Distributed PowerShell Load Generator (D-PLG): A new tool for dynamically generating network traffic

Paul Jordan*, and Chip Van Patten†
Air Force Institute of Technology
Dayton, Ohio 45433
Email: *paul.jordan@afit.edu, †donald.vanpatten@afit.edu

***Abstract*—Failure in cloud infrastructure is a relatively common occurrence due to an array of issues. This problem is often masked by the use of excessively redundant systems in virtual environments and storage area networks, but can still cause service interruption. Further, properly reasoning about failure reduces the need for so much infrastructure reduncancy and leads to more efficient systems. Fortunately, machine learning techniques have been presented to predict failure [10]. Unfortunately, much of this work has gone unused as seen in [4] due to the manual and arduous maintenance these techniques require and general lack of labeled training data. Recently, a framework has been developed to automate the training of prediction algorithms but has only been tested on one system in [4]. In order to generalize the approach a few key functions must be performed. One of these functions is load generation. Unfortunately, a valid load generator has not been developed for a Microsoft Windows active directory environment. In this paper we introduce and detail a tool that we have developed to help make the implementation of this new framework possible in a Microsoft domain, we present data generated by our tool to demonstrate its efficacy, and finish with several extensions and applications for our tool.**

I. INTRODUCTION

There are many approaches to generating realistic traffic or capturing live traffic for replay. Unfortunately, none of these approaches are capable of independently generating full-stack network traffic without naively replaying previously recorded traffic, which is of limited utility in simulating a production environment where realistic data transmission is not necessarily confined to what's already been observed. This research is the result of an ongoing attempt to generalize the framework developed by Irrera et al. in [4] called the Adaptive Failure Prediction (AFP) framework.

AFP automates the process of retraining a failure prediction algorithm after an underlying system change by placing the target system under load before injecting software faults to accelerate failure. Our target system is a Microsoft Windows active directory domain services server and, as a result, we need to generate a full-stack authenticated session in order to sufficiently load the service. To that end, we have developed a tool for generating this type of traffic and introduce it in this paper. Further, we demonstrate the validity of our tool and demonstrate that our technique is generalizable and can leverage the AFP framework in order capture network transactions of arbitrary arity between unbounded network components with dynamic volume, variety, veracity, and velocity.

II. RELATED WORK

Many software tools exist for generating network traffic. Generally, these tools are classified into three categories: application-level, flow-level, and packet-level generators [2] [16]. Application-level generators emulate traffic produced by applications on a network, flow-level generators replicate actual traffic using statistical modeling, and packet-level generators create and inject packets into the network. Network traffic generators are further classified as open- or closed-loop. Open-loop generators use a packet arrival model for packet timing, whereas closed-loop generators wait for a response to a sent request prior to sending the next request [15]. Unfortunately, as far as we can tell, none of the tools available generate the necessary interaction with a deployed Microsoft Windows active directory environment necessary to facilitate the implementation of the AFP framework. Active directory implements the Kerberos authentication protocol in Windows domains and due to its cryptographic nature cannot be tested against replayed or random traffic; rather, a sequence of valid and invalid requests and responses are necessary to stress test this framework. Indeed, multi-step "handshakes" are necessary for rich service delivery, and this capability is not realized by the current tools with any degree of modularity or extensibility.

A brief review of the traffic generators we considered when researching this problem follows. The Distributed Internet Traffic Generator (D-ITG) [2] is, as its name implies, a distributed traffic generator capable of performing application, flow, and packet-level generation using both open- and closed-loop operations – sessions are initiated at specific time intervals and, within each session, new requests are not sent prior to receiving a response to the previous request.

NTG [16] is an application-level, distributed network traffic generator which is both open- and closed-loop. A key feature of NTG is that it interacts with existing network services, specifically web, mail, and multimedia servers.

Swing [14] is a flow-level, closed-loop traffic generator that observes live network traffic, extracts distributions from the traffic, and generates new traffic in a manner consistent with the observed traffic. Additional network generation tool reviews can be found in [2], [16].

While all of these tools are, in general, sufficient for generating traffic in a network, they do not generate full-stack,

two-way authentication that we need in order to sufficiently load the active directory domain services service. Due to the nature of the authentication protocol, it is not possible to realistically generate this traffic without valid user credentials.

