A Passive Traffic Algorithm for Detecting Unavailable Periods in TCP Services

Iria Prieto, Mikel Izal, Eduardo Magaña and Daniel Morato

Public University of Navarre
Navarre, Spain
Email: iria.prieto, mikel.izal, eduardo.magana, daniel.morato @unavarra.com

***Abstract*—This paper presents a simple passive algorithm to monitor service availability. The algorithm is based on packet counting over a passive traffic trace of a population of clients accessing servers of interest. The major advantage of the algorithm is that it is passive and thus not invasive while usual monitor systems that can be found on Internet are active probing agents. The proposed system does not communicate to actual servers. It is easy to build as an online monitoring system with no big constraints in software or hardware. It does not relay on a distributed number of network placements for probing agents but works on a single network observing point near network edge. Initial proof of work of the algorithm is presented by studying the influence of different kinds of disruptions on packet level traffic and analyzing unavailability problems for popular servers at an academic network at Public University of Navarre.**

Keywords–Availability service; network; traffic

I. INTRODUCTION

As networks constantly evolve, network application servers are improved in software and hardware in order to cope with the growth of client's demand. In spite of this rapid development, sometimes, clients can not gain access to the servers due to communication problems or server saturation, due to flash crowd demands, human errors, updates, routing failures, etc.

Nowadays, even few minutes unavailability can be critical. For an enterprise offering products to clients through a web server, an interruption of this service means loss sales. Another example, which shows the threat of service interruption is the use of an antivirus update server. In case of banks, or other organization where security is a priority, an interruption of the update server entails possible infection problems.

As a result of the importance of being online all the time, several monitoring systems have been developed along the time. The target of these system is to detect as soon as possible when a network was experiencing problems. Typically, these kinds of systems are based on active probing. The main disadvantage of active probing is that sometimes, depending on the network condition, it will not be able to be applied. In high-loaded networks the methods of active probing, although it will try to be as simple as possible, is not always desired since an slight increment of the traffic may causes more packet delays and losses.

Nowadays, the vast majority of the services offered on Internet use as the application layer the Transport Control Protocol (TCP). The target of this work is to consider a passive method to detect service disruptions based on the study of the TCP packets. In [1] the algorithm was introduced showing the results for real services. In this work, the study has been extended to analyze how the unavailable periods caused by different problems can be distinguished using the proposal algorithm.

The paper is organized as follows. First of all, Section II describes the State of the art about the problem to face up. After that, the algorithm and configuration parameters are introduced in Section III. Section IV analyzes how the nature of the disruptions can affect the observed traffic and how it will be detected by applying the algorithm on the captured traffic. Section V describes the network scenario used to check the proposed algorithm. Section VI presents the results, comparing it to active detection of popular public services. Finally, Sections VII and VIII present conclusions and future work.

II. STATE OF THE ART

In order to detect when the clients of a network are not being able to successfully use a server application, a wide range of monitoring clients, such as Nagios [2], Zabbix [3], Cacti [4], Munin [5], have been developed. These systems warn the network administrator that a given server of interest is unavailable. These kind of systems work based on active probes, such as ICMP (Internet Control Message Protocol) ping or automatically requesting a server web page in case of monitoring HTTP (Hypertext Transfer Protocol) server. They are required to be installed and configured in monitoring client machines or at the server.

In cases where problems need to be detected at different client networks, at least one client has to be installed on each network. Otherwise, some problems will not be detected, like cases of routing problems in the path from clients to the servers of interest, if the monitoring client may use other route to reach the server.

As it is shown by Liu et al. [6], depending on the location of the system resources the application will achieve more effectiveness. Therefore, depending on where our monitoring clients will be located we would have only the vision of this location. Also, checking the configuration of these monitoring clients can be a problem for multi-tier system where the number of them will be high. In the literature, some papers explore how to face up testing the configuration in these scenes, [7]. Another problem of taking active measurement across an entire network is that for wide area networks it will not be scalable and some paths should be chosen and the rest of statistics inferred through predictive algorithms [8].

On the other hand, active probing can be a problem in high loaded systems or when monitoring third party servers, which may not react well to external continuous requests. Nowadays,

more and more enterprises rely on public services on Internet that would need to be monitored. In these cases, firewalls and intrusion detectors may deny probes or even ban future normal requests as response to continuous monitoring.

Configuring and using these kinds of distributed monitoring systems is not trivial as shown by different studies on how to approach the problem of monitoring for distributed programs [9]–[15].

Another disadvantage of active availability monitoring comes from cases where the clients access servers through proxy-caches. In that case, the monitoring client may be requesting a webpage from the server and receiving a response just because it is cached at the proxy system even if the final server is unreachable or has some problem. Thus, the active measurement does not actually check for server availability and other clients in different networks or served by different proxies may be experiencing access problems for the same server. In these cases the system would not detect the problem until the timeout of the cached object. This situation can be addressed by proxy configuration (may not be an option depending on proxy ownership) or crafting requests so they are not cached.

In some cases, due to misconfiguration or network issues, the monitoring client may experience problems to reach the server while actual client access is working, thus giving rise to false positive alerts to the network administrator. The cause of this failures may be things as memory problems or CPU or network overload of the monitoring client. This is often due to the fact that the same agent is probing a large number of servers. Therefore, the dimensioning of these clients has to be considered carefully.

Another issue to consider is the reaction time of the monitoring system. The minimum and maximum acceptable time for problem detection has to be decided. Longer times imply slower reaction, smaller times may generate higher overhead and interference to normal clients.

Currently, the majority of cloud services available on the Internet offer services over TCP protocol for communications with clients [16], [17]. It has been observed that some servers, due to overload, start refusing new TCP connections by answering with RST packets to clients for some time. In many cases the observed time of these kind of events is on the order of seconds, but usually less than half a minute. After this event the server recovers its normal behaviour and accepts again new clients. As stated before, even if it only lasts for seconds this problem may be critical for some businesses, causing user complaints and bad server reputation.

There have been proposals to cope with the downsides of active monitoring. Schatzmann et al. [18] proposed a method to detect temporary unreachability based on flow-level analysis by capturing traces from different routes. Although their method was able to work online the main disadvantage was the need to monitor in different points of a network. Besides, it should be taken into account that the setup of these kinds of measurements is not an easy task [19].

