J Netw Syst Manage
DOI 10.1007/s10922-013-9278-0

Measuring Cloud Service Health Using NetFlow/IPFIX: The WikiLeaks Case

Idilio Drago · Rick Hofstede · Ramin Sadre · Anna Sperotto · Aiko Pras

Received: 18 March 2012 / Revised: 11 June 2013 / Accepted: 20 June 2013

Abstract The increasing trend of outsourcing services to cloud providers is changing the way computing power is delivered to enterprises and end users. Although cloud services offer several advantages, they also make cloud consumers strongly dependent on providers. Hence, consumers have a vital interest to be immediately informed about any problems in their services. This paper aims at a first step toward a network-based approach to monitor cloud services. We focus on severe problems that affect most services, such as outages or extreme server overload, and propose a method to monitor these problems that relies solely on the traffic exchanged between users and cloud providers. Our proposal is entirely based on NetFlow/IPFIX data and, therefore, explicitly targets high-speed networks. By combining a methodology to reassemble and classify flow records with stochastic estimations, our proposal has the distinct characteristic of being applicable to both sampled and non-sampled data. We validate our proposal and show its applicability using data collected at both the University of Twente and an international backbone during the *WikiLeaks Cablegate*. Our results show that, in contrast to *Anonymous*' claims, the users of the targeted services have been only marginally affected by the attacks.

I. Drago (✉) · R. Hofstede · A. Sperotto · A. Pras
University of Twente, Enschede, The Netherlands
e-mail: i.drago@utwente.nl

R. Hofstede
e-mail: r.j.hofstede@utwente.nl

A. Sperotto
e-mail: a.sperotto@utwente.nl

A. Pras
e-mail: a.pras@utwente.nl

R. Sadre
Aalborg University, Aalborg, Denmark
e-mail: rsadre@cs.aau.dk

Published online: 05 July 2013

Keywords Cloud computing · Performance · Measurements

1 Introduction

The recent advent of cloud services is changing the way computing power is delivered to enterprises and end users. Cloud services are very attractive in terms of costs, provisioning and scalability. We see indeed an increasing interest from organizations in migrating services, such as data storage or e-mail, to cloud providers [1]. However, outsourcing such essential services makes consumers extremely dependent on cloud providers. Hence, consumers have a vital interest to be immediately informed about any kind of problems with their services. Only then can they notify the provider or, depending on the problem, start to locate the cause in their own network.

This raises the question of whether remote services can be monitored without direct access to the servers. Monitoring approaches such as active probing or client instrumentation are generally application-specific, create additional load, or need modifications in client software [2, 3]. This paper aims at a first step toward a network-based approach to monitor cloud services: in many cases, the services are damaged in such a severe way that clients are not even able to communicate with the servers. Such situations can be caused, for instance, by data-center outages, loss of connectivity or extreme server overload. These cases are surprisingly common among cloud providers [4] and, therefore, are primary problems that consumers need to be aware of. This motivates the search for a general monitoring solution that can be applied to *generic cloud services*.

Ideally, a complete solution to network-based service monitoring should consider all traffic to the service as well as the application layer protocol semantics. However, such a solution requires deep packet inspection, which is already unfeasible due to both the amount of traffic in enterprise networks and the increasing use of end-to-end encryption [5]. Because of that, we investigate whether the monitoring can be achieved by analyzing only flow measurements (i.e., NetFlow [6] or IPFIX [7] data), which are typically exported at border routers of enterprise and campus networks. By doing so, we look for a network-based solution that is (1) applicable to a wide range of existing services, (2) robust enough to handle encrypted cloud traffic and, (3) scalable and easy to be deployed in existing high-speed networks.

Our research question is, therefore, *how severe problems on cloud services can be monitored using only NetFlow/IPFIX data*. For achieving that, we restrict our monitoring scope to TCP and propose a new metric to indicate the status of cloud services, named *health index*. This decision of limiting the scope is justified by (1) our explicit goal of looking for solutions that are generic, instead of application-specific; (2) the predominance of TCP as the transport protocol among current cloud services [8, 9] and; (3) the high severity of problems in our scope, which also impact traffic characteristics at the transport layer.

Our contribution is a method for measuring the health index of cloud services that relies only on flow data. The key part of our approach is its ability to estimate the status of TCP connections from flow records. Since intercepting all packets for

flow accounting is not always possible in high-speed networks, our method is designed to cope with both non-sampled and sampled data.

The proposed method is validated in two steps. First, data collected at the University of Twente are used to show that the health index calculated by our flow-based approach is equivalent to what would be obtained from full packet traces. After that, real-life cases show that the index reveals problems in cloud services. We first use non-sampled flow data collected in our network during 10 consecutive weeks to analyze performance anomalies in popular Web services. Then, packet-sampled flows collected at an international backbone during the *WikiLeaks Cablegate* [10] are evaluated to verify whether end users have been affected by those attacks.

The remainder of this paper is organized as follows. Section 2 describes the envisaged deployment scenario along with our solution to handle non-sampled and sampled data. Section 3 validates the method against a packet-based alternative. Section 4 shows performance anomalies revealed from non-sampled data. Section 5 applies the method to packet-sampled flows related to the *WikiLeaks Cablegate*. Related works are described in Sect. 6. Section 7 discusses limitations of our approach and future work. Finally, Sect. 8 concludes this paper, and Appendix 1 and 2 provide background information on flows and on the validation methodology, respectively.

2 Approach: Flow-Based Service Monitoring

Assuming that a set of enterprise routers is exporting flow records while handling the traffic from a group of end users, we aim to use these data to monitor cloud services. A deployment scenario is depicted in Fig. 1. In this example, two links between an enterprise network and the Internet are monitored by flow exporters. Such setup is already very common, thanks to the popularity of network devices with flow export capabilities. The example makes clear that a single enterprise does not have access to all traffic related to the services, but it can still observe a significant fraction, generated by its own users.

Our method for monitoring cloud services is summarized in Algorithm 1. The algorithm receives a set of flow records as input and outputs the health index,

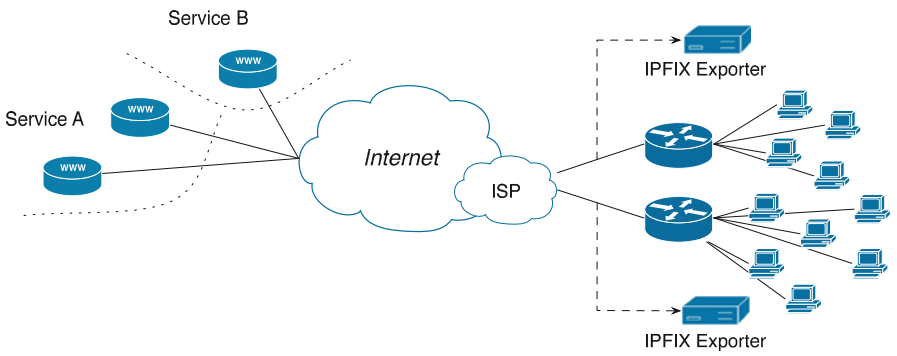

Fig. 1 Envisaged deployment scenario for our monitoring approach

defined as *the fraction of connections that are healthy* in a given time interval. A TCP connection is defined to be *healthy* if it exchanges any productive traffic at the transport layer. This implies that only the following connections are *unhealthy*:

1. Unidirectional connections. For instance, those produced when a peer sends a TCP `SYN` packet and does not obtain any reply from the remote server;
2. Connections that could never exchange payload. For instance, those produced when a peer rejects or tries to close a connection during the TCP handshake.

The health index is, therefore, correlated to the service availability. When a service is fully available, client connections are expected to be normally established, making the index to approach 1; when the service becomes unavailable, failed attempts will decrease the index. Note that this definition provides an upper bound for determining the actual service status. The health index decreases only if a service is unavailable. A high index, however, does not guarantee that a service is operating, since application layer errors are not taken into account.

Algorithm 1 Measuring the health index from flow data

Input: S, the set of flow records for a given service
Output: Current health index
1: **if** flow records are from non-sampled packets **then**
2: $C \leftarrow$ reassembled flow records
3: $n \leftarrow$ total number of TCP connections in C
4: $n_h \leftarrow$ number of healthy TCP connections in C
5: **return** $\frac{n_h}{n}$
6: **else**
7: Estimate $\hat{n}$, the total number of connections
8: Estimate $\hat{n}_h$, the number of healthy connections
9: Determine confidence intervals of n and n_h
10: **if** $\hat{n}_h > \hat{n}$ or the intervals overlap **then**
11: **return** 1
12: **else**
13: **return** $\frac{\hat{n}_h}{\hat{n}}$
14: **end if**
15: **end if**

Since flow records normally do not map directly to TCP connections (see Appendix 1 for an example), Algorithm 1 needs to remove the effects of different deployments before calculating the health index. This is done either from non-sampled or from packet-sampled records using different methods. In the former case (lines 2–5), the algorithm determines the connection state by reassembling flow records and inspecting TCP flags. In the latter case (lines 7–14), since only part of the packets are considered, the algorithm estimates the number of healthy connections directly from the records. Both methods are detailed in the coming Sects. 2.1 and 2.2. Algorithm 1 is executed periodically in an on-line deployment, processing a new batch of records at every time interval. This scheme is further discussed in Sect. 2.3.

2.1 Non-sampled Flow Data

Algorithm 1 calculates the health index from non-sampled data in two steps: (1) flow records are merged such that they approximate the original TCP connections

(line 2); (2) the fraction of healthy connections is calculated (lines 4–5). For merging records, we propose a heuristic based on [11], which is summarized in Algorithm 2. Flow records with identical keys are merged until the connection is complete. Since flow records may be unidirectional, records from both traffic directions are also aggregated. Note that the state of Algorithm 2 is saved between execution rounds, ensuring that long connections are properly reconstructed.

Algorithm 2 evaluates whether a connection is complete before updating or expiring it (lines 7 and 15). The high aggregation of packets into flow records, however, makes this decision hard. In [11], a connection is complete if it is not updated for a predefined interval or if records with TCP `FIN` flag set are observed from both end-points. This heuristic is shown to produce good results in general, but the authors also show that it does not perform well for unhealthy connections. Errors occur when client applications immediately reuse sockets after a failed connection attempt.