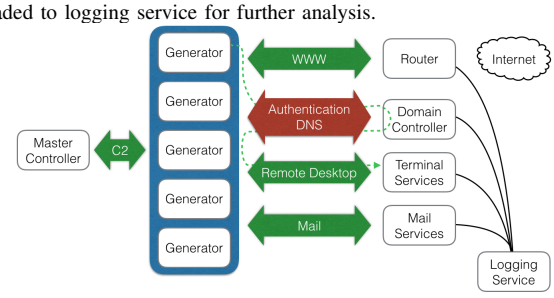
A final tool worth mentioning, while not a network traffic generator, is Microsoft's Active Directory Performance Testing Tool (ADTest) [8]. Limited documentation is available, but in [1], [9], [11], [12] we find that ADTest tests the ability of Microsoft 2003/2008/2012 Active Directory Lightweight Directory Services (AD LDS) servers to add organization units and users, and make various changes to Active Directory to aid in developing requirements for an AD LDS deployment. It is important to note that Microsoft no longer supports this tool [9]. Also of importance, ADTest is not capable of testing other services that rely on active directory domain services for authentication (e.g. RDP, SMB, etc), nor can it be extended to do so, and is therefore insufficient for our goals.

III. DISTRIBUTED POWERSHELL LOAD GENERATOR (D-PLG)

We present D-PLG, a new tool for the generation of realistic network traffic in a Microsoft Windows domain for the purposes of software testing or load generation. D-PLG can be classified as an application-level closed loop traffic generator and is a basic Windows PowerShell script that uses native PowerShell cmdlets for all of its functionality ensuring that we generate the most realistic traffic possible without overburdening the client machines used for generating load. D-PLG offers what we have not seen in other traffic generation products or tools by making actual service requests and producing actual challenges and responses for Windows authentication protocols.

We chose the Windows PowerShell environment because it affords us a tremendous amount of power and flexibility to generate traffic that would actually be generated by a users interaction with network services since the same software applications and libraries are used. As a result, D-PLG does require the use of client machines, but in this work we intend to show that generating this type of realistic traffic is possible by utilizing only a small number of machines, or without producing a noticeable burden on in-use client machines.

Fig. 1. How each type of traffic that is generated is routed. Log events are offloaded to logging service for further analysis.



In general, the intended architecture can be seen in Figure 1. D-PLG is most effective if used by a few client machines during idle downtimes but is developed in such a way that a user can still use a machine that is generating load, but may notice degraded performance depending upon how much traffic that particular client is being asked to generate. D-PLG is currently designed to run from one central location, asking a configurable list of clients to produce traffic for a fixed period of time.

In its present form, D-PLG is comprised of three modules capable of generating full-stack web requests, Microsoft remote desktop protocol, Microsoft server message block (SMB) file sharing, and all associated authentication traffic. An intended byproduct of all of this traffic is domain name system (DNS) requests. An important part of any active directory domain is DNS and as a result, no load generator would be complete without performing DNS lookups.

The rest of this section outlines each of the three modules currently implemented as well as our plans for future modules.

A. Web Browsing

D-PLG is capable of generating full-stack web requests and presently simulates an actual user browsing. We take advantage of the 'Invoke-WebRequest' PowerShell cmdlet which upon completion returns an object representing the full document object model (DOM). The return of this object allows us to programmatically simulate random browsing within a returned page. As a result, our tool generates very realistic web traffic against a web server. This functionality is very different from the functionality implemented in many of the existing tools that only generate one-way transmission of the web request. As a result, this module allows users of our tool to generate realistic load against web servers and potentially automate realistic web application testing.

The web browser was created with minimal effort as our approach to D-PLG emphasizes rapid generation of modern internet-based interactions. In future versions, we plan to implement more dynamic web browsing to facilitate the use of D-PLG as an automated web application testing tool. Since the entire DOM is returned, it is possible and relatively simple to programmatically complete web forms, and submit REST API calls in only a few lines of PowerShell code.

B. Remote Desktop Protocol

Remote Desktop Protocol (RDP) is a simple protocol that allows the sharing and remote control of a Windows desktop environment. We included this module to generate more authentication traffic with our active directory domain services server as well as place load on our remote desktop services server. Applications for this module could include network infrastructure capacity and server sizing planning. The module takes advantage of a modified third party cmdlet [3] which invokes a call to the native windows remote desktop application (mstsc.exe). Our modification only tells the cmdlet not to present a window as to avoid interrupting an individual who may be using the computer at the time of load generation.

