A Scalable Architecture for Network Traffic Forensics

Viliam Letavay
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: iletavay@fit.vutbr.cz

Jan Pluskal
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: ipluskal@fit.vutbr.cz

Ondřej Ryšavý
Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: rysavy@fit.vutbr.cz

Abstract—**The availability of high-speed Internet enables new opportunities for various cybercrime activities. Security administrators and Law Enforcement Agency (LEA) officers call for powerful tools capable of providing network communication analysis of an enormous amount of network traffic as well as capable of analyzing an incomplete network data. Big data technologies were considered to implement tools for capturing, processing and storing packet traces representing network communication. Often, these systems are resource intensive requiring a significant amount of memory, computing power, and disk space. The presented paper describes a novel approach to real-time network traffic processing implemented in a distributed environment. The key difference to most existing systems is that the system is based on a light-weight actor model. The whole processing pipeline is represented in terms of actor nodes that can run in parallel. Also, the actor-model offers a solution that is highly configurable and scalable. The preliminary evaluation of a prototype implementation supports these general statements.**

Keywords–Network forensic analysis; Network traffic processing; Actor model.

I. Introduction

The expansion of computer networks and Internet availability opens new opportunities for cybercrime activities and increases the number of security incidents associated with network applications. The number of connected devices grows, and traffic speed increases. Security administrators and Law Enforcement Agency (LEA) officers call for powerful tools that enable them to extract useful information from network communication [1]. The network forensics that is responsible for capturing, collecting and network data analyzing is becoming more important [2].

In the forensic investigation, the network traffic is continuously captured from multiple sources. The captured network data has a form of packet traces that have to be processed and analyzed up to the application layer. The network forensic tool has to decode protocols at different network layers of the Transmission Control Protocol/Internet Protocol (TCP/IP) model and various encapsulations. For LEA officers, interesting information lies in application messages, such as instant messaging, emails, voice, localizable information, documents, pictures, etc. The form and relevance of extracted artifacts may differ from case to case. Often, communication is encrypted. In this case, meta-data can be the only piece of information available. In all cases, the network forensic processing system has to be able to extract artifacts from the network traffic reliably, even if the packet capture is corrupted, for instance, some connections are incomplete, packets are malformed, or chunks of packets were not recorded because of capturing device issues.

The amount of data that needs to be processed to extract evidence from the network communication depends on the kind of a case that is investigated but usually gets large. It is very difficult to decode, extract and store the immense mass of information for further processing. We propose a distributed network forensic framework based on the *actor model* that is computation effective and capable of linear scalability. Scalable properties of *actor model* design for network forensics are promising, as shown by the Visibility Across Space and Time (VAST) platform [3]. Similarly to VAST, our solution provides real-time data ingestion and interactive data analysis, but in addition to VAST, we consider the full artifact extraction up to the application layer. Although it requires more computation resources, we demonstrate that it can still be achieved in a more straightforward and less resource consuming environment compared to Apache Hadoop technology, which is the norm for big data processing.

In Section II, we describe tools used by network forensics practitioners. Section III addresses issues faced by investigators and our proposed solution, which architecture is broadly discussed in Section IV. Section V evaluates preliminary performance results, and Section VI concludes the paper.

II. Background & Related Work

Network forensics is a process that identifies, captures and analyzes network traffic. Network forensic techniques are used by several network forensic frameworks [4]–[9] and tools intended for intrusion detection (Zeek, VAST, Moloch) [10]–[12], network security monitoring (Microsoft Network Monitor, TShark, Wireshark, tcpdump) [13]–[16], and network forensic investigation for LEAs (Netfox Detective, PyFlag, NetworkMiner, EnCase, XPlico) [17]–[21]. Commonly available forensics tools are implemented either as a classic desktop or command line application or a traditional client-server solution.