Others works have compared the advantages and disadvantages of using active probing or passive monitoring to service discovery [20]. This work was based on service discoveries and not in detecting disruptions so for passive traffic only observed the connection establishments and abrupt finalisation.

The goal in this work is the development of a simple online disruption detection method for TCP servers. It has been noticed how the disruptions affect packet level network traffic. This analysis of packet level traffic allows to distinguish when clients are having problems with one service. The proposed method avoids active measurement and works just by passive observing network traffic. This method is based on simple packet level counting such as the number of RST and data packet received. It does not require large amounts of memory or CPU power and it is able to detect problems for clients in different networks and for different services without using distributed agents. It will be shown that it is able to detect micro-access-failures with a configurable granularity in the reaction time.

III. Proposed algorithm

As stated in the introduction, the method is based on passive traffic capture. By capturing traffic close to the clients in a given network it will detect when some services will not be available to this community (in this work, the sample community will be the clients at Public University of Navarre network). The main target of the proposed algorithm is to find when a service disruption event has occurred, that means that the clients on the monitored network can not successfully use the service. The server may be down or may just be unreachable from this point due to network or some other problem. In any case this local unavailability is what the network administrator wants to detect more than the global server state. The objective is to detect availability problems, including the case where clients are able to reach the servers but not to use their services. To achieve this, a simple algorithm has been proposed, which does not require big hardware or software constraints.

The flow of traffic from the clients to the servers of interest is captured and some simple counters are evaluated every fixed time interval. The counters used are the number of data packets and reset packets sent by the full group of clients and target servers seen during a given (i.e., 5 seconds) time interval. Reset packets are TCP protocol packets with RST flag activated. They are used by a TCP endpoint to reject incoming connections and also whenever an abnormal packet is received by a TCP endpoint, to signal to the other side that it should abort the connection. The algorithm bases on the fact that a server sending just TCP RST packets and not any other valid packet to a group of clients during even a small period of time is an indicator of unavailability. Although sometimes it has been observed that the servers finish their connections in an unexpected way such as, sending RST packets to the clients after a client has sent a FIN packet, the algorithm will not show false positives since it will have a high probability that another client will be sending or receiving data packets in the same period. The mechanism consists on dividing time in fixed sized intervals. On every interval the number of packets seen from clients and servers are considered and related to previous interval. When a client sends packets to servers, which do not send anything back to it, a server issue is suspected.

If in subsequent seconds the servers keep silent but send reset packets the servers are confirmed as not working. Also, if the client keep sending packets and the servers keep silent it is confirmed as not working. The previous identification idea is built with two simple filters for every interval. On

each time interval, counters for clients and servers are updated in order to describe the situations explained before. On the side of the client, the counter is the number of packets sent to the servers, regardless if they are data packets or not, $packet\_cli$. On the other hand, on the side of the server, two counters are taken into account: The number of data bytes sent, $bytes\_servers$, and the number of packets with the reset flag activated, $reset\_packets$.

If during a given interval the counters show the client was sending packets but the servers did not send any data packet (even they may send reset packets) the result of the first filter for that time slot is 1. Also, the result is 1 when there are no packet sent by the client and the server only sends RST packets. That indicates the server is not answering requests. The second filter would be 1 whenever the result of the first filter of the interval being analazying is 1 and the result of the first filter of the previous interval was also 1. The process can be easily explained through two membership functions, like the ones used in fuzzy logic [21], which are applied in each period. Firstly, the used variables are defined:

- x= Number of client packets sent in an interval.
- y= Server Bytes sent by the servers in an interval.
- z= Number of RST packets sent by the servers in an interval.
- i = i^{th} Interval to be analazyed.
- $\psi_i(x, y, z)$= First pass of the compound filter applied in each interval i.
- $\varphi_i(\psi_i, \psi_{i-1})$= Second pass of the compound filter applied in each interval i, it takes into account the result of the first pass.

The two membership functions are described in equation 1.

$$\psi_i(x,y,z) = \begin{cases} 1 & if \quad ((x>0) \quad and \quad (y=0)) \quad or \\ & ((x=0) \quad and \quad (y=0) \quad and \\ & \quad (z>0)) \\ 0 & Otherwise \end{cases}$$

$$\varphi_i(\psi_i, \psi_{i-1}) = \begin{cases} 1 & if \quad (\psi_i = 1) \quad and \quad (\psi_{i-1} = 1) \\ 0 & Otherwise \end{cases} \tag{1}$$

Each period is labeled with the result of applying the two membership functions, $(\psi_i(x, y, z), \varphi_i(\psi_i, \psi_{i-1}))$. When both results are 1 an availability problem is considered for the duration of both intervals. We define the unavailability period since the first second of the interval labeled as $(1, 1)$ until the next interval labeled as $(0, 0)$. An example of the algorithm operation is shown in Table I.

In the second interval of Table I, there was one packet sent by a client, but there was no data sent to him by servers so the first flag is 1 and the second one is 0 because it was the first suspected interval. After this first interval, the servers, which belong to Hotmail service, sent 8 packets being all of them TCP RST packets. As there were only reset packet we label this second interval as $(1, 1)$. During the next 5 seconds the servers seem to have recovered because data packets from servers are seen again.

The example is a real case disruption interval detected for Hotmail server at the scenario. During that interval only reset packets were captured from servers and the packet trace was examined to show that servers were closing connections that had been inactive for more than 30 seconds.

These resets were not a response to any observed packet, so it seems reasonable that the server was experiencing problems and thus this is the kind of event the algorithm addresses. The main parameter of the algorithm is the time interval duration, that can be chosen by the network administrator depending on the desired reaction time. Smaller values will increase resolution and will detect microfailures but will also increase false positives.

From our experience, values between 5 and 15 seconds are recommended.

IV. Analazying the effect of the disruptions on the traffic observed

As it has been explained in the introduction, different problems can affect the client network. In this section, the effect of some of the most typical connection problems and how these problems are revealed on the traffic at packet level are studied.

For this purpose a testbed was set up, since the study of the nature of the problems in a real network would mean to have documented why each unavailable interval happened. In cases where the network suffered a problem close to the sniffer, those intervals could be labeled, but if the problems were outside our university network it is not possible to know exactly the cause.