Currently, the RDP module makes a full-stack remote desktop connection with an RDP server without producing a

window which can allow us to take advantage of clients in active states. The script then sleeps for a few seconds and then closes the connection. In future versions, we would like to implement some sort of actual interaction with the RDP server like file upload or application use. We based this functionality on a tool previously developed by Microsoft [7] but is unfortunately no longer maintained as evidenced here [13].

C. Server Message Block (SMB) File Sharing

D-PLG implements an SMB file sharing module that connects to a local or remote share, creates a file in the share, fills that file with random ASCII data, saves the file, deletes the file, and finally deletes the share. By performing this sequence of operations, we have ensured that the full-stack SMB file sharing requests are utilized thus causing the domain services server to authenticate the transaction and the file sharing services server to process the data being uploaded. This simple module could additionally be used to ensure a file server is live before beginning more complex operations.

Like the previous module, the SMB module was rapidly built due to the flexibility of our framework and implemented in only fourteen lines of code. In future versions, we plan to implement a variable amount of upload data or allow the user to select his or her own file. By allowing the user to upload a custom file, this module could be used to test application aware firewall rules to ensure certain types of files are or are not allowed to traverse a network.

D. Future Modules

We have already implemented many core active directory domain services as a proof of concept, but would like to point out how easy additional services would be to implement in our script. For example, simple message transfer protocol (SMTP) traffic could easily be implemented using the 'Send-MailMessage' cmdlet. Additionally, the 'Out-Printer' cmdlet would allow for the sending of realistic full-stack network printer traffic.

These modules demonstrate our platforms extensibility and are representative of sophisticated network interactions that are necessary to create a performant load generator for the tableau of modern networking services. To facilitate future development, we have published D-PLG in its current form under the MIT license on GitHub [5].

IV. METHODOLOGY

This section is split into two subsections. In the first, we describe in detail our virtual environment. In the following section, we detail the design of our experiments which utilized our virtual environment.

A. Virtual Environment

The virtual environment was hosted on two VMWare ESXi 5.5 hypervisors each with two 2.6 GHz AMD Opteron 4180 (6 cores each) CPUs and 64 GB memory. The individual virtual machines are detailed in Figures 2, and 3. D-PLG uses cmdlets that did not exist until PowerShell version 3.0 so each of the Microsoft (MS) Windows computers had the MS Windows Management Framework version 4.5 installed. The installation of this framework also necessitated the installation of the MS .NET Framework version 4.5. In an enterprise environment these software frameworks would more than likely already be installed as they are part of the service pack updates that have since been released by Microsoft.

Fig. 2. Hypervisor 1.

Qty.	Role	Operating System	CPU / Mem.
1	DC	Win. Server 2008	2 / 2 GB
5	Client	Win. 7	1 / 512 MB

Fig. 3. Hypervisor 2.

Qty.	Role	Operating System	CPU / Mem.
1	RDP	Win. Server 2008	1 / 4 GB
1	Log	Ubuntu 14.04 LTS	1 / 1 GB

After installing the requisite software, each client was added to the domain and required a few minor modifications. First, D-PLG creates remote 'PSSessions' on each client machine and then invokes the cmdlets we have assembled to generate the load we desired. In order for this to happen, the credentials of the controller must be delegated so that they may be used to make the connections through the PSSession. This delegation is done very simply through the PowerShell cmdlet 'Enable-WSManCredSSP'. The final modification was for convenience; we placed a copy of the scripts to be executed remotely on the desktop of the Administrator user.

The domain controller had two MS Windows Server roles enabled: active directory domain services, and domain name service (DNS). One domain administrator account was used for command and control traffic, and individual user accounts were created and used for RDP and simple authentication traffic. The RDP server only had one MS Windows Server role enabled: remote desktop services.

The Ubuntu server was deployed and used as a central log repository for analyzing load on the domain controller and RDP server. The default rsyslog application was simply configured to accept incoming connections and then the rsyslog Windows agent was installed on the domain controller and RDP server.

D-PLG is divided into two scripts. The first is the 'Local-LoadGen' script and is placed on each client computer. We note here that placing the script on the each client computer may not be ideal in a production environment and this step could easily be automated when the controller runs. Further, upon completion, the script could be removed in a single PowerShell command. The second script 'RunLoadSim' is designed to act as a command and control element that connects to each client and executes the 'LocalLoadGen' script as an asynchronous job. In our experiments, the command and control script was executed from our RDP server.