Algorithm 2 Reassembling flow records

Input: S, the set of flow records for a given service
Output: C, the set of connections terminating in the interval
1: *active* ← partial connections from the previous execution
2: **for each** *record* in S **do**
3: k ← unique bidirectional key of the *record*
4: **if** *active*[k] does not exist **then**
5: *active*[k] ← new connection from the *record*
6: **else**
7: **if** *active*[k] is complete **then**
8: move *active*[k] to C
9: *active*[k] ← new connection from the *record*
10: **else**
11: update *active*[k] using the *record*
12: **end if**
13: **end if**
14: **end for**
15: Move all complete connections from *active* to C
16: Save the remaining entries in *active* for the next execution

To improve these results, we extend the heuristic as follows. Besides using timeouts and `FIN` flags, all TCP flags and the start time of flow records are used to reproduce the TCP state-machine. In analogy to the TCP state-machine implemented in Bro [12], our method traces end-points using the following states: *Inactive*, `SYN` *sent*, `SYN-ACK` *sent*, *Established*, *Partial*,[1] *Closed* and *Reset*. By means of this encoding, a connection is complete if any of the following conditions are met:

1. An *inactive timeout* interval has elapsed;
2. The connection is not established yet and an *attempt timeout* interval has elapsed;
3. The connection is in a final state and either a *close timeout* interval has elapsed or a new `SYN` flow record (sharing the same key) has been observed.

[1] Only midstream traffic without any flags has been observed.

Section 2.3 will discuss possible choices for these parameters. Our method relies on the usual transitions of the TCP state-machine, but it may execute several transitions at once (line 11). This is done by inspecting each flag in the flow record independently, and checking whether this single flag would cause a transition in the TCP state-machine. For instance, when receiving a record with `SYN`, `ACK` and `RST` flags set, our method first transfers an end-point from the initial *Inactive* state to `SYN` *sent*, then to `SYN-ACK` *sent* and, finally, to the *Reset* state in a single update.

Finally, the health index is calculated by inspecting the final state of the complete connections. For example, all connections that do not reach the *Established* state are counted as unhealthy. Note that, in contrast to [11], determining the originator and responder of a connection is not a critical step, since we are interested only in a list of predefined services. Those are, therefore, known in advance to be responders.

2.2 Sampled Flow Data

Algorithm 1 calculates the health index from packet-sampled flows (lines 7–14) as follows. First, it estimates the total number of connections and the number of healthy connections (line 7 and 8). Owing to sampling, those numbers are realizations of random variables. In order to compare the quantities in a meaningful way, confidence intervals have to be determined (line 9) and taken into account: if the intervals overlap, there is not enough evidence of unhealthy connections (line 11); otherwise, the health index is calculated as in the non-sampled case (line 13).

2.2.1 Estimating the Number of Connections

The consequences of packet sampling have been described extensively in [13]. If packets are sampled independently with probability $p = 1/N$, some properties of the original data stream can be estimated by rescaling the observed quantities by a factor N. In [13], $\hat{f} = Nk_S$, where k_S is the number of observed records with `SYN` flag set, is proven to be an unbiased estimator for the number f of TCP flows, under the assumption of one single `SYN` packet per TCP flow.

We rely on a similar reasoning to estimate the number of healthy TCP connections. Given our definitions, we assume that: (1) *healthy connections have exactly one* `SYN` *packet from both originators and responders*; (2) *unhealthy connections have exactly one* `SYN` *packet from originators, but none from responders*. Let k_{orig} and k_{resp} be the number of observed flow records with `SYN` flag set from originators and responders, respectively, in a time interval. Then, $\hat{n} = Nk_{orig}$ and $\hat{n}_h = Nk_{resp}$ are unbiased estimations of the total number of connections n and of the number of healthy connections n_h in that time interval.

2.2.2 Calculating the Confidence Intervals

If a service is healthy, the expected values of $\hat{n}$ and $\hat{n}_h$ are equal. However, a system can only be considered unhealthy if the difference between $\hat{n}$ and $\hat{n}_h$ is statistically significant. We evaluate the difference between $\hat{n}$ and $\hat{n}_h$ by estimating the

probability distributions of n and n_h. A service is unhealthy if confidence intervals for a given level of significance show a low probability of n and n_h to be equal—for example when the confidence intervals do not overlap.[2]

The basic idea is that we can only observe a flow record with SYN flag set if a SYN packet has been sampled. Sampling SYN packets forms a finite sequence of Bernoulli trials with success probability $p = 1/N$. If n connections occur within a fixed time interval, the number k_{orig} of sampled records with SYN flag set from originators follows the Binomial distribution $f(k|n,p) = \binom{n}{k} p^k (1-p)^{n-k}$. If $n_h \leq n$ connections are healthy in the interval, the number k_{resp} of sampled records with SYN flag set from responders follows the Binomial distribution $B(k|n_h, p)$.

Given a series of observations $\mathbf{k_{orig}} = (k_{orig,1}, k_{orig,2}, \ldots, k_{orig,r})$ and $\mathbf{k_{resp}} = (k_{resp,1}, k_{resp,2}, \ldots, k_{resp,r})$ over r time intervals, a Bayesian method to estimate n and n_h is found in [14]. The method assumes that n and n_h are constant over r consecutive time intervals. In our context, this is a realistic approximation for highly loaded services in short time intervals. Given the success probability p and the observations $\mathbf{k_{orig}}$, the posterior distribution of n is determined by:

$$f(n|\mathbf{k_{orig}}, p) \propto (1-p)^m f_0(n) \prod_{i=1}^{r} \frac{n!}{(n-k_{orig,i})!}, \tag{1}$$

where $f_0(n)$ is a prior distribution of n. If no prior knowledge about n is available, the uniform distribution is used. The mode of $f(n|\mathbf{k_{orig}}, p)$ is a biased estimation for n (with maximum likelihood). The confidence interval for n can be calculated numerically [15], for instance, by computing $f(n|\mathbf{k_{orig}}, p)$ for $n = max(k_{orig,i}) \ldots M$ such that, for a given boundary value δ, $f(n > M|\mathbf{k_{orig}}, p) < \delta$. After that, the probabilities of neighbor values of the mode are summed up until the total probability is higher than the specified significance level. The same steps are taken to calculate the posterior distribution of n_h from the observations $\mathbf{k_{resp}}$, and, based on that, the confidence interval for n_h.

Since Eq. (1) requires the calculation of large factorials, approximations may be necessary. Depending on the values of n and p, the Normal or Poisson approximations for the Binomial distribution can be used. A practical usage example of the Normal approximation can be found, for instance, in [16].

Finally, confidence intervals resulting from Eq. (1) depend on the sampling probability p. In particular, sampling less packets implies more loss of information and wider intervals. This, in turn, makes it harder to conclude that the difference between $\hat{n}$ and $\hat{n}_h$ is significant. Section 3.3.2 illustrates this effect with a numeric example.

2.2.3 *Handling SYN Retransmissions*

Service failures can cause retransmissions of SYN packets, violating the assumptions made in Sect. 2.2.1 and resulting in biased intervals. Section 3 will show that, not

[2] Note that Algorithm 1 ignores the cases where $\hat{n}_h > \hat{n}$ by chance.

surprisingly, most retransmissions are from connection originators. In the following, we provide an estimation of n taking into account that $n = \sum_{i \geq 1} n_i$, where n_i is the number of connections with exactly i SYN packets from originators.

For a given time interval, let x_i, $i \geq 1$, be the number of records with i SYN packets. It should be noted that most NetFlow exporters do not report the number of observed SYN packets per record, although some IPFIX exporters might use the tcpSynTotalCount information element to report that. Section 3 will also discuss how $x_1, x_2, \ldots$ can be estimated directly from NetFlow records.

Given a TCP flow with s SYN packets, the number s' of SYN packets sampled for this flow follows the Binomial distribution $B(s'|s, p)$. In order to simplify the following calculations, we assume $n_i = 0$ for $i > 2$, i.e., at most one SYN retransmission per TCP flow.[3] Consequently, the probability of sampling the SYN packet of a flow with one SYN packet is p, the probability of sampling the two SYN packets from a flow with two SYN packets is p^2, and the probability of sampling one SYN packet from a flow with two SYN packets is $2p(1 - p)$.

The probability of observing x_2 flow records with two SYN packets out of n_2 original flows with two SYN packets is $B(x_2|n_2,p^2)$. The probability of observing $x_{2,1}$ flow records with only one SYN packet out of the remaining $n_2 - x_2$ flows with two SYN packets is $B(x_{2,1}|n_2 - x_2, 2p(1 - p))$. Finally, the probability of observing $x_{1,1}$ flow records with one SYN packet out of n_1 flows with one SYN packet is $B(x_{1,1}|n_1, p)$.

For small p, the above Binomial distributions can be approximated by Poisson distributions. Since $x_1 = x_{1,1} + x_{2,1}$, the conditional probability of observing x_1 flow records with one SYN packet is given by

$$f(x_1|n_1, n_2 - x_2, p) = \frac{\lambda_1^{x_1}}{x_1!} e^{-\lambda_1}, \tag{2}$$

with $\lambda_1 = n_1\, p + 2\, (n_2 - x_2)\, p\, (1 - p)$. The joint probability of observing x_1,x_2 is

$$f(x_1, x_2|n_1, n_2, p) = \frac{\lambda_1^{x_1}}{x_1!} \frac{\lambda_2^{x_2}}{x_2!} e^{-\lambda_1 - \lambda_2}, \tag{3}$$

with $\lambda_2 = n_2\, p^2$ and λ_1 as in Eq. (2). Similarly to Eq. (1), the posterior distribution of n_1 and n_2 can be estimated by

$$f(n_1, n_2|x_1, x_2, p) \propto f_0'(n_1) f_0''(n_2) \lambda_1^{x_1} \lambda_2^{x_2} e^{-\lambda_1 - \lambda_2}, \tag{4}$$

where f_0' and f_0'' are prior distributions of n_1 and n_2, respectively. Given the posterior distribution of n_1 and n_2, the posterior distribution of $n = n_1 + n_2$ can be computed numerically, and its mode $\hat{n}$ gives a maximum likelihood estimation of n. Confidence intervals are calculated again by summing up the probabilities of neighbor values of the mode until the desired significance level is reached.