Using the testbed unavailable periods between clients and servers were created intentionally. The duration of these conditions can be decided at first and, after some minutes where the clients will not able to gain normal access to the service on the server, the problem will be solved. These periods are labeled and compared to the values obtained by applying the proposed algorithm.

The following subsections described de testbed scenario, the effects of the disruptions on the traffic captured and how the algorithm detects these intervals.

A. Testbed scenario

The testbed consists of two client networks where several agents continuosly request a webpage to the server. Both networks are connected to the servers networks by two routers as seen in Figure 1. These routers perform also the role of sniffers. All the scenario was emulated using a completely virtual environment using Virtual Box. All the virtual networks operate at 10Mbps with a loss rate of 1%. The purpose of the loss rate is to make a more realistic environment. The whole scenario is shown in Figure 1.

As the amount of traffic can influence the time when the unavailable periods are detected by the algorithm, the requests are made following exponentially distributed interarrival times whose average load are different on both networks. The network 1 had an average load higher than the network 2, 8 Mbps and 3 Mbps, respectively (Table II).

One of the most frequent problems that networks suffer is due to link disconnections. Sometimes some link in the path between the client network and the server falls down by hardware problems on some interface or maybe the link

TABLE I. EXAMPLE OF THE DOUBLE CHECK ALGORITHM DEVELOPED FOR A INTERVAL OF THE DAY 2013/11/8 AND HOTMAIL SERVERS

Start	End	Bytes Serv (x)	RST Serv (z)	Packet Cli (y)	ψ	φ
9:24:55	9:25:00	7016	0	14473	0	0
9:25:00	9:25:05	0	0	1	1	0
9:25:05	9:25:10	0	8	0	1	1
9:25:10	9:25:15	1699	1	3288	0	0

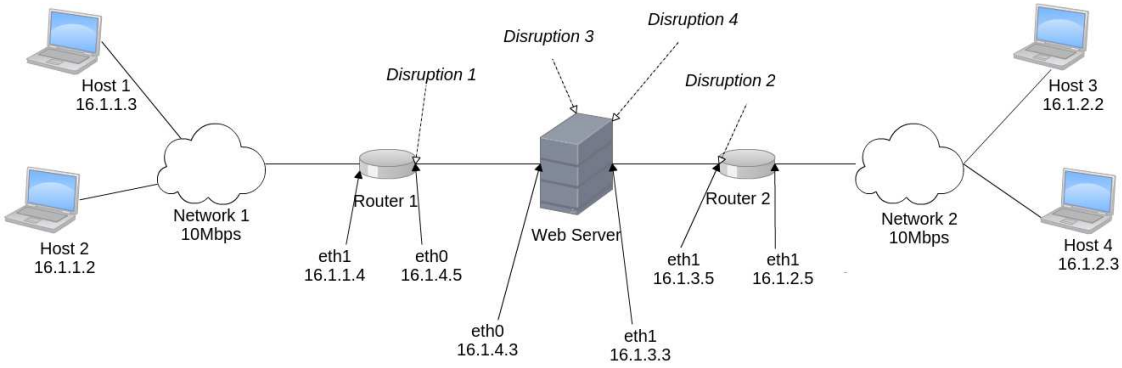

Figure 1. Testbed used to study the effect of different disruptions

TABLE II. CHARACTERISTIC OF THE NETWORKS

	Network 1	Network 2
Clients	2	2
Mean load	8 Mbps	3 Mbps
Losses (%)	1%	1%

is not disconnect but any packet is observed because the route is being changed. This unreachability situation is emulated in the testbed by switching off the outgoing interfaces on the respectively routers, at different times. After a short period the interfaces were switched on again. We will refer to these disruptions as Disruption 1, when the link on the router 1 was switched down but the rest were working properly, and Disruption 2 when the link on the second router was switched down.

Another usual service problem is due to servers being rebooted. Even if a service is being offered to clients based on a pool of servers, a particular client network can be affected by a single reboot on one server while the traffic of the clients is not addressed by another server. The reboot can be provoked by software updates, which require the reboot, by a hardware problem or simply by bad working. In the testbed, as a unique server was being emulating this error affected to both client networks. The reboot was made by a command and it is the time that took to recover the normal running wrote down; we will refer to this interruption as Disruption 3.

Finally, another common problem is that suddenly the actual service software falls down or does not respond. These kind of problems are caused, for example, by misconfigurations or changes on the services, for instance a port change, or by software problems. They are different from the previous one because in these cases the packets sent by the clients will be received on the server but it will not answer their requests. This problem was emulated by stopping the apache service on the server during some minutes. It was labeled as Disruption 4 and again it affected to both client networks.

All the disruptions described are emulated and it can be observed in the Table III. The estimated disruption time is the time, which was written down when the disruption started while, the estimated recovery time is the time that the service was available again. The estimated times between the disruptions and its recovery are approximated, in spite that these times were carefully taken, because they were written down by a human observer.

B. *Effects on the passive captured traffic*

The different problems emulated will not have the same effect on the observed traffic. In fact, in the cases of the disruptions 1 to 3, the packets sent by clients will not be able to reach the server, so it will not be received any TCP server packets. In the last case the clients will reach the server but they will not establish the TCP connections, so the server will likely answer with TCP Reset packets.