B. Experiment Design

We have designed two experiments to test and demonstrate the efficacy of our tool and detail them here. In both of the following tests, we ran D-PLG five times, where each execution consisted of five minutes of traffic generation within our virtual environment. The domain controller was sized based on Microsoft's community recommendation for up to fifteen thousand users in [6]. We used ESXi to collect the relevant data in the form of packet captures at the virtual switchports of one client machine, the terminal server, and the domain controller. Further data collected came from the ESXi performance data. After each round of our tests, the performance data were exported from each of the hypervisors on the terminal server, one client, and the domain controller. In these data, CPU utilization, memory utilization, disk operations, and network traffic are reported on twenty second intervals. Finally, as previously stated, the rsyslog Windows Agent was used to forward the logs from the domain controller and RDP server to an Ubuntu server. These log entries were then split into pieces that corresponded with each round of the tests.

The first question we wanted to answer was, how much traffic can a PowerShell script really generate, and is it enough to sufficiently load an enterprise domain controller? We designed the first experiment to answer that question. To maximize the amount of traffic and subsequent load generated, the client machines were only configured to make a single request. To do this, the 'RunLoadSim' was only tasked to perform a basic authentication request to the domain controller. To prevent over-burdening the client machines, we found that the highest frequency at which these events could be created and handled was 10 per second. It should be noted here that the client machines we used were significantly less powerful than average desktop computers typically found in an enterprise environment. In timing this authentication event, we were able to determine that on average it took 20 milliseconds so it is not difficult to imagine a scenario in which the client machines could be configured to produce 50 requests per second.

In the second experiment we wanted to determine how much load could be produced without having a significant effect on the resources available to each client machine. We configured the clients to utilize each of the modules that are currently implemented in D-PLG. In each round of the test the client machines looped continuously making an authentication request to the domain controller, a full RDP connection, an SMB share connection, a web request to a randomly selected URL, and finally a web request to a URL randomly selected from the page returned by the first request. The loop was configured to run twice per second, however due to the high latency of the web requests, we did not expect the client to make that many requests every second of the test. This configuration choice was made to ensure maximum utilization when possible.

V. EXPERIMENTAL RESULTS

In this section we detail the results and data collected after conducting the tests described in Section IV-B. To answer the quantity question, we explored the number of packets produced per second as well as CPU utilization, memory utilization, log events, and network operations on the domain controller. In this first round of tests, the domain controller reported an average of 56,291 log events over each five minute test or approximately 187 log events per second. In addition, we captured an average of 6,267 packets per second between the five tests. Figure 4 shows the distribution on the number of packets sent and received by the domain controller for each test and tells us that the load was consistently high throughout the each test. On the client side, as we predicted, the load was also relatively high as seen in Figure 6.

Fig. 4. How many packets per second were sent or received by the domain controller across all five rounds of the first test. In each test, we captured approximately 1.8 million packets.

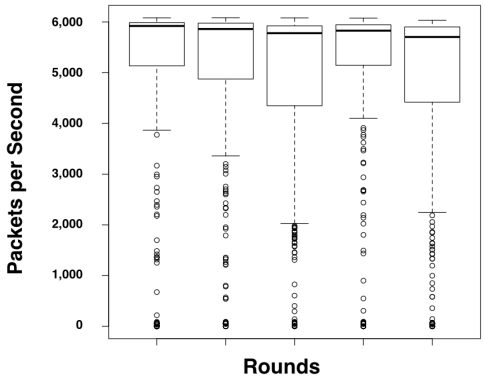


Fig. 5. How many packets per second were sent or received by one of the clients across all five rounds of the first test.

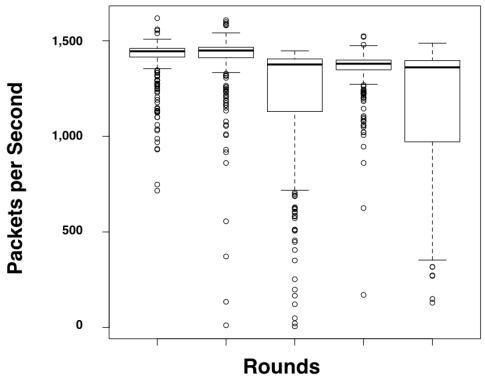


Fig. 6. Client CPU and memory utilization during the first test.