It should be noted that this approach estimates $\hat{n}$ only from a single observation, in contrast to Eq. (1), which considers a window of r observations. Although using

[3] Some bias will remain when $n_i > 0$ for $i > 2$, similarly to Sect. 2.2.2.

more than one interval would yield smoother estimations, it would render the numerical computation harder, owing to the involved small probabilities.

2.3 Choice of Parameters and Impact on Latency

The timeout parameters described in Sect. 2.1 control the interval in which connections are kept open, waiting for new (non-sampled) flow records. Packet-based analyzers, such as Bro [12] or Tstat [17], have long used a similar scheme to monitor TCP traffic. Default parameters in Bro are, for instance, 5 min for the inactive timeout and 5 s for both the attempt and close timeouts. Since unhealthy connections are short by definition and, therefore, terminated by one of the last two timeouts, these parameters have a negligible impact on the latency of our method.

Timeouts of flow exporters also impact the latency of our approach because flow records are held in exporters until expiration. A flow is typically expired few seconds after its last observed packet (e.g., Cisco NetFlow default idle timeout is 15 s). That is, in practice records related to unhealthy connections are sent out fast, adding only few seconds to the overall delay of Algorithm 1.

Finally, the time between the executions of Algorithm 1 could be set to any value. However, since we assume the method will be deployed in existing high-speed networks, this parameter has to adhere to the flow export infrastructure. The delay of collectors in saving and forwarding flow records to analysis applications is the most important variable in this case. Collectors, such NFDUMP [18] or flow-tools [19], typically rotate flow logs and only pass on data periodically. Default parameters range from several seconds to a few minutes—e.g., NFDUMP uses 5 min. Therefore, the total latency of our method to report a problem will be dominated by this parameter, and is expected to be of a few minutes (maximum) in a typical deployment.

3 Health Index Validation

This section performs the first step in our validation. The goal is to check whether the health index calculated by our method solely from flow data matches what would be obtained from full packet traces. The coming Sects. 4 and 5 will show that the index reveals problems in cloud services.

Section 3.1 describes the dataset and the validation methodology. After that, the health index is validated with non-sampled flow data in Sect. 3.2 and with packet-sampled flow data in Sect. 3.3.

3.1 Dataset and Methodology

Data collected at aggregation switches in our university are used to reproduce the scenario in Fig. 1. We captured a dataset in 2011 consisting of 24 h of traffic and around 4 million TCP connections. Because our method is validated using a packet-based alternative as ground truth, we use YAF [20] to generate flow records from

Table 1 Mapping between Bro states and our definition of *health*

	Meaning	Bro	Freq. (%)	Description
Healthy	Ongoing	S1	0.9	Established, but not terminated
		OTH	0.5	Midstream traffic (no flags observed)
	Closing	S2	0.3	Established, originator attempt to close
		S3	0.4	Established, responder attempt to close
	Aborted	RSTO	20.9	Established, originator aborted
		RSTR	5.7	Established, responder aborted
	Complete	SF	57.6	Established and terminated
Unhealthy		S0	7.7	Attempt without reply
	–	SH	0.4	Attempt, followed by FIN from originator
		REJ	5.2	Rejected
		RSTOS0	0.4	Attempt, followed by RST from originator

packet traces. Next sections will, however, illustrate how our approach performs when receiving NetFlow records as well.

We have extended YAF to allow us to control which rules described IPFIX standards (see Appendix 1) are considered when expiring flow records. This is used to verify the effects of such rules when dealing with non-sampled data. To ensure that our results are applicable to NetFlow exporters, we disable functionalities that are specific to IPFIX, such as bidirectional flow export. Besides that, we have implemented a random sampler in YAF for evaluating how the method performs with packet-sampled flows. For the analyses that focus on a specific service or organization, we rely on MaxMind's GeoIP Organization database [21] and on DNS queries for isolating traffic.

In both the non-sampled and packet-sampled cases, we follow a validation methodology similar to [22]. Based on the same dataset, TCP connection summaries are generated (1) using a packet-based solution, and (2) after converting the traces into flow records. The results of both the packet-based and the flow-based approaches are then compared. As [11, 22], we use Bro [12] as the ground truth for our comparisons. Bro is a stateful network monitor, which contains a module for analyzing TCP connections. By considering complete TCP headers, Bro is able to reproduce the TCP state-machine and determine final connection states precisely. Bro encodes final states using the names listed in Table 1.[4]

We map Bro states into the two classes of our problem, as follows. *Ongoing*, *closing*, *aborted* and *complete* connections are *healthy*. Most connections marked as *ongoing* or *closing* occur because we have captured data for a limited time interval. Connections marked as *complete* are those with normal establishment and termination. Connections marked as *aborted* are those that have a normal establishment, but are reset by one of the end-points afterward. This is mainly

[4] Bro has other states that should not be reached when all packets are observed. Less than 0.1% of the connections in our datasets are terminated in those states owing to packet loss.

caused by some client applications that intentionally reset connections after exchanging application layer payload, to avoid the TCP termination delays (see [23]). Those are likely to be successful service requests and, therefore, are healthy in our context. Finally, the TCP connections classified as *unhealthy* in Table 1 are only those that could not carry productive traffic at the transport layer. Table 1 also shows the frequency of each state in our dataset [see column Freq. (%)].

3.2 Health Index from Non-sampled Data

When dealing with non-sampled data, Algorithm 2 receives flow records as input and outputs connection statistics augmented with a discrete label (the final connection state), therefore acting as a classifier [24]. Assuming the heuristic to reassemble flow records and label the rebuilt connections is correct, the health index will be equivalent to what would be calculated directly from full packet traces.

Because the packet-based alternative (i.e., Bro) produces similar output, a natural way of comparing the two approaches is by matching their output, in a typical comparison of classification models. Appendix 2 and [24, 25] provide the background on the methodology for comparing classification models used in this section.

Besides the comparison against a packet-based alternative, two extra analyses are performed. Firstly, because exporters may be configured differently and may implement only part of the IPFIX expiration rules (see Appendix 1), the validation is repeated in different export scenarios. Secondly, since we extend [11], we re-implement the original heuristic and show that our extension produces better results.

3.2.1 Scenarios

For checking the performance of our approach and verifying the effects of timeouts and expiration rules, three validation scenarios are defined:

- Scenario 1: Only timeouts expire flow records without any protocol checks. Several flow exporters, such as Vermont [26] and FlowMon [27], implement only these rules because of performance considerations.
- Scenario 2: Includes the rules of Scenario 1 and flow termination by TCP `RST`/`FIN`. Cisco IOS NetFlow and nProbe [28] export flows in this way. However, exporters may implement different heuristics. We use the heuristic of YAF: a `RST` in any direction or `FIN`s followed by `ACK`s in both directions terminate a flow. Note that YAF keeps track of both directions of a bidirectional flow to expire records. As stated in Sect. 3.1, we set up YAF to output always unidirectional records, keeping compatibility with unidirectional NetFlow exporters. This setup, however, does not affect YAF expiration heuristics.
- Scenario 3: Includes the rules of Scenario 2 and the creation of new flows by TCP `SYN`. Although not listed in IPFIX standards, this rule would force exporters to separate connections not properly closed into different records.

Sequence numbers are evaluated to differentiate SYN retransmissions from new connection attempts.

The impact of idle and active timeouts of the flow exporter is also measured for each scenario. This is done by fixing one of the two parameters to a large value (30 min) and varying the other parameter in distinct experiments. Note that both Bro and our method rely on timeouts for expiring inactive TCP connections. The numbers suggested in Bro for recording connections as accurately as possible (e.g., 2 h for inactive connections) are used in this validation.

The results of both methods are matched by looking for connections with common keys and the same start times (in s). A confusion matrix is then filled, and the *Precision*, *Recall*, and *F-measure* for both the healthy or unhealthy classes are calculated. Since our heuristic may produce a different number of connections than the ground truth, an artificial state is used to count *unmatched* connections in the confusion matrix. We refer the reader to Appendix 2 for the definitions of *Precision*, *Recall*, and *F-measure*, as well as a description about their meanings.

3.2.2 Results

The impact of timeout parameters in Scenario 1 is depicted in Fig. 2. Figure 2a, b show the precision and recall of our method when varying the idle timeout, while

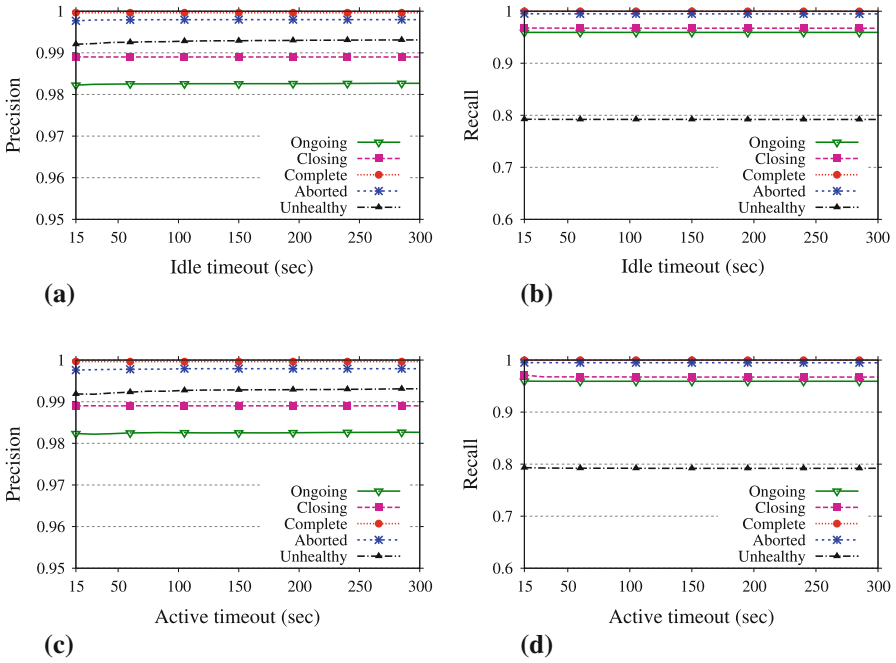

Fig. 2 Effects of timeout parameters on the precision and recall in Scenario 1. **a** Varying idle timeout—precision, **b** varying idle timeout—recall, **c** varying active timeout—precision, **d** varying active timeout—recall

Fig. 2c, d show the results when varying the active timeout. The results for the class *healthy* are shown in a finer granularity (see Table 1) to illustrate that most errors in this class are from ongoing and closing connections, which should not happen frequently in an on-line deployment.