The traffic is analysed, at packet level, for each disruption near the time of the disruption was writte down. As the algorithm works on the TCP traffic, only this traffic was considered. In the two first cases, disruption 1 and 2, the clients suddenly did not receive any more packets sent from the server. In addition, it could be observed a lot of client connection attempts without any answer packet of the server. The attempts are showed by SYN packets sent from the clients. Both networks behaved identical for the disruptions 1 and 2 respectively. For this reason, only some packets, which belonged to network 1 are shown:

```
17:56:31 IP 16.1.1.2.37011 > 16.1.4.3.80: Flags [S],
17:56:33 IP 16.1.1.3.41460 > 16.1.4.3.80: Flags [S],
17:56:37 IP 16.1.1.2.37012 > 16.1.4.3.80: Flags [S],
17:56:41 IP 16.1.1.2.37013 > 16.1.4.3.80: Flags [S],
17:56:41 IP 16.1.1.3.41461 > 16.1.4.3.80: Flags [S],
17:56:42 IP 16.1.1.2.37014 > 16.1.4.3.80: Flags [S],
17:56:44IP 16.1.1.3.41462 > 16.1.4.3.80: Flags [S],
```

The reboot of a server may affects the traffic observed at packet level. Again, the traffic is studied close to the time,

TABLE III. PROGRAMMED DISRUPTIONS

Name	Type	Network 1	Network 2	Estimated disruption time	Estimated recovery time
Disruption 1	Network problem	Affected	Not Affected	17:45	17:47
Disruption 2	Network problem	Not Affected	Affected	17:56	17:59
Disruption 3	Reboot server	Affected	Affected	18:10	18:13
Disruption 4	Service not working	Affected	Affected	10:03	10:05

which was marked for the disruption 3 in the two networks. As in the case before, the clients suddenly stopped receiving packets from the server and the new requests did not find any answer from the server. During these intervals a lot of SYN packets were observed:

```
18:10:14 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:15 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:17 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.2.37138 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.2.37137 > 16.1.4.3.80: Flags [S],
18:10:21 IP 16.1.1.3.41681 > 16.1.4.3.80: Flags [S],
```

Observing the traffic at packet level of the previous disruption with the last one, an important difference can be observed. While in the previous cases the clients did not receive packets from the server, in the last one they were able to reach the server but as the service was not running this one answered them with Reset TCP packets. In the case of the last disruption, if we were probing the availability of the service using ping requests for instance, we would not realize of the problem since the server would answer. The effect on the traffic of the two networks was identical on both networks:

```
10:04:50 IP 16.1.4.3.80 > 16.1.1.3.59757: Flags [P.],
10:04:50 IP 16.1.1.3.59757 > 16.1.4.3.80: Flags [.],
10:04:50 IP 16.1.1.3.59757 > 16.1.4.3.80: Flags [R.],
10:04:55 IP 16.1.1.2.47925 > 16.1.4.3.80: Flags [S],
10:04:55 IP 16.1.4.3.80 > 16.1.1.2.47925: Flags [R.],
10:04:55 IP 16.1.1.3.59758 > 16.1.4.3.80: Flags [S],
10:04:55 IP 16.1.4.3.80 > 16.1.1.3.59758: Flags [R.],
10:04:57 IP 16.1.1.2.47926 > 16.1.4.3.80: Flags [S],
```

Therefore, once analysed all the cases, each of them will have little or any traffic towards the client from the server. The two functions used by the algorithm will adapt with the real scenarios since the server will not reply to clients or it will send reset packets until the problem is solved.

C. Detecting the unavailable periods

During the experiments shown on a real scenario, the time interval duration chosen was 10 seconds. The intention is that it is large enough to detect minute intervals with problems and at the same time, it is not too much small to rise false positives. Again, we used this value to define the durations of the intervals on the testbed. The target was to check if this amount of time was valid for the network with a high load and for the low one.

The unavailable periods showed by applying the algorithm to the sniffed traffic are shown in Figure 2. The periods corresponds quite accurately with the estimated time estimated (Table III). No false positive was showed up.

Comparing the full volume of traffic sent by the clients of each network with the server, again the difference between the different kind of the disruptions can be seen. For the disruptions where a link will be down for an interval of period, as the disruptions 1 and 2 any packet sent by the server will be observed but in contrast, a little amount of traffic sent by clients

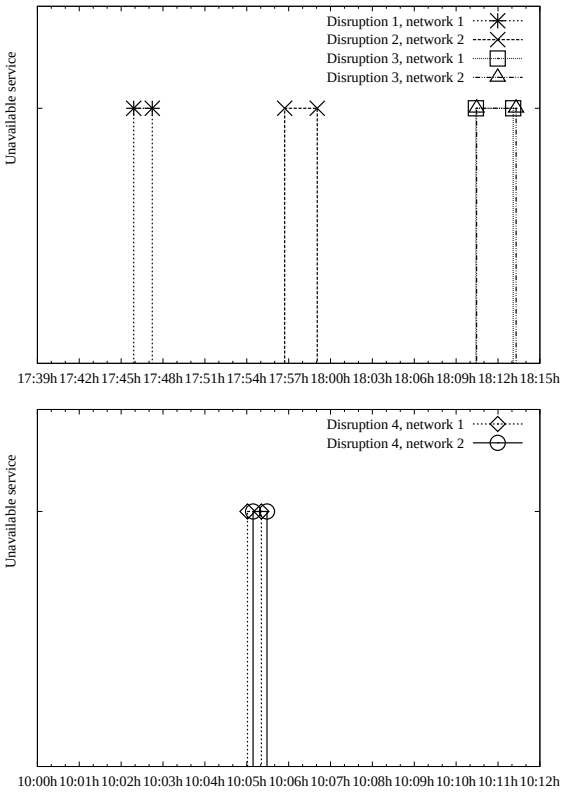

Figure 2. Unavailable periods detected on the testbed

trying to gain access to the server will be shown (Figure 3). In Figure 3, it is drawn the amount of Bps seen by the clients or servers addresses and the intervals of disruptions.

However, when the server is the one that suffers a problem, because it is being reboot or the service has fallen down, the effect on the traffic is quite different as it can be observed for the disruption 3 in Figure 2 and for the disruption 4 in Figure 4. While the first one the server will not send any packet, the last one will answer the requests with Reset packets. This means that in the last case it will observe traffic although it will be slightly lowest than before suffering the problem.