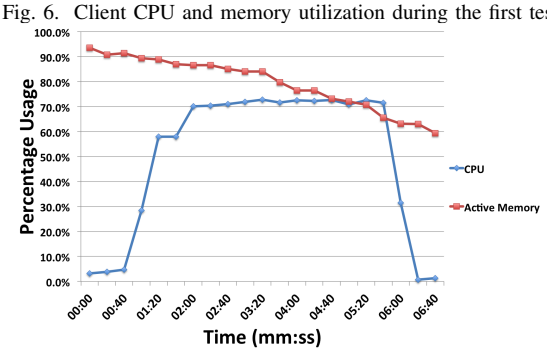
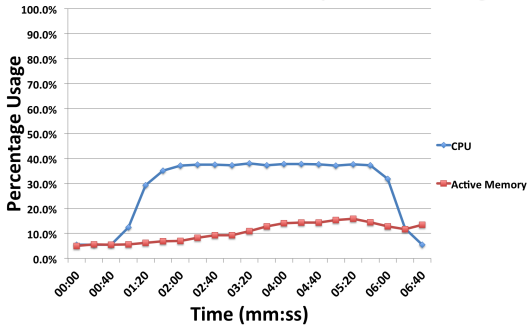


Fig. 7. Domain controller CPU and memory utilization during the first test.



These results validate our hypothesis. The load generated against the domain controller was exactly in-line with the amount of load it should be expected to endure during peak usage per the Microsoft community recommendations for sizing. Unfortunately in this case, the client machines would likely not have been usable during the test. Fortunately, because we only needed five low-end machines to produce this load over a relatively short period of time, a simple solution to this problem would be to purchase five inexpensive desktop computers for this purpose, or conduct testing during an idle downtime.

In the second test, we were trying to find out if the client machines could produce a sufficient amount of realistic traffic without being over burdened so that they could be used to generate load even as individuals use them. To answer this question we examined CPU and memory utilization, and packets transmitted per second with respect to the client. The average number of packets generated over the five minute tests was 5,499 and the remainder of the data can be seen in Figure 8. We also examined these same data with respect to the domain controller and RDP server seen in Figures 9, and 10 respectively.

Fig. 8. Client CPU and memory utilization during the second test.

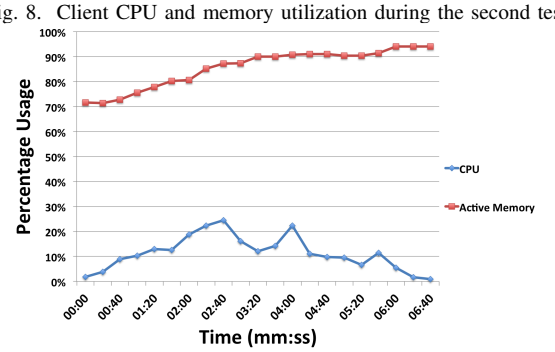


Fig. 9. Domain controller CPU and memory utilization during the second test.

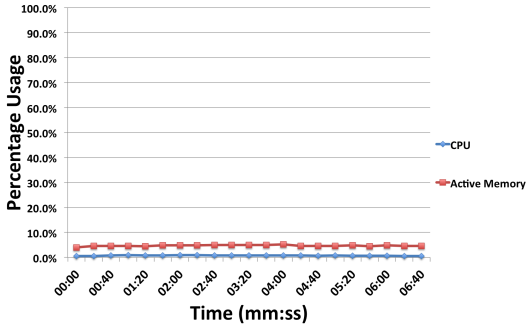
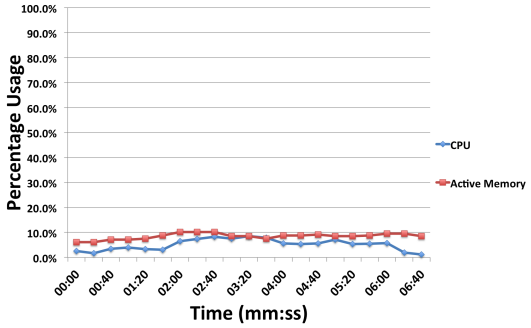


Fig. 10. RDP server CPU and memory utilization during the second test.



While the number of packets was relatively high, we don't believe that these results demonstrate a sufficient amount of load on either the domain controller or RDP server. We suspect that this is due to the majority of the time spent during the test retrieving web-pages. As a result, if a proxy server or application aware firewall is the target of this load generation, we believe that this level of traffic would be sufficient for that purpose. We also believe that these results show that a client computer asked to generate traffic could still be used during a test. In future work, we plan to test this infrastructure with non-blocking web-requests so that more load can be placed against the local services while web requests are being processed externally. Alternatively, more client machines could be used during the test.