The figures show that our method is invariant to timeouts. Moreover, the precision is very high for both classes, whereas the recall for the class unhealthy is lower. This means that our method reliably classifies both classes, but some unhealthy connections are *unmatched*. This happens because the exporter may report several connections in a single flow record. As a consequence, similarly to [11], in a scenario where the exporter expires records solely by means of timeouts, the number of unhealthy connections is slightly under-counted.

These figures also show that timeout parameters of exporters can be set to any value without impacting our results. This is especially important, since several exporters handle overload situations by expiring flow records faster—i.e., by reducing timeouts at run-time. The results of varying timeouts in Scenarios 2 and 3 are not shown, but the same flat lines in Fig. 2 are obtained in these scenarios.

The performance changes, however, when other expiration rules are applied. Figure 3a shows the F-measure for all scenarios when idle and active timeouts are fixed to 30 and 120 s,[5] respectively. The low recall of the class unhealthy decreases the F-measure in Scenario 1, as previously explained. Significant improvements are seen as more information is considered for flow expiration. The improvement in Scenario 2 is caused by the proper checks of TCP flags in the exporter, which prevent connections from being merged into a single record. Scenario 3 slightly improves the results for the same reason. Overall, the F-measure is close to 1 in Scenarios 2 and 3, meaning that both precision and recall are also close to 1 in these cases.

Finally, our heuristic to regroup records is compared to the original algorithm of [11]. Figure 3b depicts the F-measure per class of both methods in Scenario 3—Bro is used as ground truth for both variations. The method in [11] produces similar

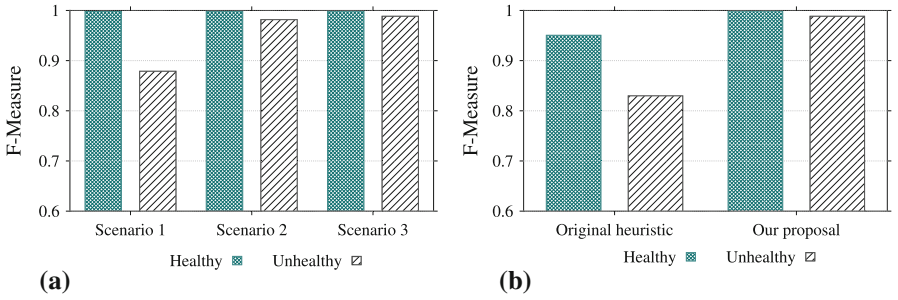

Fig. 3 Performance of our method when applied to non-sampled data. **a** Our method in different scenarios, **b** comparison to [11] in Scenario 3

[5] Since timeouts do not impact the results, other values would lead to similar conclusions.

results in other scenarios. The figure shows that our extension outperforms the original algorithm. Because of its extensive use of flags to regroup records, our heuristic is particularly more accurate when classifying unhealthy connections.

3.3 Health Index from Packet-Sampled Data

The validation with packet-sampled flows is performed against the same ground truth as in Sect. 3.2—i.e., Bro output produced from non-sampled packet headers. Because sampling implies loss of information, it is not possible to output individually labeled connections as in the non-sampled case. Instead, confidence intervals of the total number of connections per class are calculated. Similarly, expiration rules do not play an important role in the flow formation when sampling is applied, since only few packets per connection are observed.

Hence, Sect. 3.3.1 focuses on validating that our method calculates confidence intervals that include the original numbers counted from Bro output. Besides that, it shows that taking SYN retransmissions into account improves the confidence intervals. After that, Sect. 3.3.2 illustrates the effects of the sampling rate on the approach.

3.3.1 Confidence Intervals

We first evaluate the accuracy of our method in a simple example, to highlight the effects of SYN retransmissions. After that, the method is applied to the complete dataset used in the previous section. Figure 4a shows the intervals calculated by our method and the number of connections reported by Bro using only the traffic going to Facebook in a limited time interval. Confidence intervals for n and n_h are calculated as described in Eq. (1), with a significance level of 95 %, $r = 1$ and $p = 0.1$. In this example, we do not consider SYN retransmissions. Bro results show that the difference between n and n_h is very small most of the time. Around 7:55 a.m., some users are not able to access Facebook, resulting in an increase in the number of unhealthy connections. The confidence intervals always overlap, except

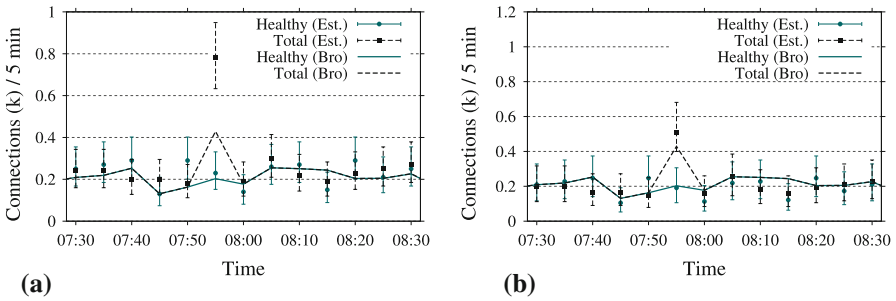

Fig. 4 An example of our estimations compared to the ground truth. **a** Without considering retransmissions, **b** considering SYN retransmissions

Table 2 Connections with more than one SYN packet (%)

State	State frequency	Originator	Responder
Ongoing	1.4	19.9	51.9
Closing	0.7	2.4	2.3
Aborted	26.6	1.6	2.1
Complete	57.6	0.8	1.1
Unhealthy	13.7	46.9	0.2
Total	100.0	7.6	1.9

on the slot at 7:55 a.m. Hence, the service is correctly identified to be unhealthy at that time.

However, it is also clear that $\hat{n}$ strongly diverges from n when there are unhealthy connections. This happens because retransmissions of SYN packets bias the estimators toward unhealthy connections. For the period in Fig. 4, Bro reports 3,087 healthy connections out of 3,314 in total. In the same period, $\hat{n}_h = 3,450$ and $\hat{n} = 3,930$ under sampling. Table 2 explains this difference by reporting the percentage of flows with SYN retransmissions in the complete dataset. In particular, 46.9 % of unhealthy connections include at least one SYN retransmission from originators. Therefore, the probability of sampling at least one SYN packet from these connections is higher.

Figure 4b illustrates the improvement when SYN retransmissions are taken into account. In comparison to the results in Fig. 4a, it can be seen that the estimation is more accurate when there are unhealthy connections.

When the approach is applied to packet-sampled flows of the complete dataset, confidence intervals calculated without considering SYN retransmissions include the real numbers reported by Bro in around 93 and 61 % of the cases for n_h and n, respectively. The retransmissions shown in Table 2 bias the intervals. The result for n is improved to 81 % when retransmissions are taken into account. Some bias therefore remains, because of connections with more than 1 retransmission, as discussed in Sect. 2.2.3. Nevertheless, the decision about the presence of unhealthy connections correctly becomes more conservative in the latter scenario. For instance, the health index is smaller than 1 in 79 time intervals when retransmissions are not taken into account, but only in 10 time intervals when retransmissions are considered.

Finally, the number k_S of SYN packets per flow is provided by YAF for the calculations in this section. This information can also be estimated directly from the number of packets and bytes in NetFlow records as follows. Given the standard TCP/IP headers, the size of SYN packets is a multiple of 4 bytes bigger than 40 (IPv4) or 60 (IPv6). In our data, SYN packets from originators rarely have more than 32 bytes of (TCP/IP) options. Therefore, k_S is set to the number of packets in the flow whenever the ratio of bytes per packet matches these characteristics and the record has *only* the SYN flag set—for all other SYN flows, $k_S = 1$. In our dataset, this method would provide an estimation for the total number of sampled SYN packets that is around 0.5 % lower than the real number reported by YAF.

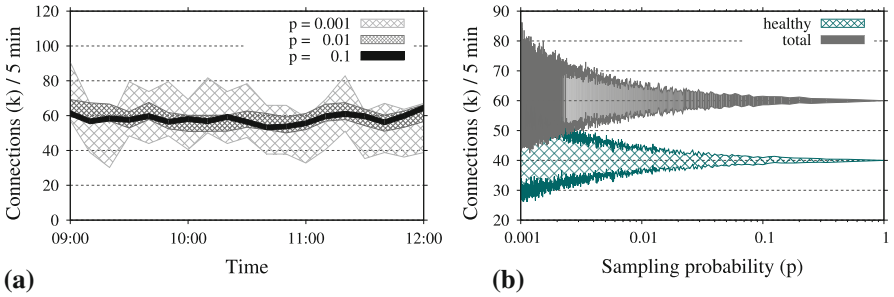

Fig. 5 Effects of the sampling rate. **a** Intervals for n when applying different p, **b** intervals for a fixed pair n and n_h

3.3.2 Effects of the Sampling Rate

Figure 5 illustrates the impact of the sampling rate on our method. Figure 5a depicts confidence intervals for n in a 3-h period. Three experiments are performed. In each of them, the complete dataset is submitted to a different sampling probability p, and confidence intervals are calculated using the methodology in Sect. 2.2.2 with a significance level of 95 % and $r = 1$. Intervals for n_h are not shown for the sake of clarity. As expected given Eq. (1), the intervals are wider for smaller p values.

Figure 5b illustrates this effect further, by plotting the confidence intervals when varying p with n and n_h fixed to arbitrary constants (note the logarithmic x-scale). Similar shapes would be obtained for other n and n_h as well. For each p, 10 random samples of $B(k|n, p)$ and $B(k|n_h, p)$ are taken, and confidence intervals for n and n_h are then calculated. Figure 5b shows only the average interval limits of these 10 repetitions to improve visualization. In this example, our method would report an unhealthy service when $p > 1/800$ (approximately). For smaller p values, however, the intervals overlap, preventing the difference between $\hat{n}$ and $\hat{n}_h$ to be considered significant. Therefore, our method is more conservative under lower p values. The conclusions also hold when `SYN` retransmissions are taken into account.

Finally, packets are randomly sampled with probability p in all our experiments. Several other sampling algorithms have been standardized in [29]. In practice, periodic sampling is the most common alternative, being implemented, for instance, in Cisco's Sampled NetFlow. In [13], it is shown that the distributions resulting from random and periodic sampling are statistically distinguishable, but their difference is very small. The effects of periodic sampling in our results are therefore negligible.