Another consequence of comparing the volume of traffic with the unavailable periods detected, is that in case of the second network the intervals are detected a little later than the network one, as it is shown in Figure 4. This is due to the clients on the second network do not request so frequently for the web page, so the problem is detected when they try to establish the connection with the server, which was later than the clients on the network 1. Similarly, the algorithm decides that the network 2 has recovered completely later than in the case of the network 1. On the second case, due to the clients

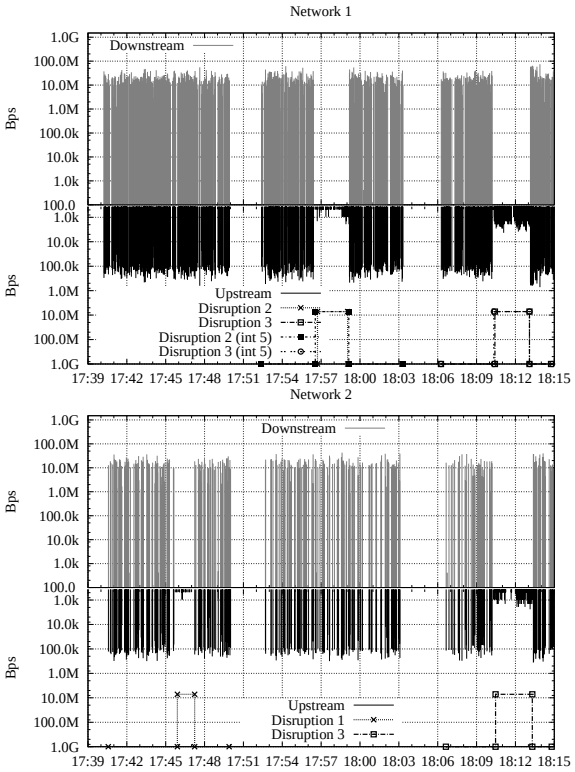

Figure 3. Bps for the use of Web service by the clients on the testbed at disruptions 1 to 3

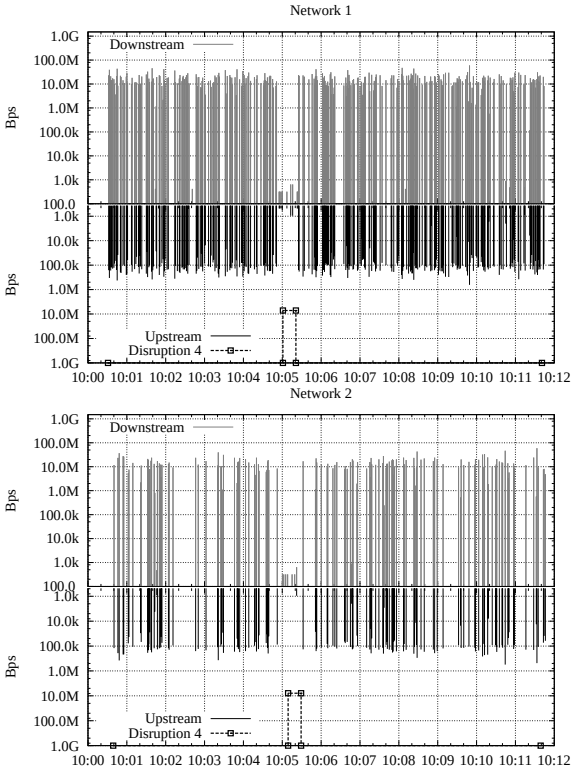

Figure 4. Bps for the use of Web service by the clients on the testbed at disruption 4

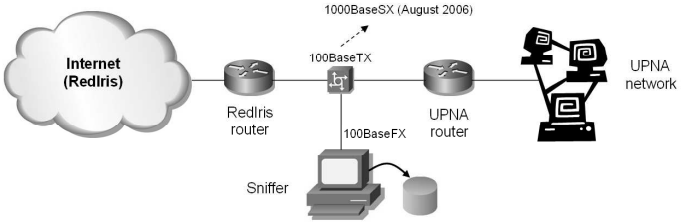

Figure 5. Traffic capturing from a University link

send less packets, the algorithm is not able to distinguish when the server is answering properly again. However, these difference between the unavailable periods detected for each network, are minimal, as can be seen in Table IV.

Another issue addressed in this section is the analysis of the impact of the time interval chosen in the algorithm. With a time interval of 10s it was checked as the results were quite accurate. Nevertheless the algorithm is applied using an interval smaller, exactly 5s. The main problem of using values for time intervals small is that the number of false positives could be increased since if the period is too small maybe the server had no time to reply at the same interval. However, in the testbed a period of 5 seconds was viable since the number of false positives was zero. The differences between using 10s and 5s were not big (Table IV). As the intervals were smaller the algorithm detected problems some seconds earlier. Moreover, the recovery times for each disruption were earlier detected also.

In spite of the fact that a value of 5 seconds behaves a little better in this scenario, in real scenarios where maybe the traffic is bursty or almost non existing at some hours, for instance at nights, we strongly recommend values of 10s instead of 5s. This value will decrease the possible numbers of false positives and still will be able to detect short intervals, in which clients were having problems to use the services properly.

V. NETWORK SCENARIO

The algorithm has been developed and tested, detecting availability problems of public internet servers for clients at Public University of Navarre. Captured data comes from author's research group infrastructure who has access to a sniffer with its own software between university main access and academic internet provider (Rediris) as seen in Figure 5. The group has an ongoing packet trace collection campaign since 2004 providing 1Gbps traces from the access of an academic community.

In this work, results are presented from captured data of the week of November 7th to 11th, 2013, checking the availability of popular servers at this community such as Facebook, Yahoo, BBC and Hotmail. In order to compare the algorithm against an active monitor (like Nagios [2]), a very basic probing system is implemented. The active monitor tests the availability of selected servers by requesting the site `favicon.ico` file. This file provides an icon to be displayed at browser window and is widely used by web servers. The program requests the favicon file every 5 seconds for every service considered in the experiment and thus provides a ground truth value of availability for comparison purposes.

The active requests are performed from a desktop computer at the university network. The number of servers probed is

TABLE IV. COMPARISON OF THE UNAVAILABLE PERIODS DETECTED APPLYING DIFFERENT TIME INTERVALS, 10S AND 5S

Network	Disruption	Estimated disruption period	Unavailable interval (t=10s)	Unavailable interval (t=5)
Network 1	2	17:56 - 17:59	17:56:33 - 17:59:03	17:56:23 - 17:59:03
	3	18:10 - 18:13	18:10:15 - 18:13:05	18:10:10 - 18:13:00
	4	10:03 - 10:05	10:04:51 - 10:04:51	10:04:46 - 10:05:16
Network 2	1	17:45 - 17:47	17:45:44 - 17:47:14	17:45:39 - 17:47:09
	3	18:10 - 18:13	18:10:18 - 18:13:18	18:10:13 - 18:13:13
	4	10:03 - 10:05	10:04:59 - 10:05:29	10:04:54 - 10:05:24

not very large so the probing computer is not loaded and no request failures can be attributed to machine overloading. The proposed passive algorithm operates on traces obtained at network edge as seen above. It is evaluated offline for the results of this work, but may be easily programmed as an online system.