To overcome the limitations of traditional tools, we propose to use distributed computing. The models for distributed processing [22][23] are more suitable for real-time network forensic analysis from multiple sources, such as logs and captured communication. The models are based on an agent system, where numerous agents perform the collection task. The extracted information is sent to the forensic network server and analyzed on this single node [24] only. The *forensic server* is the bottleneck that has to process all the data. To avoid this bottleneck, the Google Rapid Response (GRR) [25], a live forensic system, utilizes a cluster of servers. The system deploys agents running on users' computers that provide access to forensic information, e.g., remote raw disk and memory access. Processing of forensic data is done as flows. Each flow is maintained on the server. Server nodes run workers that process the active flows. Adding more server nodes enables to run more workers and thus it is possible to handle more clients simultaneously.

Elimination of bottlenecks in the architecture offers scalability and improved reliability. The *actor model* [26] is one of the attractive solutions that address the problem elegantly and efficiently. It comes with a separate unit called an *actor*. Actors execute independently and in parallel. They communicate with each other asynchronously via message passing, and their state is otherwise immutable. Actors are capable of spawning new actors, forming a parent-child relationship, allowing the creation of a tree-like structure of actors. Actor's current behavior determines how it processes the incoming messages. Every actor in an actor system is uniquely identified by an address which other actors use as destinations of the messages they want to send out. This address can identify actors at the local machine and also the ones at the remote machines, allowing easy means of communication between nodes of a cluster. Compared to another similar programming model, the *Communicating Sequential Processes* (CSP) [27], elementary units of computation – processes are anonymous and communicate with each other via established communication channels. The actor system is the key enabler for the VAST system [3]. In VAST, actors implement importing, archiving, indexing and exporting processed data. Actors live in nodes that map to system processes. The system scales by creating more nodes either on the single machine or a cluster of computers.

Moloch is another tool, worth to mention, that uses principles of distributed computing for massive scale network traffic monitoring, full packet capturing and indexing [12]. Moloch system consists of sensors that capture the communication and Elasticsearch database that is a distributed search and analytics engine. The system scales by adding new nodes running Elasticsearch instances.

III. Problem Statement and Solution

Our goal is to design and create a system capable of long-term, high-speed, real-time network traffic filtering and processing up to the application layer. The software solution should be scalable and hardware independent. To achieve this, we have to deal with the challenges elaborated in the rest of this section.

A. Architectural Design

How to create a system for packet filtering and analysis of communication that can identify application protocols, gets forensics artifacts and searches through them?

Network forensics is a tedious work that strictly relies on completeness and precision of all undertaken steps to gain a piece of a puzzle that fits together as a shred of evidence. Considering the current speeds of regular users' home network connection(s), a comprehensive classical analysis on a single machine would require enormous computation resources. Try to imagine, that each network packet would be analyzed by many protocol dissectors with a goal to extract, for example, an acknowledgment of email delivery. To achieve this goal, with optimal computational resources, we must revisit currently utilized methods and redesign them to work in a distributed environment which brings new challenges to architecture design, application of algorithms, data synchronization, and so on.

B. Scalability on Commodity Hardware

How can the solution be scalable and hardware independent despite the hardware limitations?

Let us consider this imaginary demonstration. The math is simple, one computer with $1\,\mathrm{Gbps}$ Network Interface Card (NIC) that has a relatively simple task to capture traffic during full line load would be required to write to a disk under the constant speed of $1000\mathrm{Mbps} \approx 125\,\mathrm{MB/s}$. Our system has to guarantee that no data loss occurs during the capture. A suspect can simultaneously download and upload data which means that the monitoring device cannot have only one $1 * 1\,\mathrm{Gbps}$ NIC, but it needs $2 * 1\,\mathrm{Gbps}$ cards, one for uplink, one for downlink. Thus, the required speed of continuous disk writing would be $2 * 125\,\mathrm{MB/s} \approx 250\,\mathrm{MB/s}$. Now, if the requirement is to store the communication for one day, the disk capacity has to be $250\,\mathrm{MB/s} * 86\,400\,\mathrm{s} \approx 21.6\,\mathrm{TB}$. This is achievable with commodity hardware, e.g., $2 * 12\,\mathrm{TB}$ drives with Redundant Array of Inexpensive Disks (RAID) 0 or $4 * 12\,\mathrm{TB}$ with RAID 1+0 — assuming higher write/read speed than $250\,\mathrm{MB/s}$. However, what if only one day is not enough? For a typical forensic case, capturing period spawns through weeks or months.