Other sampling algorithms, such as the *sample and hold* [30] and *adaptive sampling* [31], could potentially reduce the effects of p shown in Fig. 5, improving our method. However, these algorithms are hardly found in existing flow exporters and, therefore, are not analyzed in this paper.

4 Non-sampled Data: Popular Web Services

This section shows that the health index calculated from non-sampled flow data is informative to monitor performance problems in cloud services. Similar analysis will be performed in Sect. 5 using packet-sampled flow data. In the following, Sect. 4.1 describes our dataset and methodology. After that, the health of popular services is analyzed in Sect. 4.2. Finally, Sect. 4.3 discusses the obtained results.

4.1 Dataset and Methodology

A prototype of our method has been implemented as a plug-in for NfSen [18] and is deployed at our university network. All flow data exported by edge routers (non-sampled NetFlow v9) are processed, and statistics of the reassembled records are archived in 1-min bins. This section evaluates the data collected by this prototype in the first 10 weeks of 2013.

Four organizations are analyzed: Dropbox, Twitter, Google and Facebook. These organizations have been selected because their services are highly popular in our network. We have documented all IP addresses serving Dropbox, and the MaxMind databases are used to filter flows of Twitter, Google and Facebook. Note that some contents related to these organizations are hosted in third-party networks (e.g., Facebook's static content is hosted by Akamai). Our results do not include this traffic.

The most serious performance anomalies in this dataset have been analyzed manually and are discussed in the follow. Furthermore, the official channels in which Dropbox [32], Twitter [33] and Google [34] report the status of their services are used to understand how our system reacts to different performance anomalies. Facebook is left out of this last analysis because no historic information about its services could be found.

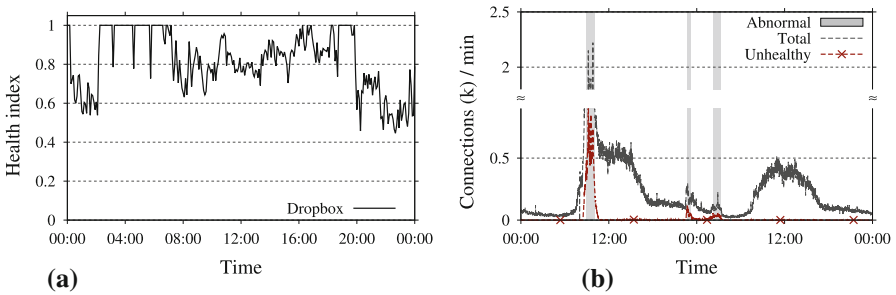

Fig. 6 Monitoring Dropbox within the UT network. **a** Blacklisted clients: Mar 7, 2013, **b** Jan 10–11, 2013

4.2 Health Analysis

Our results show an index normally close to 1 in all cases, with some fluctuations during the nights, because of the little amount of traffic in the campus in the period. However, several abnormal situations can be noticed. Figure 6a depicts a 1-day example of the index for Dropbox. By manual inspection, we have identified that this low index is caused by abandoned Dropbox clients, which are sometimes blacklisted by the university firewall. Such clients keep trying to contact Dropbox servers, increasing the number of unhealthy connections. Even though the root cause of the problem is in our own network, the prototype correctly points to an unhealthy service.

After removing blacklisted clients, few abnormal intervals remain. The index for the selected organizations is lower than 0.8 in 524 min (0.52 % of the 1-min bins) and lower than 0.7 in 285 min (0.28 % of the 1-min bins), for instance. For validating these results, all cases in which the index stays below 0.7 for more than 5 consecutive minutes have been evaluated. These thresholds are chosen in order to limit the time consuming manual analysis to the most severe service degradations. In total, 8 (non-overlapping) intervals with a total duration of 201 min have been manually analyzed, distributed among Dropbox (7) and Twitter (1).

Dropbox reported problems twice in the period. Abnormal intervals (5 in total) appear in our dataset in both cases. Figure 6b depicts examples seen on Jan 10–11. By inspecting the remaining 2 abnormal intervals, we conclude that only a non-essential part of the service has been affected (which collects run-time statistics—see [35]), suggesting that the 7 most serious cases identified by our prototype are all correct.

The abnormal interval related to Twitter (Fig. 7a) does not coincide with official reports. However, a simultaneous increase on the number of unhealthy connections to several destinations can be seen in the data. Hence, the system seems to correctly point to a problem in our network, although the root cause is unclear in this case.

Twitter reported problems 6 times in the period. None of the reports passes the threshold set to this experiment. However, 2 of them are clearly visible in our dataset. Figure 7b shows an example. The interval with an increased number of unhealthy connections coincides with a Twitter's announcement of site problems.

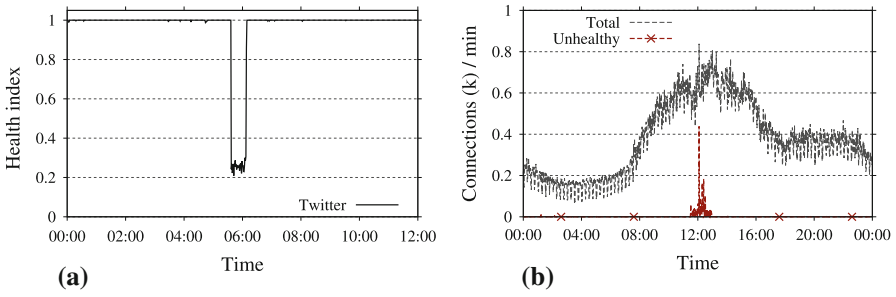

Fig. 7 Monitoring Twitter within the UT network. **a** 12 h on Feb 26, 2013, **b** Feb 27, 2013

Twitter also reported site problems affecting part of its users on Jan 17 and Jan 31. These 2 cases are not visible in our data, and it is not clear whether our users experienced problems. The remaining 2 cases involved a malfunction and a bug of specific site functionalities, which are not expected to be detected by our system.

Finally, Google reported 5 problems in the period, including longer response times and errors with attachments on Gmail. None of the issues prevented users from accessing the services. Our system, as expected, does not report such problems.

4.3 Discussion

The analyzed companies reported 13 incidents in the 10 weeks interval studied in this section. Among those, 7 cases are out of our target, since users were still able to access the services. Moreover, 2 other incidents might not have affected our network. The remaining 4 cases are all detectable in our data, even though the services were partially accessible. Our method also identified other anomalies that either were not officially reported or affected only our own network. Overall, these cases show the applicability of the health index to monitor severe problems in cloud services.

Note that, while we have analyzed the drops in the health index manually in this section, a more user friendly alert system will require an automatic post-processing of the output of Algorithm 1. The design of such a system is left for future work.

5 Packet-Sampled Data: The WikiLeaks Case

We now validate the applicability of our method using packet-sampled flow data. The case study in this section is the series of attacks promoted by a group of *hacktivists* (named *Anonymous*) after the release of US embassy cables by WikiLeaks in 2010. As a reaction to those leaks, several companies retracted their business support to WikiLeaks. *Anonymous* reacted to this boycott by coordinating denial-of-service attacks against MasterCard, PayPal, among others. Some of the targets were reported to be inaccessible during the attacks, although this has never been confirmed.

Anonymous used a tool named *LOIC* against the targets. *LOIC* has been originally developed for stress testing on servers and, as such, does not implement any sophisticated attack method. The tool uses standard libraries for establishing TCP connections and sending user-defined messages. Hence, *LOIC* generates regular traffic at the transport layer. For this attack, the tool was extended to obtain configuration parameters from IRC servers—i.e., *hacktivists* voluntarily joined a botnet.

Section 5.1 describes our dataset and methodology. Section 5.2 evaluates the health of several targets. After that, Sect. 5.3 discusses our results.

5.1 Dataset and Methodology

Packet-sampled NetFlow records (with sampling probability $p = 0.01$) collected at an international backbone are used in this analysis. The following services involved in the attacks are analyzed: PayPal's payment Web services, MasterCard's Web page, Moneybookers' Web page, the Web page of Senator Liberman and PayPal's blog. We give particular attention to the case of PayPal, since the data indicate that active attack containment measures have been taken. Our decision to focus on these websites is motivated exclusively by the amount of traffic to each target in our dataset. Several other organizations and politicians have been targeted by *Anonymous*. A complete description of the *WikiLeaks Cablegate*, including all targets and *Anonymous*'s motivations for attacking each specific website, is presented in [10, 36].

For each target, the method in Sect. 2.2 estimates the number of connections as well as the number of healthy connections. We do not compensate for `SYN` retransmissions in the analysis, since the routers in this backbone do not account TCP `ACKs` reliably. For each time bin of 1 min, confidence intervals for a significance level of 95 % and parameter $r = 10$ [see Eq. (1)] are calculated—therefore, all results in the following are smoothed by a 10-min window moving average.

5.2 Health Analysis

PayPal's payment Web service was targeted by *hacktivists* on Dec 9, 2010. In order to provide a better insight into our approach to sampled flow data, we show the computed confidence intervals for the estimated number of connections, instead of the health index. Figure 8a shows the obtained intervals for the number of connections to PayPal's service. Whenever the confidence intervals do not overlap, a possible degradation of the service has occurred. We observe that the number of unhealthy connections increased for more than 1 h around 15:00 and for almost 2 h after 22:30, which indicates anomalies. For all other periods, there are no clear indications that the attack caused any damage or health degradation.

