a NAT device effectively isolates a private network from incoming connections, but it also malfunctions with special protocols that carry connection information (IP, port) at application level, such as File Transfer Protocol (FTP) in active mode. Therefore, some of these NATs work as Application Layer Gateways (ALGs) and are able to rewrite the payload [6].

Some of these rare cases are unusual and require special treatment. In the present state, our methodology avoids them.

III. DUPLICATE DETECTION METHODOLOGY

Duplicate packets are not in general identical. Although choosing to compare only payloads might be a good first approach (note, however, that there will be many packets without payload), knowing the type of duplicates is very valuable in order to understand the traffic patterns. This information can only be obtained from packet headers. Indeed, most analysis may require full deduplication, but some of them may not:

- If we are calculating the utilization factor per VLAN, switching duplicates must be removed, but routing duplicates must not.
- Transport level statistics require removing the routing, routing NAT and proxying duplicates too.
- Preserving the routing NAT and proxying duplicates allows us to study both sides of the NAT or proxy.

Our methodology works off-line on previously saved captures. It relies on a simple sliding-window model. Every packet is compared against the packets in a windowed buffer using policies derived from the previous section. There is a prototype of this methodology coded in C available at Github [7].

A. The sliding window

Fig. 3 shows the sliding window behaviour. The steps are described below:

1) With a window of size k, the n-th packet is read from the trace file.
2) The n-th packet is compared against every packet in the window, from $n-1$ to $n-k$.
3) If a match is found at the i-th comparison, the n-th packet is marked as a duplicate from the $n-i$. The search through the window stops.

4) The n-th packet is added to the window.
5) The window is trimmed to fit a maximum size (older packets are removed).

B. Packet comparison process

Every packet is dissected in order to determine the highest layer payload that remains unchanged according to our analysis (other protocols might be examined and added):

- The TCP/UDP payload.
- The IP payload for non-TCP/UDP packets.
- The Ethernet payload for non-IP packets.

This payload is compared first. If it matches, the header fields are used in order to confirm the identification and decide the type of duplicate. Section II classifies the distinct types of duplicates and analyzes the expected changes. The rules below must be followed:

- **Fields that do not change**. All of them must be compared.
- **Fields that change**. TTL and checksums are not compared. Source and destination MACs must be compared to ensure that they change.
- **Fields that may change**. Trunking encapsulation, DSCP value and options are not compared. The pairs source/destination IP, source/destination port and ACK/sequence number are compared to ensure that only one changes and the other remains the same.

Each type of duplicate has a specific comparator that implements these policies.

IV. EFFICIENCY ASPECTS

A. Efficiency of a single comparison

According to the previous section, there is a set of fields in a packet that are essential to resolve a duplicate pair. On the other hand, it may be obvious that a pair does not match using a small subset of fields. Therefore, it is important to determine the best order of comparison to avoid wasting valuable time.

The payload constitutes the most significant difference between non-duplicate packets. In order to prove this assumption, we collected a one-day capture at our university's outgoing link. Starting from the deduplicated trace (of around $2 \cdot 10^9$ packets), every packet payload was compared against the entire window using the described methodology. For each pair, we collected how many bytes were compared until the mismatch was found.

Four window sizes were tested: 0.1 ms ($\sim 9 \cdot 10^9$ comparisons), 1 ms ($\sim 6 \cdot 10^{10}$ comparisons), 10 ms ($\sim 5 \cdot 10^{11}$ comparisons) and 100 ms ($\sim 5 \cdot 10^{12}$ comparisons). The probability survival curve in Fig. 4 shows that, at worst, about 50 % of non-duplicate pairs were discarded before entering the payload comparison (because protocols or payload sizes didn't match or were equal to zero), and the following 49 % was discarded within the first two bytes.