From our previous experiments, we know that a single computation node is limited and commodity hardware is hardly sufficient to perform all required operations in real-time and over long periods. Separation of frames into a conversation which requires a dissection of the network protocols up to the application layer, which speed is roughly $300\,\mathrm{Mbps}$ [28, pp. 45-51] is not sufficient. On the other hand, we are confident that the application created and optimized for this singular purpose can do the processing faster and breach the $1\,\mathrm{Gbps}$ line speed. Nevertheless, we do not believe that a single machine solution with commodity hardware is capable of doing overall analysis and extraction of information from the application layer. We have to design our solution as a distributed system across multiple machines.

C. Overall Performance

What scalability and acceleration of data processing can be achieved?

The proposed solution is based on the actor model. Each actor represents an independent processing unit. The communication between actors is managed by messaging. Actors have no shared state; thus all of them can work in parallel. If actors run on the same node, the message passing has little additional overhead compared to a function call or a loop. However, if actors scale over multiple nodes, messages need to be serialized. This process introduces latency and consumes part of the processing power. The scalability of the actor model is linear [3].

IV. Architectural Design

Incomplete data provided by unreliable traffic interception can lead to inaccurate results; some information may be lost, some fabricated by reconstruction process [29]. Keeping the above facts in mind, the processing cannot strictly follow Requests for Comments (RFCs) and behave like a *kernel* network stack implementation, but it has to incorporate several heuristics. For example, to fill missing gaps in data, and to consider these fillings during application protocol processing, or never to join multiple frames into a single conversation unless it passes more advanced heuristic-based checks. Network forensic tools that we have worked with do mostly respect RFCs and thus may produce misleading results, as shown by Matousek et al. [29].

We propose a distributed architecture composed of commodity hardware that will be capable of linear scalability, and capable of efficient resource utilization. The overall architecture is shown in Figure 1.

At the top level, we have divided the entire process into the two main stages:

- Data preprocessing — The reconstruction of conversations at the application layer (L7) of the TCP/IP model. This process consists of consecutive segregation of captured communication into the internet (L3) and transport (L4) conversations and deploying a reassembling heuristics [29] to recognize individual L7 conversations inside a parent L4 conversations and to reassemble their payloads with respect to data loss, reordering or duplication. Every L7 conversation holds information about the source and destination endpoints (IP addresses, ports), timestamps, type of transport protocol (UDP or TCP) and reassembled payloads of exchanged application messages.
- Data analysis — The analysis of each application conversation consists of the identification of the application protocol, and extraction of application events, e.g., visited web pages, exchanged emails, domain name queries, etc., with proper application protocol dissector that yields sets of forensic artifacts.

A. Data Prepossessing

The First stage is executed on a set of independent *Reassembler* nodes. These reconstruct L7 conversations from the stream of captured packets which can originate from *Packet Capture (PCAP) files* or can be captured from *the live network interface*.

In the most common use-case, we have one source stream (i.e., one PCAP file) which we want to analyze. Therefore, to utilize multiple *Reassembler* instances, we have to split packets from this stream into smaller sub-streams, which will be distributed among available *Reassembler* instances. For this split, we cannot use a naive method such as *Round Robin*, because *Reassembler* nodes operate independently of each other and to fully reconstruct L7 conversation a particular *Reassembler* has to obtain all the pieces of that particular L7 conversation. In case we would use *Round Robin*, a situation could occur when half the packets from one L7 conversation would end up in one *Reassembler* node and the second half in another; both nodes would have incomplete data and none of them would be able to reconstruct the conversation entirely.

Our proposed solution to this problem is another type of node – *L4 Load Balancer*, which will be positioned in front of the *Reassembler* nodes and which, as a name suggests, distributes packets based on their associations to L4 conversations each of which can consist of multiple L7 conversations. *L4 Load Balancer* extracts source and destination IP addresses and ports and transport protocol from each packet of the source stream and uses this information to decide to which instance from the available *Reassemblers* should it forward to. This way, all packets of a particular L7 conversation will always be forwarded to only one *Reassembler* instance.