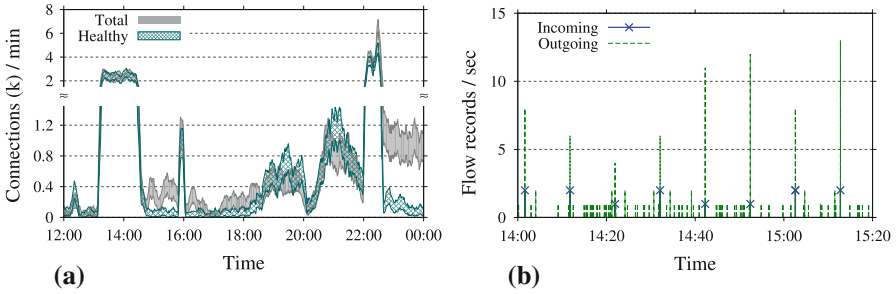

Fig. 8 Traffic to PayPal's service on Dec 9. Note the discontinuity in the y axis. **a** Health analysis, **b** firewall regulating a *hacktivist*

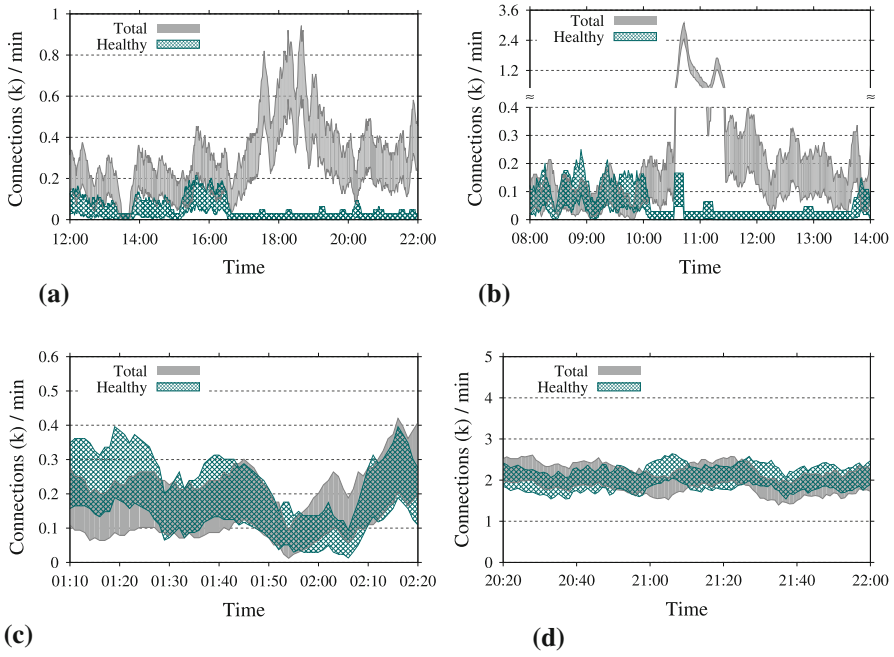

Fig. 9 Health of several *Anonymous*' targets. Note the discontinuity in the *y* axis. **a** MasterCard's Web page on Dec 8, **b** Moneybookers' Web page on Dec 10, **c** Senator Lieberman's Web page on Dec 8, **d** PayPal's Blog on Dec 10

For the periods in which the confidence intervals do not overlap, there are indications that *hacktivists* were being blocked by a firewall. Figure 8b illustrates that, by showing the number of raw flow records from/to one potential *hacktivist*. The figure shows that incoming traffic occurred in well-spaced intervals, which is a typical behavior of gray-listing firewalls. The way in which *LOIC* is implemented explains the peaks of outgoing records every time the server started responding again. Since the service can be considered unhealthy from the point of view of the *hacktivist*, the algorithm produces correct results also for this case.

Figure 9 shows the analysis for other targets. Figure 9a shows two clearly different phases regarding MasterCard's Web page on Dec 8: until around 17:00, the number of healthy connections was consistently lower than the total number of connections, although the difference cannot be considered significant in all time intervals; after that, there were almost no responses from servers anymore. During the first phase, it is likely that the service was experiencing problems, but it was still able to handle a portion of the traffic. In the second phase, either the server was completely off-line, or *hacktivists* were being blocked. Note that this backbone normally does not transport significant amount of requests to MasterCard and, therefore, most of the observed traffic may have been generated by *hacktivists*. As in the case of PayPal, our algorithm correctly indicates an unhealthy service in several intervals, even though this might be a consequence of firewalls blocking *hacktivists*.

The same comments are valid for the Web page of Moneybookers, as depicted in Fig. 9b. For this case, however, the Web page was healthy almost all the time, until it stopped handling *hacktivists*' requests completely (around 10:30). Finally, Fig. 9c, d show that both the website of US Senator Lieberman and PayPal's blog were not affected by the attacks during the analyzed time intervals.

5.3 Discussion

Our approach indicates that some targets were unhealthy in parts of the analyzed periods. However, the high number of unhealthy connections may be a consequence of firewalls blocking *hacktivists*. A closer look into the flows suggests that the attacks only marginally affected normal users in the analyzed network. The constant behavior of *LOIC* makes it easy to identify *LOIC* traffic even in flow datasets. By manually analyzing the traffic in the attack days, it can be seen that most unhealthy connections are from few sources producing an abnormally high number of requests that match with *LOIC* behavior. This pattern is, on the other hand, not observed on other days in the same network. Therefore, it is likely that most unhealthy connections were produced by potential hacktivists only.

Finally, some *Anonymous*' targets (e.g., Visa) are hosted by cloud providers that use dynamic IP address allocation. As a consequence, the traffic from/to those services could not be isolated, since the addresses used by the services during the attacks are unknown. Section 7 discusses this potential limitation further.

6 Related Work

This work discusses how to measure the health of services deployed in the cloud. Several works rely on active measurements to analyze the performance of services or to benchmark cloud providers [37–39]. Other works [40, 41] focus on measuring the performance of the infra-structure providing the services. In some cases [35, 42, 43], a particular cloud service or provider is analyzed in details. Our work differs from those in several aspects. Firstly, the increasing variety of client and server platforms—e.g., mobile devices and *Software as a Service* (*SaaS*) offers [44]—motivates our decision to monitor at the network only. Secondly, we do not focus on a specific service, but instead, we propose a generic method applicable to a variety of cloud services. Lastly, we take a passive approach to cloud monitoring because the active alternatives lack the ability to capture the impact of problems on actual end users. Besides that, they generate extra load to services and the network. Active approaches are, however, complementary to ours, since they provide another view of the cloud services. For example, in contrast to active methods, our method can only identify problems in a cloud service if end users are interacting with it.

The use of flow data for measuring the health of a remote service is central in our method. A similar approach is proposed in [45, 46] to assist operators in identifying connectivity problems and outages. Incoming/outgoing flow records are matched and alerts are triggered when the number of records without a match increases. Our work differs from those in two aspects: firstly, because we aim at checking the

health of cloud services, evaluating whether the traffic is valid or not is prime to our goal. When using non-sampled data, our method uses all available flag information to classify connections precisely. Secondly, [45, 46] assume non-sampled flow data, while our method also provides support to packet-sampled flows.

When dealing with non-sampled data, our work relies on a technique for reassembling flow records. In [11], the authors present an algorithm for grouping NetFlow records into TCP connections. The authors of [22] use an opposite approach, by tuning timeout parameters for approximating flow records to TCP connections in a better way. In contrast to [11], we intensively make use of all flag information when reassembling flow records, obtaining more accurate results. Additionally, we do not evaluate a single NetFlow source, but verify the effects of several implementation decisions on the approach. Our work diverges strongly from [22], since timeout parameters are shown to have a negligible impact on our results. Both works also lack support for packet-sampled data, which is provided by our method.

When dealing with packet-sampled flows, we rely on the theoretical framework presented in [13]. This work shows how to estimate several properties of the original data stream, such as the flow length distribution, using only packet-sampled flow records. We follow a similar reasoning to estimate the distributions of the number of connections per health state. We also go one step further, by proposing an estimation that does not require the assumption of a single `SYN` packet per connection, which improves the results when there are unhealthy connections.

Finally, some works (e.g., [47]) aim at discovering dependencies in flow data automatically. Such solutions will be considered in our future works, since they could help to identify problems beyond the transport layer.

7 Limitations and Future Work

7.1 Traffic Isolation

Our method assumes that flow records related to a service are already isolated (see Algorithm 1). Our case studies show that the method performs well if flow records are filtered properly. However, manually isolating traffic based on IP addresses is neither convenient nor always effective, as exemplified in Sect. 5. New methods to identify flows, such as the inclusion of external information in flow definitions (e.g., see [48]), are needed and will be included in the approach in future works.

7.2 Unhealthy Application Layer Requests

Application layer protocols are not analyzed by our method. This is motivated by our goal of building a generic monitoring system. Our results show that the health index is sufficiently informative under the most extreme conditions, in which service requests cannot even be started. However, it is also common that servers are still able to handle part of users' requests under less severe problems.

In these situations, we expect a mix of healthy traffic, unhealthy traffic at the transport layer, and unhealthy traffic at the application layer. Figure 6b shows an example of that. The number of connections to Dropbox was significantly increased during the periods in which the service performance degraded. A closer look into the flows reveals an increase also in the number of healthy connections, suggesting errors at the application layer. Depending on the proportion of unhealthy traffic per protocol layer, such problems are detectable by our approach, as shown in Fig. 6b.

A complete characterization of how these mixed problems occur in practice will be performed in future works. Longitudinal data containing more cases of performance problems in cloud services are needed for that. Moreover, we plan to extend our method toward the application layer. In particular, some protocols (e.g., HTTP) are also widely used in cloud services, and therefore are a natural extension to the approach. In an opposite direction, some services (e.g., cloud storage) are of high importance to consumers, justifying the development of application-specific analyzers. Such analyzers could perform a next classification stage, evaluating the traffic currently reported as healthy by our method.

7.3 Monitoring Other Performance Problems

This paper introduces a metric (health index) that helps to reveal the status of cloud services based on flow data. As illustrated in Sect. 4, other equally important performance aspects, such as the service response time, cannot be monitored by means of the health index. Our future works will also investigate whether other performance problems can be monitored from flow data.

8 Conclusions

Our contribution is a method for monitoring the health of cloud services that relies solely on NetFlow/IPFIX data. This is achieved by means of a *health index*, which is proposed and evaluated in this paper. Our method has the distinct characteristic of being able to cope with both non-sampled and sampled data. Hence, it explicitly targets high-speed networks. In the case of non-sampled data, our method reassembles flow records into the original TCP connections, and classifies the connections as *healthy* or *unhealthy*. For sampled data, it applies a stochastic technique for estimating the number of connections per health state.

The method was validated against a packet-based ground truth built with Bro, using packet traces collected at the University of Twente. The effects of several flow export parameters were evaluated, and we conclude that the approach is robust against variations in flow exporter setups. In addition, our results show that the method precisely estimates the health index also from packet-sampled flows. The applicability of the approach was confirmed first by analyzing performance incidents in cloud services and, then, by studying data collected during the *WikiLeaks Cablegate*. Our results show, for instance, that the users of targeted services were only marginally affected by the attacks, contradicting *Anonymous*' claims.