This finding states that comparing the payload in the first place, before the header fields, is more efficient in order to discard non-duplicate pairs.
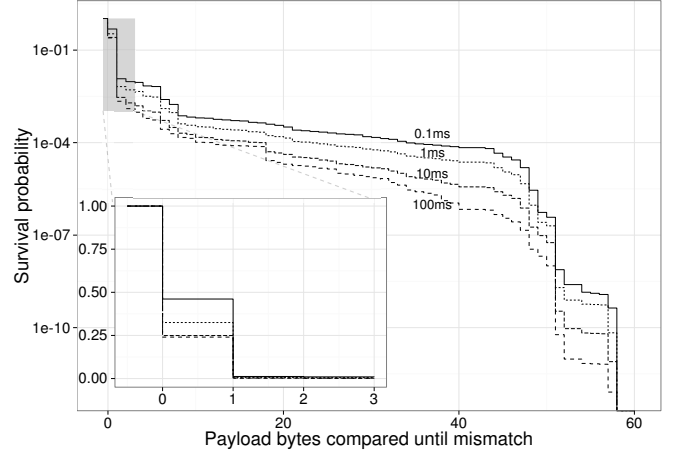

Fig. 4. Survival curves for the byte-by-byte payload comparison process with a deduplicated capture. Four window sizes were considered.

B. Reducing the number of comparisons

In the worst case, a non-duplicate packet is compared against the entire window. Therefore, using the smallest possible window without compromising the quality of the detection process is desirable because the algorithm complexity scales linearly with this size. This size can be established in terms of time (time-sliding window) or number of packets (element-based sliding window). In order to decide the window size, we are interested in the maximum distance between duplicates.

Previous sections established that a pair of duplicates is formed by an ingress and an egress copy of the same packet. We are going to assume that the switch internally behaves as follows:

1) One packet arrives at a monitored port.
2) The mirror port gets the ingress copy.
3) The packet is switched to an output queue of another (or the same) monitored port.
4) After the queue, the mirror port gets the egress copy.
5) The packet is transmitted.

This behaviour is going to be analytically modeled and experimentally tested.

Between both copies, there is a time spent in the system s_n that can be decomposed into the switching time (or decision time) x_n and the queueing time (or waiting time in the output queue) w_n. Applying the expectation operator:

$$\bar{s} = \bar{x} + \bar{w} \qquad (1)$$

The time spent in the system can be measured as the time difference between the ingress and egress copy of a packet at the mirror port. Within this lag, several copies of other packets might be sent to the mirror port and fall between them, as seen in Fig. 2. In our model, the packet difference between duplicates, Δn, is expected to be proportional to this system time:

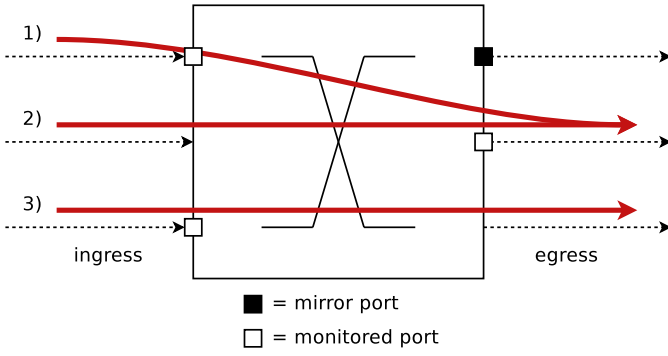$$\bar{\Delta n} = \sum \mu_i \bar{s} \qquad (2)$$

Fig. 5. Experimental setup.

Rate (pps)	Rate (Mbps)
5000	3.36
25000	16.80
45000	30.24
65000	43.68
85000	57.12
105000	70.56
125000	84.00

where μ_i is the average service rate at the i-th monitored port, what we shall call interfering traffic (i.e., packets 1 and 3 in Fig. 2).

This simple model predicts a device-dependent upper bound in terms of time. On the other hand, the number of packets falling between duplicates grows linearly with the device load. In order to test this hypothesis, we conducted an experimental study making use of our implementation [7]. Fig. 5 sketches the experimental setup. Three streams are defined:

1) *Main stream*: it is monitored at both the incoming and the outgoing link, so it produces duplicate packets.
2) *Auxiliary stream*: it is monitored at the outgoing link in the same port that the main stream. It allows us to force queueing at this port.
3) *Interfering stream*: it is monitored once at the incoming link and lets us insert packets between duplicates at different rates.

Our testbed comprises the following hardware:

- One Cisco Catalyst 3560 Switch with 8 FastEthernet ports and 1 Gigabit port (used as mirror port).
- Two Dell PowerEdge T110 servers with Intel Xeon X3460 2.8 GHz CPU, 4 GB DDR3 RAM and Intel Gigabit E1G44ET Quad Port PCIe card.