As servers used are very popular, there are other sources of availability information that were considered. Several web pages provide down times and real time user complaints of public servers but usually this information has not enough time granularity to test less than ten minute disruption events.

VI. RESULTS

In this section, results of unavailability detection with a week trace of traffic are presented (November 7th to 11th, 2013). Public servers addressed are: "Yahoo", "Facebook", "BBC", "Hotmail" and also a local newspaper "Diario de Navarra", which are frequently visited by users at the University. Those servers, except the local newspaper, are also used by a large mass of users around the world and they are served by a pool of different IP addresses. They are probably distributed over large server farms or content distribution networks.

But even if those farms are probably designed to balance load and support peaks of demand, sometimes, the clients of the University are not able to reach these services.

Experiments with the basic active monitor that request `favicon.ico` file show the results in Table V for the servers under analysis. Figure 6 shows the events of unavailability with time. The service with more suspected intervals detected was Hotmail.

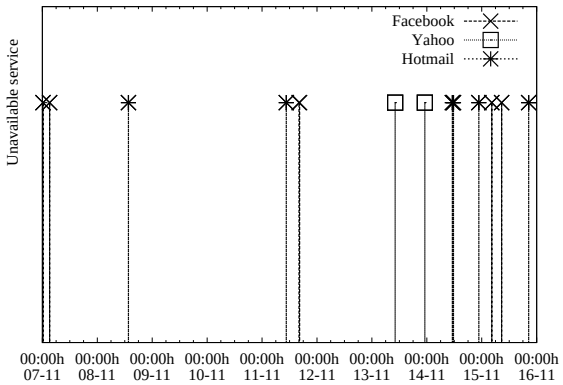

Figure 6. Events of time where the favicon was not be obtained

To test the proposed algorithm the packet trace of a full day is processed and the algorithm is applied on the traffic.

TABLE V. UNAVAILABLE SERVICE INTERVALS DETECTED BY REQUESTING THE FAVICON

Start	End	Day	Service
0:15:49	00:16:58	07/11/2013	Facebook
3:10:01	03:10:06	07/11/2013	Facebook
13:36:35	13:36:51	08/11/2013	Hotmail
16:08:12	16:25:40	11/11/2013	Facebook
10:34:59	10:35:21	11/11/2013	Hotmail
10:10:23	10:10:29	13/11/2013	Yahoo
23:08:21	23:08:27	13/11/2013	Yahoo
11:08:54	11:09:06	14/11/2013	Hotmail
11:39:18	11:39:36	14/11/2013	Hotmail
22:43:00	22:43:05	14/11/2013	Hotmail
8:40:31	08:40:44	15/11/2013	Facebook
20:30:03	20:30:13	15/11/2013	Hotmail
4:22:29	04:22:34	15/11/2013	Facebook

The rest of the results are for day 08/11/2013 although other days are similar.

First, the network traffic is filtered to select packets from the probing agent and selected servers of interest. Although this is not the target of this work, addresses of these servers have first to be identified. To solve this, the payload of packets is examined to search for these server names in HTTP requests.

Both methods active and proposed algorithm show some unavailability issues for the Hotmail service, see Figure 7. The plot shows the volume of traffic from client machine to Hotmail as well as the time events identified by the passive algorithm and active favicon requester. Both algorithms identified the same event. Packet level examination of the event showed a single connection, which suffered an unexpected reset from the server. The comparison also revealed that the time difference is due to the monitor client, which was not NTP synchronized as the passive sniffer is. This shows a point to take into account in a distributed monitoring system when monitor clients are distributed time synchronization plays a critical role. The passive sniffer has a unique clock source so the problem of synchronization is simplified.

Packet level analysis of previous event showed the dialog of the packets below. The *x.x.x.x* represents the IP of the client and the *y.y.y.y* the IP of a Hotmail server. After the connection is established, the client sent the request through a push packet of 176 bytes. Usually, after this packet was sent by the client the server answered with the `favicon.ico`. However, in this case the server sent an ACK packet without data and after some seconds, around 11, closed the connection sending a reset packet. This kind of behaviour is unexpected and during these seconds the client would have noticed a malfunction using the service.

```
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: S
13:36:02 IP y.y.y.y.http > x.x.x.x.59133: S
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: . ack 1
13:36:02 IP x.x.x.x.59133 > y.y.y.y.http: P 176
```

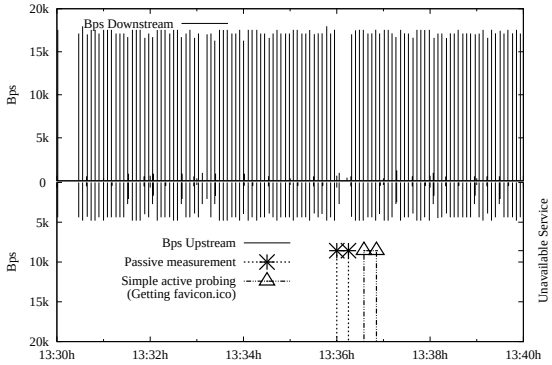

Figure 7. Intervals of time in, which the monitoring client had problems for day Nov 8th

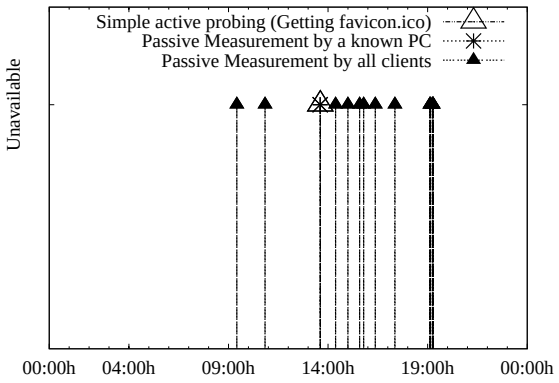

Figure 8. Comparison of the events of unavailable Hotmail service detected from request client for day Nov 8th

```
13:36:02 IP y.y.y.y.http > x.x.x.x.59133: . ack 177
13:36:13 IP y.y.y.y.http > x.x.x.x.59133: R 1 ack 177
```

Others cases of non-typical reset packets were also observed in the intervals of unavailability studied. In many cases, before a server went down it did not answer to the clients, and after some time it started to send them reset packets to clients since they did not reconignize the previous established connections.