Determining the health of a remote service solely from network traces is challenging. Our proposal is by no means a comprehensive solution to all monitoring needs of cloud consumers. However, by restricting the scope to serious performance problems, and by relying on flows exported by widely deployed technologies, our method delivers an essential first layer of monitoring, while being applicable to a wide-range of services and scalable to high-speed networks.

Finally, a prototype of our proposal has been implemented as an open source plug-in for NfSen and is available at http://www.simpleweb.org/wiki/Cloud_Monitoring.

Acknowledgments This work has been carried out in the context of the FP7 FLAMINGO Network of Excellence Project (CNECT-ICT-318488), the EU FP7-257513 UniverSelf Collaborative Project and the IOP GenCom project Service Optimization and Quality (SeQual). SeQual is supported by the Dutch Ministry of Economic Affairs, Agriculture and Innovation via its agency Agentschap NL

Appendix 1: Background Flow Monitoring

Definition

We assume flow records to be defined as in NetFlow/IPFIX [7]: a unidirectional set of packets crossing an observation point, sharing a common set of attributes (the *flow key*). NetFlow v9 [6] uses a fixed flow key composed of source and destination IP addresses and port numbers, IP protocol number, IP type of service and the input interface index. IPFIX offers more flexibility by supporting variable flow definitions [7]. More important for this work, NetFlow and IPFIX [6, 49] define the following reasons for a flow record to be exported:

1. *Idle timeout*: No packets belonging to a flow have been observed for a specified period of time;
2. *Active timeout*: A flow has been active for a specified period of time;
3. *End of flow detected*: The end of a flow has been identified. For instance, TCP flags can be used to expire flow records;
4. *Forced end*: An external event, such as a shutdown of the exporting device, forces all flow records to be exported;
5. *Lack of resources*: Special heuristics can expire flow records prematurely in case of resource constraints on the exporting device.

After expiration, flow records are sent to flow collectors. In IPFIX, only the protocol for transporting records from exporters to collectors is specified. All details related to flow measurement, including the configuration of expiration rules, are considered implementation decisions [50]. Therefore, flow data can differ per exporting device. Since our method aims to be robust against different flow definitions, it must handle data exported according to any of these rules. See Sect. 3 for a discussion about the consequences of different flow definitions to our methodology.

[6] Similar figures would be obtained with other typical setups, such as the Cisco NetFlow default timeouts (15 and 1,800 s, respectively).

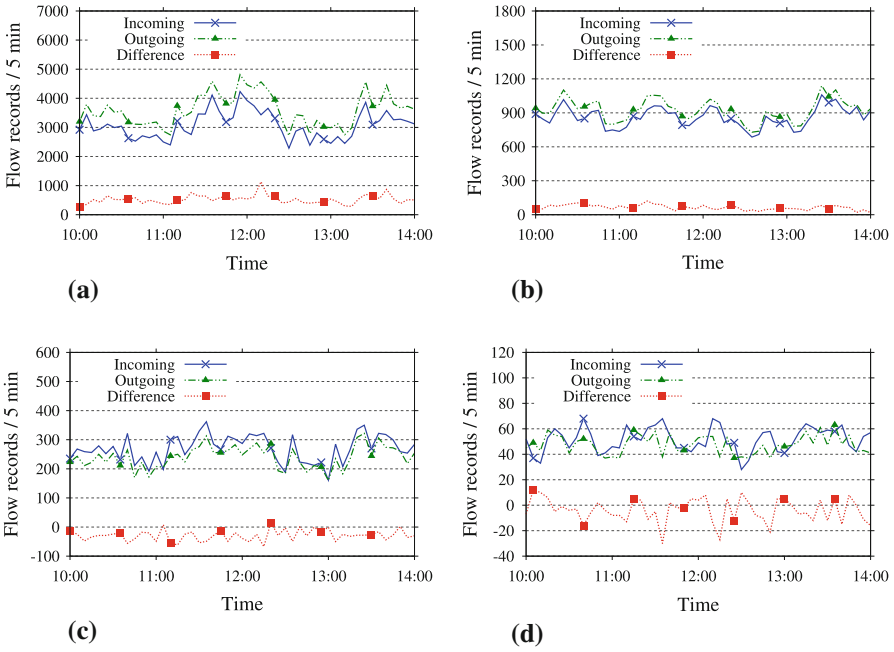

Fig. 10 Incoming/outgoing flow records for healthy Web services. **a** Google (non-sampled), **b** Facebook (non-sampled), **c** Google (sampled, $p = 0.01$), **d** Facebook (sampled, $p = 0.01$)

Why Raw Flow Records are Not Sufficient

All NetFlow/IPFIX expiration rules can cause TCP connections to be split into multiple flow records. However, since TCP is bidirectional, one should expect that those rules would produce the same number of records in both traffic directions for a healthy service. Figure 10 illustrates the invalidity of this reasoning using traffic to Google and Facebook in our network.

Figure 10a, b depict the number of flow records when packet sampling is not applied. The packet traces of our validations are converted into NetFlow records, with idle and active timeouts set to the typical values of our flow exporters (30 and 120 s, respectively).[6] The difference between the quantities is also shown, with negative values representing more incoming records. In general, more outgoing than incoming records are seen. This difference is mainly caused by the widespread use of TCP `RST` to terminate connections (see [23]). This can result in an extra flow record in either traffic direction, depending on timeout parameters.

Interestingly, an opposite pattern is seen when sampling is applied. For illustration, Fig. 10c, d depict the results of applying independent random sampling with probability $p = 0.01$ to the same packet traces. The figures show that the number of incoming records tends to be higher than the number of outgoing records. This happens because servers normally send more packets to clients, increasing the probability of sampling incoming flows.

Table 3 Confusion matrix M when classifying instances in n classes

True classes	Prediction			
	C_1	C_2	...	C_n
C_1			...	
C_2			...	
...	...	...	...	...
C_n			...	

Appendix 2: The Performance of Classification Models

As defined by [24], "a classification model (or classifier) is a mapping from instances to predicted classes". In the classification problem studied in Sect. 3, the instances are TCP connections, which are mapped to discrete labels (*healthy*, *unhealthy*, *unmatched*). This appendix reviews the background on performance metrics commonly used to compare classifiers.

Confusion Matrix

The confusion matrix is a way of presenting results when evaluating classifiers [24]. It presents the original number of instances per class (i.e., the ground truth in a test set) versus the total number of instances that are predicted to belong to each class. Table 3 shows an example of such a matrix M, in a problem composed of n classes ($n = 3$ in our case). The diagonal contains correctly classified instances, whereas the remaining cells contain the errors per class.

Note that, in our problem, no instance can be correctly classified as *unmatched*, since no connections of this class exist in the ground truth set. This artificial state, however, is needed because the output of our approach may have different cardinality than the ground truth set. Instances in the *unmatched* state penalize our method when it wrongly merges flow records.

Performance Metrics

Several performance metrics to compare classification schemes can be derived from the confusion matrix. The most simplistic one is the *accuracy* A, which is defined as the overall fraction of correctly classified instances:

$$A = \frac{\sum_{i=1}^{n} M_{ii}}{\sum_{i=1}^{n} \sum_{j=1}^{n} M_{ij}} \tag{5}$$

The accuracy, however, fails to provide insights about the source of errors of a classifier. This is particularly the case when the different classes in the classification problem are unbalanced—that is, when the probability of observing instances of a specific class is much lower than the probability of observing the remaining classes.

It is very common that the rarest class is exactly the most important one to be correctly classified. In our case, instances of the *healthy* class are much more frequent, accounting for almost 90 % of the samples in our test set. Any classification model could reach high accuracy by predicting that all instances belong to the *healthy* class, although such models would be of no use. Metrics that capture the performance of a classifier in a finer granularity are therefore needed.

Three metrics are of primary interest in our analysis: *Precision* (P_i), *recall* (R_i), and *F-measure* (F_i) [24, 25]. Assuming $\boldsymbol{\omega}$ to be the classes in a classification problem (i.e., $\boldsymbol{\omega} = (healthy, unhealthy, unmatched)$), these metrics are defined as a function of the class ω_i. The precision P_i represents the fraction of instances correctly classified as being of class ω_i. It can be calculated from the confusion matrix, as follows.

$$P_i = \frac{M_{ii}}{\sum_{j=1}^{n} M_{ji}} \tag{6}$$

The recall R_i is the fraction of all original instances of class ω_i that are correctly classified:

$$R_i = \frac{M_{ii}}{\sum_{j=1}^{n} M_{ij}} \tag{7}$$

As for the overall accuracy, tuning a classifier to optimize only one of these metrics for a specific class may produce degenerated models. For instance, a classifier that labels all instances as ω_i has $R_i = 1$, but P_i equals to the prior probability of ω_i. To overcome that, the *F-measure* (F_i) can be applied to assess the classifier:

$$F_i = \frac{P_i R_i}{(1-\alpha)P_i + \alpha R_i} \quad 0 \leq \alpha \leq 1 \tag{8}$$

The F-measure is a combination of precision and recall that increases faster when both metrics are simultaneously increased. The parameter α weights the importance of each metric in the results. This can be used, for instance, when failing to identify instances of a class has a different cost than making classification mistakes. In all our experiments, we calculate F_i using $\alpha = 0.5$, which means that both precision and recall have equal importance. We refer the readers to [25] for a deeper discussion and to [51] for a practical example of the F-measure use in another domain.