The aim of this experiment is to evaluate the impact of the interfering stream into the maximum distance between duplicates. Main and auxiliary streams are formed by maximum size Ethernet packets (PDU + headers + MAC preamble + frame delimiter + CRC + inter-frame gap = 1538 octets) in order to maximize the processing time. The interfering stream is formed by minimum size packets (84 octets), allowing us to sweep a longer range of packets per second. All these streams were genered with exponential interarrival times, making use of the software D-ITG [8].

If no losses occur at the output queue, it is equivalent to an infinite queue. Furthermore, if we suppose approximately Poisson arrivals and a deterministic service time, the output queue can be described as a M/D/1. Based on this premise, we can calculate the average queue length [9]:

$$\bar{N}_q = \frac{\rho^2}{2(1-\rho)} \qquad (3)$$

and the system time [9]:

$$\bar{s} = \frac{\delta}{2}\left(\frac{2-\rho}{1-\rho}\right) \qquad (4)$$

where ρ is the utilization factor and δ is the service time. The default maximum length of the output queue for our Cisco 3560 switch is 40 packets. In order to avoid losses, we target an average queue length of 5 packets. Thus, Equation (3) with $\bar{N}_q = 5$ gives a utilization factor $\rho \approx 0.916$, or 91.6 of 100 Mbps (45.8 Mbps, or 3722 pps, per stream). A greater utilization factor increases the system time. Therefore, we are working with a moderately bad case.

The deterministic service time can be obtained as follows:

$$\delta = \frac{L}{C} \qquad (5)$$

where L is the length of packets for the main stream ($L = 1538 \cdot 8$ bits) and C is the link capacity ($C = 100$ Mbps). With these values, Equation (4) predicts an average system time $\bar{s} \approx 0.79$ ms.

Different interfering stream rates were used, as shown in Table II. For each case, a traffic capture was collected and analyzed so as to extract time differences and number of packets between packet pairs comprising duplicates. Fig. 6 compares the measured average time difference with the theoretical value. As expected, our model predicts the mean time spent in the system. It is constant and independent from the interfering stream rate. Fig. 7 shows the linear growth predicted in the average number of packets falling between duplicates as the interfering stream rate increases.

The number of packets falling between duplicates depends as much on the system time as on the packet arrival rate of the interfering traffic. Meanwhile, the time difference depends only on the system time. As a result, a time-sliding window becomes the best strategy.

Furthermore, it can be assumed that the switching time is negligible as compared to the queueing time ($x_n << w_n$). This fact suggests that an upper limit for the system time could be established as follows:

$$max(s_n) \approx max(w_n) = \frac{max(N_q) \cdot max(M)}{C} \qquad (6)$$

where $max(N_q)$ is the maximum length of the output queue, $max(M)$ is the maximum length of a packet in that port and
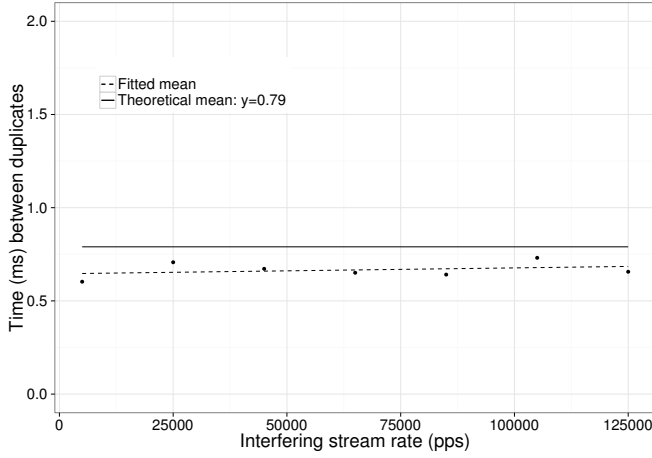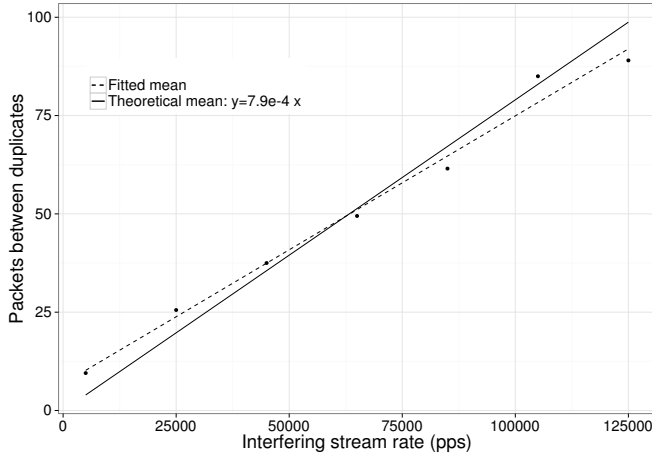
Fig. 6. Average time difference between duplicates.