A. Comparison between active probing vs passive analysis unavailability detection method

The total traffic from all the clients using services that previously have been identified to have unavailability periods is analysed. The objective is to distinguish the periods of time where all the users experience service access problems of the periods of time of isolated problems for individual clients.

To achieve this for each service, all the requested servers are joined together to study if in some period the clients were active but the servers were not working properly. The proposed algorithm is applied to the aggregated network traffic. The algorithm is configured using the IP addresses of all the servers as an unique service to be monitored and a time interval duration of 5 seconds. The unavailability events detected are shown in Figure 8. Interestingly, there are more unavailable periods detected that way than the issues detected by using the favicon requester alone.

The previous event, which was observed through the favicon.ico requests and observing the traffic for a single client who requests the favicon.ico (Figure 7), now is not labeled as problematic because at the same time other clients were able to use Hotmail. This interval was a problem of one server giving service to an individual client but it was not a problem of availability for the observed server since other clients were using the same service (other IP addresses of the same service). Thus, this is revealed as a false positive warning that shows the risk of using only the monitoring client as a method to detect service failures.

But this experiment shows other more important fact. By using the service as an aggregation of individual IP address of servers we are able to identify some unavailability intervals of a few seconds where the clients were suffering access problems but were not detected by active monitoring clients. These

TABLE VI. UNAVAILABLE HOTMAIL SERVICE INTERVALS

Start	End
09:25:05	09:25:15
10:50:00	10:50:10
14:22:40	14:22:50
14:59:40	14:59:50
15:35:00	15:35:10
15:47:15	15:47:25
16:22:05	16:22:15
17:21:10	17:21:30
19:06:50	19:07:00
19:07:20	19:07:35
19:13:35	19:14:05
19:16:10	19:16:30

periods were not observed by the monitoring client because the favicon.ico was served by a proxy cache. Table VI shows all the final disruption events detected.

These periods correspond to the sending of unexpected resets by the severs to the clients. The study of the traffic did not reveal any previously wrong behaviour of the clients, which could provoke the send of resets packets by the server. During this seconds, suddenly one or more servers decide to abort the established connections with one or more different clients. As the duration of the intervals were short, these were not actually critical disruptions since the next connections were established. In case that this kind of periods had to be ignored it may be done by just increasing the time interval duration, for example, to 10 seconds.

TABLE VII. UNAVAILABLE HOTMAIL SERVICE INTERVALS

Start	End
15:34:50	15:35:10
19:13:40	19:14:10

Table VII shows events detected from the same traffic by using an interval duration of 10 seconds. Two cases detected correspond to two intervals of 20 and 30 seconds. During this time there were only reset packets sent from the servers to clients, which have previously completed a connection establishment. Other intervals of 10 seconds are not detected since as the service recovered faster the reset packets sent in order to abort client connections felt inside the same interval

as the data packets sent by the servers once that they had recovered. Also, the intervals may not coincide exactly due to interval and event synchronization. The maximum error will be given by the minimum interval of time considered. For example, in the examples presented in this paper, the time of the disruption would be more or less 5 seconds since the interval is said, or 10 seconds when this is the used time interval.

We have checked also the rest of services whose some intervals were detected as unavailable by the monitoring client. The study of the traffic did not reveal any period with problems, there were not any interval of time where the server did not answer to the clients. The periods showed by the monitoring client were due to problems of the own client with the proxy cache or a particular server but not with the service.

B. Traffic profiling of the requested services

As a sanity check the full volume of traffic from the scenery network to the servers is observed to check that the amount of traffic was significant. Traffic for the 8th of November to Hotmail service is shown in Figure 9. The first image in the figure shows the whole day traffic, while the second one is a zoom to some of the main intervals detected. Hotmail is shown since it is the service with more disruption events detected by the algorithms. The intervals of unavailability detected by the algorithm are drawn also. The first plot shows a full day of traffic and the second one zooms to 1 hour around the previous discussed event.

It can be seen that the amount of traffic suggests the service is working and the gap around 15:35:10 is clearly visible. After these period without traffic the service seems to reestablish normally, creating a traffic peak after the detected problem that reaches almost 5 MBps.

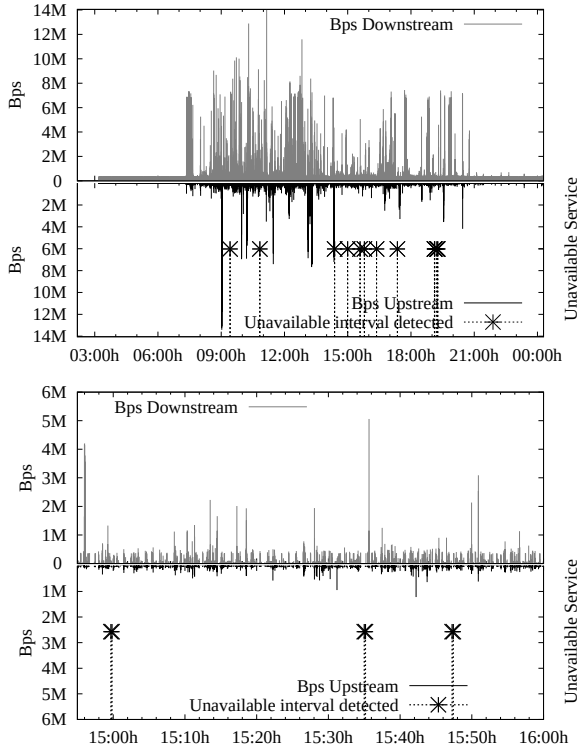

Figure 9. Bps for the use of Hotmail service by the university community

VII. Conclusions

In this paper, a simple algorithm to detect periods of unavailability services has been presented. It is based only on passive capture of traffic.