In general, the results we observed lead us to believe that our tool can be extended and used under any circumstance where

dynamic network transactions are required between unbounded network components. Further, we believe that these results show that our tool will allow us to leverage the AFP framework in our future work.

VI. FUTURE WORK

There are many opportunities for improvement in D-PLG. If D-PLG is to be used to simulate realistic network traffic patterns that would normally be generated by humans, much work would have to be done to balance the kinds of requests that get made. For example, a typical user might log in, browse the web for a few minutes, check his or her e-mail, then maybe send an e-mail. Currently, D-PLG is extremely predictable with respect to what kind of request it will make next. The framework can be made a lot more relevant with programmatic generation schemes such as REGEX-based pattern generation or training of input validity via machine learning.

More configuration options could be added like the depth of a browser simulation or having the browser simulate filling out web forms using configurable data. D-PLG could also allow for finer grain control over the SMB module allowing users to select a file or specify the size of the randomly generated file. We believe that most of these configuration options would be implemented in a straightforward way.

Finally, as previously discussed, other modules which take advantage of more of the native Windows PowerShell cmdlets like 'Send-MailMessage' and 'Output-Printer' could be implemented with relative ease.

VII. CONCLUSION

Based on the results of our tests, we believe that D-PLG is capable of providing sufficient load of network services in a Microsoft Windows enterprise domain. Our five clients were able to generate fifteen thousand authentication requests over a five minute time period which was very near the limit that our domain controller could handle based on the Microsoft community recommendations. Further, since we based our configuration on these recommendations, we also believe that our experiment will scale for larger networks. We believe that the use of client machines to produce this load is negligible and have proven that it can be done centrally, with only a few machines, without placing a significant burden on those client machines, and without installing any additional software by use of the native Windows PowerShell cmdlets. Finally, we believe that if necessary and client machines are used during idle down times, significant load can be generated by D-PLG. We believe that this level of load is sufficient for our purposes of conducting further research in the area of online failure prediction and could easily be used for other applications.

There are many established needs for having network traffic and load generators, and we believe that the results presented in this paper suggest that D-PLG can serve many of the same purposes. For example, in cybersecurity training events, traffic generators are used to simulate real traffic to mask malicious traffic. Other uses include equipment sizing, stress testing, and software testing. We believe that D-PLG can fill these needs as well and in general can be naturally extended and used under any circumstance where dynamic network transactions are required between unbounded network components.

REFERENCES

[1] P Bijaoui. *Microsoft Exchange Server 2003 Scalability with SP1 and SP2*. HP Technologies. Elsevier Science, 2011.

[2] A. Botta, A. Dainotti, and A. Pescapé. A tool for the generation of realistic network workload for emerging networking scenarios. *Computer Networks*, 56(15):3531–3547, 2012.

[3] J. Brasser. Script connect-mstsc - open rdp session with credentials, 2015.

[4] I. Irrera, M. Vieira, and J. Duraes. Adaptive Failure Prediction for Computer Systems: A Framework and a Case Study. *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, pages 142–149, 2015.

[5] P. Jordan and D. Van Patten. Afp-dc - github, 2016. https://github.com/paullj1/afp-dc/tree/master/Load.

[6] S Makbulolu and G Geelen. Capacity planning for active directory domain services, 2012.

[7] Microsoft. Download remote desktop load simulation tools from official microsoft download center, 2009.

[8] Microsoft. Download Active Directory Performance Testing Tool (ADTest.exe) from Official Microsoft Download Center, 2012.

[9] Mark Morowczynski. How To Use the Active Directory Performance Testing Tool on Windows Server 2012 — Ask Premier Field Engineering (PFE) Platforms, 2014.

[10] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, March 2010.

[11] H. Suyanto and M. Tiwari. Windows 2008 AD LDS Load Testing using ADTEST - Part 1 - TechNet Articles, 2010.

[12] H. Suyanto and M. Tiwari. Windows 2008 AD LDS Load Testing using ADTEST - Part 2 - TechNet Articles, 2010.

[13] A. Szeto. Debug assertion failed: sockcore.cpp, line 623, 2012.

[14] K. Vishwanath and A. Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009.

[15] M. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. Smith. Tmix: A Tool for Generating Realistic TCP Application Workloads in ns-2. *ACM SIGCOMM Computer Communication Review*, 36(3):67–76, 2006.

[16] P. Zach, M. Pokorny, and A. Motycka. Design of Software Network Traffic Generator. *Recent Advances in Circuits, Systems, Telecommunications and Control*, pages 244–251, 2013.