Detecting Disruption Periods on TCP Servers with Passive Packet Traffic Analysis

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 communicates 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 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.

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 [1], Zabbix [2], Cacti [3], Munin [4], 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. [5] depending of 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 scenarios, [6]. Another problem of taking active measurement across an entire network is that for wide networks it will not be scalable and some paths should be chosen and the rest of statistics inferred through predictive algorithms [7].

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, [8]–[14].

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 [15], [16]. 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 accept 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. [17] 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 [18].

The goal in this work is the development of a simple online disruption detection method for TCP servers. This method avoids active measurement and work just by passive observing network traffic. The proposed 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.

The paper is organized as follows. First of all, the algorithm and configuration parameters are introduced. Section III describes the network scenario used to check the proposed algorithm. Section IV presents the results, comparing it to active detection of popular public services. Finally Sections V and VI present conclusions and future work.

II. 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 filter 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 [19], 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 the 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 where 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.

III. 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 on Figure 1. 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 [1]), 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.

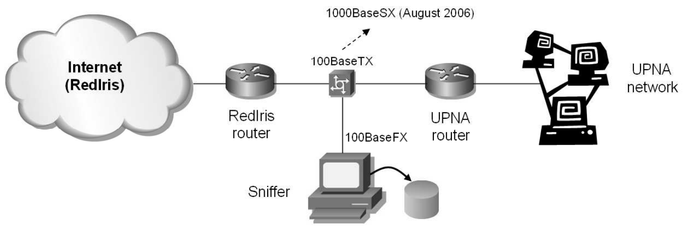

Figure 1. Traffic capturing from a University link

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

IV. 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 II for the servers under analysis. Figure 2 shows the events of unavailability with time. The service with more suspected intervals detected was Hotmail.

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

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

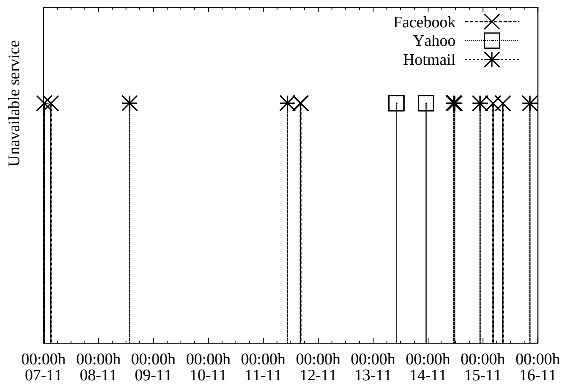

Figure 2. Events of time where the favicon was not be obtained

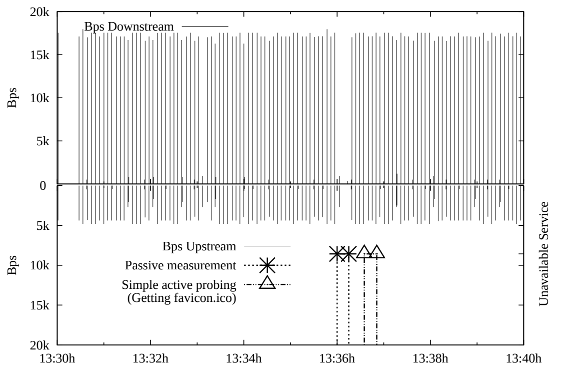

Figure 3. Intervals of time in which the monitoring client had problems for day Nov 8th

TABLE II. 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

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 3. 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
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 analyzed. 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 4. Interestingly there are more unavailable

TABLE III. 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 detected that way than the issues detected by using the favicon requester alone.

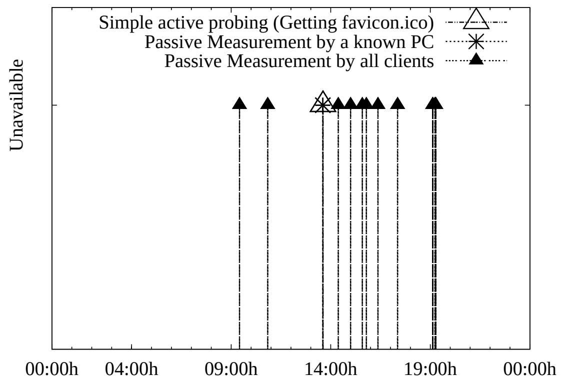

Figure 4. Comparison of the events of unavailable Hotmail service detected from request client for day Nov 8th

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 3, 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 show 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 periods were not observed by the monitoring client because the favicon.ico was served by a proxy cache. Table III 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 IV. UNAVAILABLE HOTMAIL SERVICE INTERVALS

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

Table IV 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 5. 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 suggest the service is working and the gap around 15:35:10 is clearly visible. After these period without traffic the service seem to reestablish normally, creating a traffic peak after the detected problem that reaches almost 5 MBps.

V. 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

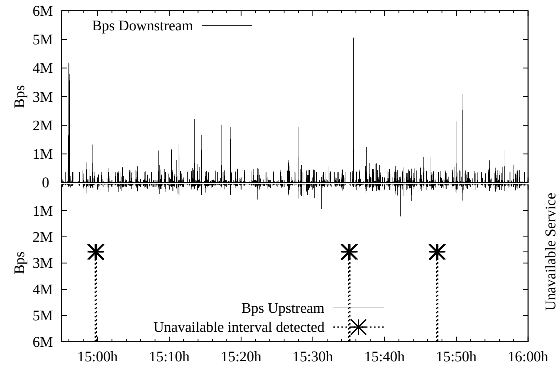

Figure 5. Bps for the use of Hotmail service by the university community

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 be 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.

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 have 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.

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

[2] "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]

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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

[17] 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.

[18] 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.

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