Although there are more service monitoring software available, to the authors best knowledge, they are based on active probing systems. Active monitoring presents some disadvantages, which may discourage network administrators of its use in scenarios where the impact of the monitoring needs to be minimized. First, because it requires to check if a service is available it would imply to make periodically requests to different servers. In some scenarios, like high loaded servers or monitoring third party services it is not possible to make these requests as frequently as needed, in order to avoid overhead or security alarms. Apart from that, the probing requests should be chosen carefully in order to avoid problems with proxy caches, which could give the impression that the service is working properly while other clients would not be able to use the service. Another problem is the difficulty to select a location for monitoring clients in multiple subnet scenarios. In these cases, at least a pair of clients should be placed in each subnet in order to detect possible problems inside. Moreover, every client should be clock synchronized in order to report coherent times with the rest of monitoring clients.

As the proposed model is passive and based only on the study of packet counts between servers and clients it will not interfere with the traffic on the network. Therefore, any problem of interference, monitoring client overload, network problems with measured server availability is avoided.

Although the algorithm is based on the expected behaviour of the transfers between client and servers seen at packet level, it has been shown how different kinds of disruptions, as for instance, an unexpected server reboot has an influence on the network traffic at packet level captured at client's side.

Another advantage of using the proposed model is that it is based in a single location. That means the measure is not dependent on the location of multiple monitoring agents. The network administrator has just to select an appropriate passive observing location, where it can see the traffic between the population of clients to monitor and the servers of interest. This is a much simpler decision that can be typically solved by placing the sniffer at organization's network's edge.

VIII. Future work

Currently, we are working to extend the algorithm to detect service failures without focusing on specific servers, just by analyzing sniffed traffic and applying the current algorithm to every connection seen. In this manner, the algorithm can work as an service anomaly detection system that warns administrator of service issues. This is useful in large organizations that may not have a clear list of services accessed by users but, nevertheless, need to react to service unavailability problems.

An improvement that can be implemented in order to reduce the number of false positives, is to use the two membership functions described in the algorithm to apply some method of fuzzy logic.

ACKNOWLEDGMENT

This work was supported by the Spanish Ministry of Science and Innovation through the research project INSTINCT (TEC-2010-21178-C02-01). Also, the authors want to thank Public University of Navarra for funding through PIF grant.

REFERENCES

[1] I. Prieto, M. Izal, E. Magana, and D. Morato, "Detecting disruption periods on tcp servers with passive packet traffic analysis," in SOFTENG 2015, The First International Conference on Advances and Trends in Software Engineering, April 2015, pp. 34–40.

[2] "NAGIOS, a commercial-grade network flow data analysis solution," 2009-2015. [Online]. Available: http://www.nagios.com/ [accessed: 2015-01-30]

[3] "ZABBIX, the ultimate enterprise-level software designed for monitoring availability and performance of it infrastructure components," 2001-2014. [Online]. Available: http://www.zabbix.com [accessed: 2015-02-02]

[4] "CACTI, a complete network graphing solution." 2004-2012. [Online]. Available: http://www.cacti.net/ [accessed: 2014-12-29]

[5] "MUNIN, networked resource monitoring tool," 2003-2013. [Online]. Available: http://munin-monitoring.org/ [accessed: 2015-01-15]

[6] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web applications using queueing predictor," in Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, 2006, pp. 106–114.

[7] D.-J. Lan, P. N. Liu, J. Hou, M. Ye, and L. Liu, "Service-enabled automatic framework for testing and tuning multi-tier system," in e-Business Engineering, 2008. ICEBE '08. IEEE International Conference on, 2008, pp. 79–86.

[8] D. Chua, E. Kolaczyk, and M. Crovella, "Efficient monitoring of end-to-end network properties," in INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 3, 2005, pp. 1701–1711.

[9] Y. Park, "Systems monitoring using petri nets," in Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, vol. 4, 1997, pp. 3245–3248.

[10] C. H. Choi, M. G. Choi, and S. D. Kim, "CSMonitor: a visual client/server monitor for corba-based distributed applications," in Software Engineering Conference, 1998. Proceedings. 1998 Asia Pacific, 1998, pp. 338–345.

[11] C. Steigner, J. Wilke, and I. Wulff, "Integrated performance monitoring of client/server software," in Universal Multiservice Networks, 2000. ECUMN 2000. 1st European Conference on, 2000, pp. 395–402.

[12] G. Song, "The study and design of network traffic monitoring based on socket," in Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on, 2012, pp. 845–848.

[13] G. Fang, Z. Deng, and H. Ma, "Network traffic monitoring based on mining frequent patterns," in Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on, vol. 7, 2009, pp. 571–575.

[14] A. Tachibana, S. Ano, and M. Tsuru, "Selecting measurement paths for efficient network monitoring and diagnosis under operational constraints," in Intelligent Networking and Collaborative Systems (INCoS), 2011 Third International Conference on, 2011, pp. 621–626.

[15] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, vol. 1, 2003, pp. 134–144.

[16] K. Claffy, G. Miller, and K. Thompson, "The nature of the beast: Recent traffic measurements from an Internet backbone," in International Networking Conference (INET) '98. Geneva, Switzerland: The Internet Society, Jul 1998, pp. 1–1.

[17] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," in Proceedings of the 2011 31st International Conference on Distributed Computing Systems, ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 310–321. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2011.27

[18] D. Schatzmann, S. Leinen, J. Kgel, and W. Mhlbauer, "FACT: Flow-based approach for connectivity tracking," in Passive and Active Measurement, ser. Lecture Notes in Computer Science, N. Spring and G. Riley, Eds. Springer Berlin Heidelberg, 2011, vol. 6579, pp. 214–223.

[19] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," vol. PP, no. 99, 2014, pp. 1–1.

[20] G. Bartlett, J. Heidemann, and C. Papadopoulos, "Understanding passive and active service discovery," in Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 57–70. [Online]. Available: http://doi.acm.org/10.1145/1298306.1298314

[21] G. Klir and B. Yuan, Fuzzy sets and fuzzy logic. Prentice Hall New Jersey, 1995, vol. 4.