Fig. 7. Average number of packets between duplicates.

C is the link capacity. With our testbed, $max(N_q) = 40$, $max(M) = 1538 \cdot 8$ bits, $C = 100$ Mbps and $max(s_n) \approx 5$ ms.

On the other hand, this study does not take into account the potential effect of a congested mirror port. The reason is that this scenario would have a harmful effect into a monitoring system because of losses, and therefore it needs to be avoided. In our experiments, the mirror port deals with an utilization factor of about 0.2 (200/1000 Mbps), which means that there can be an additional queueing delay.

In summary, the time difference between duplicates involves three contributions: the switching time x_n, the queueing time at output port w_n and the queueing time at mirror port w'_n. Being very cautious and assuming that $x_n \le w_n$ and $w'_n \approx w_n$, Equation (6) can be modified to infer a time-based upper limit for the window size (WS):

$$WS = \frac{3 \cdot max(N_q) \cdot max(M)}{min(C)} \quad (7)$$

where $max(N_q)$ is the maximum length of the largest queue,

$max(M)$ is the maximum length of a packet and $min(C)$ is the slowest link capacity. In our case, a window size of $WS = 15$ ms would be advisable. In fact, the maximum separation observed was 8.3 ms.

C. Towards an on-line detection methodology

This work proposes an off-line methodology. However, many monitoring tools perform on-line traffic analysis without saving network packets to disk. Moreover, one of the most harmful effects of duplicate packets is that they consume a lot of bandwidth at the mirror port. Accordingly, it seems clear that it would be desirable to include an on-line methodology in network devices. Unfortunately, making searches over a sliding window are a very heavy task. Thus, even with parallelization, on-line detection becomes challenging.

Nevertheless, as mentioned in a previous section, duplicates emerge because of a fixed configuration and, therefore, they occur in a deterministic way. It should be possible to find an algorithm that learns how duplicates are generated. After the learning phase, this algorithm could be able to remove duplicates without further analysis. In this way, an on-line detection methodology would be feasible and needs to be further investigated.

V. CONCLUSIONS

This paper addresses an important and unattended problem concerning network traffic monitoring activities. The theoretical background has been exposed: generating mechanisms, types of duplicates and their characteristics are described. Within this context, a detection methodology is proposed, which shall serve as reference for future works.

Moreover, an analytical and experimental study has been conducted regarding the fine tuning of this methodology. As a result, the use of a time-sliding window is recommended. The dimensioning rule provided slightly overestimates the maximum distance between duplicates. Thus, further research with other equipment is needed in order to refine this result.

REFERENCES

[1] A. Elmagarmid, P. Ipeirotis, and V. Verykios, "Duplicate record detection: A survey," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 1, pp. 1–16, 2007. 1

[2] G. Costa, A. Cuzzocrea, G. Manco, and R. Ortale, "Data De-duplication: A Review," in *Learning Structure and Schemas from Documents*, ser. Studies in Computational Intelligence, M. Biba and F. Xhafa, Eds. Springer Berlin Heidelberg, 2011, vol. 375, pp. 385–412. 1

[3] (2013, Apr.) Editcap man page. Wireshark. [Online]. Available: http://www.wireshark.org/docs/man-pages/editcap.html 2

[4] "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc2663.txt 3

[5] M. Syme and P. Goldie, *Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing*. Prentice Hall, 2003. [Online]. Available: http://www.amazon.com/Optimizing-Network-Performance-Content-Switching/dp/0131014684 3

[6] "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt 3

[7] (2013, Aug.) The infodups tool for duplicate detection at Github. [Online]. Available: https://github.com/Enchufa2/nantools 3, 5

[8] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.02.019 5

[9] L. Kleinrock, *Queueing Systems: Theory, Volume 1*. Wiley, 1976. 5