References

1. Hajjat, M., Sun, X., Sung, Y.W.E., Maltz, D., Rao, S., Sripanidkulchai, K., Tawarmalani, M.: Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. SIGCOMM Comput. Commun. Rev. **40**(4), 243–254 (2010)
2. Rish, I., Brodie, M., Odintsova, N., Ma, S., Grabarnik, G.: Real-Time Problem Determination in Distributed Systems Using Active Probing. In: Proceedings of the IEEE/IFIP Network Operations and Management Symposium, NOMS'04, pp. 133–146 (2004)

3. Xu, K., Wang, F., Wang, H.: Lightweight and informative traffic metrics for data center monitoring. J. Netw. Syst. Manag. **20**, 226–243 (2012)
4. Clarke, R.: How reliable is cloudsourcing? A review of articles in the technical media 2005–11. Comput. Law Secur. Rev. **28**(1), 90–95 (2012)
5. Gehlen, V., Finamore, A., Mellia, M., Munafò, M.M.: Uncovering the Big Players of the Web. In: Proceedings of the 4th International Conference on Traffic Monitoring and Analysis, TMA'12, pp. 15–28 (2012)
6. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational) (2004)
7. Claise, B.: Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information. RFC 5101 (Standards Track) (2008)
8. Garcia-Dorado, J., Finamore, A., Mellia, M., Meo, M., Munafò, M.M.: Characterization of ISP traffic: trends, user habits, and access technology impact. IEEE Trans. Netw. Serv. Manag. **9**(2), 142–155 (2012)
9. Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., Jahanian, F.: Internet Inter-domain Traffic. In: Proceedings of the ACM SIGCOMM 2010 Conference, SIGCOMM'10, pp. 75–86 (2010)
10. Mansfield-Devine, S.: Anonymous: serious threat or mere annoyance? Netw. Secur. **2011**(1), 4–10 (2011)
11. Sommer, R., Feldmann, A.: NetFlow: Information Loss or Win? In: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement, IMW'02, pp. 173–174 (2002)
12. Paxson, V.: Bro: a system for detecting network intruders in real-time. Comput. Netw. **31**(23–24), 2435–2463 (1999)
13. Duffield, N., Lund, C., Thorup, M.: Estimating flow distributions from sampled flow statistics. IEEE/ACM Trans. Netw. **13**(5), 933–946 (2005)
14. Draper, N., Guttman, I.: Bayesian estimation of the binomial parameter. Technometrics **13**(3), 667–673 (1971)
15. Tang, V.K.T., Sindler, R.B., Shirven, R.M.: Bayesian estimation of n in a binomial distribution. Tech. Rep. CRM 87–185, Center for Naval Analyses (1987)
16. Tang, V.K.T., Sindler, R.B.: Confidence interval for parameter n in a binomial distribution. Tech. Rep. CRM 86–265, Center for Naval Analyses (1987)
17. Finamore, A., Mellia, M., Meo, M., Munafò, M.M., Rossi, D.: Experiences of Internet traffic monitoring with Tstat. IEEE Netw. **25**(3), 8–14 (2011)
18. Haag, P.: Watch your flows with NfSen and NFDUMP. 50th RIPE Meeting. http://meetings.ripe.net/ripe-50/presentations (2005). Accessed June 2013
19. Fullmer, M., Romig, S.: The OSU Flow-tools Package and CISCO NetFlow Logs. In: Proceedings of the 14th USENIX conference on System administration, LISA'00, pp. 291–304 (2000)
20. Inacio, C.M., Trammell, B.: YAF: Yet Another Flowmeter. In: Proceedings of the 24th International Conference on Large Installation System Administration, LISA'10, pp. 1–16 (2010)
21. MaxMind: GeoIP Organization. http://www.maxmind.com/en/organization (2013). Accessed June 2013
22. Limmer, T., Dressler, F.: Flow-Based TCP Connection Analysis. In: Proceedings of the 2nd IEEE International Workshop on Information and Data Assurance, WIDA'09, pp. 376–383 (2009)
23. Arlitt, M., Williamson, C.: An analysis of TCP reset behaviour on the Internet. SIGCOMM Comput. Commun. Rev. **35**(1), 37–44 (2005)
24. Fawcett, T.: An introduction to ROC analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
25. van Rijsbergen, C.: Information Retrieval, 2 edn. Butterworth, London (1979)
26. Lampert, R.T., Sommer, C., Munz, G., Dressler, F.: Vermont—A Versatile Monitoring Toolkit for IPFIX and PSAMP. In: Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation, MonAM'06 (2006)
27. Čeleda, P., Kováčik, M., Koníř, T., Krmíček, V., Špringl, P., Žádník, M.: FlowMon Probe. Tech. Rep., CESNET (2007)
28. Deri, L.: nProbe: An Open Source NetFlow Probe for Gigabit Networks. In: Proceedings of the Terena, TNC'03 (2003)
29. Zseby, T., Molina, M., Duffield, N., Niccolini, S., Raspall, F.: Sampling and Filtering Techniques for IP Packet Selection. RFC 5475 (Standards Track) (2009)
30. Estan, C., Varghese, G.: New directions in traffic measurement and accounting. SIGCOMM Comput. Commun. Rev. **32**(4), 323–336 (2002)
31. Estan, C., Keys, K., Moore, D., Varghese, G.: Building a Better NetFlow. In: Proceedings of the ACM SIGCOMM 2004 Conference, SIGCOMM'04, pp. 245–256 (2004)

32. Dropbox: DropboxOps. http://twitter.com/DropboxOps (2013). Accessed June 2013
33. Twitter: Status. http://status.twitter.com (2013). Accessed June 2013
34. Google: Apps Status Dashboard. http://www.google.com/appsstatus (2013). Accessed June 2013
35. Drago, I., Mellia, M., Munafò, M.M., Sperotto, A., Sadre, R., Pras, A.: Inside Dropbox: Understanding Personal Cloud Storage Services. In: Proceedings of the 12th ACM Internet Measurement Conference, IMC'12, pp. 481–494 (2012)
36. Pras, A., Sperotto, A., Moura, G.C.M., Drago, I., Barbosa, R.R.R., Sadre, R., de Oliveira Schmidt, R., Hofstede, R.: Attacks by "Anonymous" WikiLeaks Proponents not Anonymous. Tech. Rep. TR-CTIT-10-41, CTIT, University of Twente, Enschede (2010)
37. Kossmann, D., Kraska, T., Loesing, S.: An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In: Proceedings of the ACM SIGMOD International Conference on Management of data, SIGMOD'10, pp. 579–590 (2010)
38. Lenk, A., Menzel, M., Lipsky, J., Tai, S., Offermann, P.: What Are You Paying for? Performance Benchmarking for Infrastructure-as-a-Service Offerings. In: Proceedings of the 4th IEEE International Conference on Cloud Computing, CLOUD'11, pp. 484–491 (2011)
39. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: Comparing Public Cloud Providers. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC'10, pp. 1–14 (2010)
40. Meng, S., Iyengar, A.K., Rouvellou, I.M., Liu, L., Lee, K., Palanisamy, B., Tang, Y.: Reliable State Monitoring in Cloud Datacenters. In: Proceedings of the 5th IEEE International Conference on Cloud Computing, CLOUD'12, pp. 951–958 (2012)
41. Meng, S., Liu, L.: Enhanced monitoring-as-a-service for effective cloud management. IEEE Trans. Comput. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6231620&queryText%3DEnhanced+Monitoring-as-a-Service (2012)
42. Hu, W., Yang, T., Matthews, J.N.: The good, the bad and the ugly of consumer cloud storage. SIGOPS Oper. Syst. Rev. **44**(3), 110–115 (2010)
43. Wang, G., Ng, T.E.: The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. In: Proceedings of the 29th Conference on Information Communications, INFOCOM'10, pp. 1–9 (2010)
44. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl. **1**, 7–18 (2010)
45. Glatz, E., Dimitropoulos, X.: Classifying Internet One-Way Traffic. In: Proceedings of the 12th ACM Internet Measurement Conference, IMC'12, pp. 37–50 (2012)
46. Schatzmann, D., Leinen, S., Kögel, J., Mühlbauer, W.: FACT: Flow-Based Approach for Connectivity Tracking. In: Proceedings of the 12th International Conference on Passive and Active Network Measurement, PAM'11, pp. 214–223 (2011)
47. Caracas, A., Kind, A., Gantenbein, D., Fussenegger, S., Dechouniotis, D.: Mining Semantic Relations using NetFlow. In: Proceedings of the 3rd IEEE/IFIP International Workshop on Business-driven IT Management, BDIM'08, pp. 110–111 (2008)
48. Bermudez, I., Mellia, M., Munafò, M.M., Keralapura, R., Nucci, A.: DNS to the Rescue: Discerning Content and Services in a Tangled Web. In: Proceedings of the 12th ACM Internet Measurement Conference, IMC'12, pp. 413–426 (2012)
49. Quittek, J., Bryant, S., Claise, B., Aitken, P., Meyer, J.: Information Model for IP Flow Information Export. RFC 5102 (Standards Track) (2008)
50. Trammell, B., Boschi, E.: An introduction to IP flow information export (IPFIX). Commun. Mag. **49**(4), 89–95 (2011)
51. Lewis, D.D., Gale, W.A.: A Sequential Algorithm for Training Text Classifiers. In: Proceedings of the 17th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR'94, pp. 3–12 (1994)

Author Biographies

Idilio Drago is a Ph.D. student at the Design and Analysis of Communication Systems Group (DACS) of the University of Twente, The Netherlands. He received his M.Sc. in Computer Science from the Federal University of Espírito Santo, Brazil in 2007. His Ph.D. thesis titled "Monitoring Cloud Services" will be presented in Dec. 2013.

Rick Hofstede is a Ph.D. student at the University of Twente, The Netherlands, where he graduated in Telematics in 2009 (B.Sc.) and in 2011 (M.Sc.). The working title of his thesis is "Real-Time and Resilient Intrusion Detection: A Flow-Based Approach". His main topics of interest are network security (intrusion detection and forensics in particular), Internet measurements and network data visualization.

Ramin Sadre is an Assistant Professor at the Distributed and Embedded Systems group of the Aalborg University, Denmark. He received a Ph.D. degree from the University of Twente for his thesis titled "Decomposition Based Analysis of Queueing Networks". His research interests include traffic modeling, the design and analytical performance evaluation of communication systems, and the design of network intrusion detection systems.

Anna Sperotto is a Postdoctoral researcher at the Design and Analysis of Communication Systems Group (DACS) of the University of Twente, The Netherlands. She received a M.Sc. degree in Computer Science from the Ca'Foscari University, Venice, Italy, in 2006 and a Ph.D. degree from the University of Twente, in 2010. Her main topics of interest include intrusion detection and traffic monitoring and modeling.

Aiko Pras is an Associate Professor in the Departments of Electrical Engineering and Computer Science at the University of Twente, the Netherlands where he is leading the Design and Analysis of Communication Systems Group. He received a Ph.D. degree for his thesis titled "Network Management Architectures". His research interests include network management technologies, network monitoring, measurements and security. He chairs the IFIP TC6 and is a steering committee member of several conferences, including IM/NOMS and CNSM, and series/associate editor of ComMag and IJNM.