Reassemblers build a tree-like structure of L3 and L4 conversations which are represented by the actors. Each received packet is first forwarded to an appropriate L3 conversation actor, which in turn forwards it further down to an appropriate L4 conversation actor which reassembles L7 conversations. This segregation of packets into the individual L4 conversations before actual L7 conversation reassembling is required, as implemented reassembling heuristics expect to operate on packets from a single L4 conversation at the time. The use of a hierarchical actor design allows us to perform independent portions of the processing in parallel and also to easily implement management strategies such as passing management messages to a particular L3 conversation actor and its children L4 conversation actors. The reconstructed L7 conversations are stored in a distributed database, ready to be retrieved in the second stage of the execution.

B. Data Analysis

In the *second stage*, a subset of reconstructed L7 conversations is retrieved from the distributed database and delivered to the *Application protocol dissector* nodes. For every L7 conversation, *Application protocol dissector* nodes identify the used application protocol and use a proper dissector module dedicated to the processing of a single application protocol, such as Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP) or Domain Name System (DNS), to extract application protocol messages from this L7 conversation. Obtained data are stored back into the distributed database. Processing of application messages is under normal circumstances possible only with unencrypted network communication. From Secure Sockets Layer/Transport Layer Security (SSL/TLS) communication which encapsulates application protocols, such as HTTP, we can extract only unencrypted portions of this data such as the server's cryptographic certificate. Possible ways to decrypt and subsequently, parse the SSL/TLS communication is to own a private key of a given SSL/TLS server or to deploy an SSL/TLS intercepting proxy [30].

V. Preliminary Evaluation

Our prototype implementation is based on C# actor system library *Akka.NET*. For testing and performance benchmarking, we have implemented two modes of operation:

1) *Offline* — isolated execution which combines the functionality of a single *L4 Load Balancer* and *Reassembler* node inside a single system's process. No inter-actor message serialization is therefore required.
2) *Online* — distributed execution spanning across multiple cluster nodes. The inter-actor message serialization is required as messages destined to remote

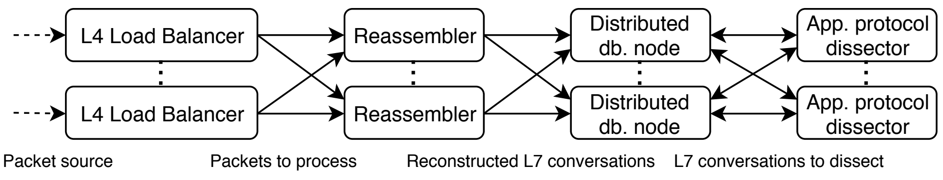

Figure 1: Architecture diagram showing the proposed system nodes with information flow between them.

actors (nodes) have to leave an originating system's process and be transmitted over a computer network in a serialized format. This introduces additional latency and performance overhead.

Additionally, for proof-of-concept benchmarking, the functionality of *Application protocol dissector* nodes was included inside *Reassembler* nodes to eliminate distributed database as a middleman between them. In the following measurements, we focus on a raw network capture's processing performance of the so-far naive implementation. Currently, our prototype implementation supports the dissection of two application protocols (DNS and HTTP).

We have measured the preliminary performance of the implementation on two different hardware configurations:

- *Workstation* — Intel i7-5930K 4.3 GHz, 12 cores, 64 GB RAM, 512 GB SSD
- *Mini-cluster* — 4x servers with Intel Xeon E5520, 2.26 GHz, 8 cores, 48 GB RAM, 1 TB SSD, 1 Gbps network

We used a public data set of M57-Patents Scenario [31], that consists of real-world data captured over a month. We merged all network traces into one PCAP file of roughly 4.8 GB and 5,707,845 frames. One large PCAP file simulates our use-case of streamed-in communication that needs to be load-balanced from a single node.

We started with measurements in an *offline* mode on a single machine, firstly with a PCAP file parsing operation and incrementally added consequent operations and measured processing speeds, as Table 1 describes. Preliminary evaluation suggests that the *raw speed* of roughly 3.8 Gbps, for PCAP file reading and packet parsing is sufficient. The process of reconstructing L7 conversations that segregates IP flows by packet source and destination IP addresses, ports and transport protocol type with additional heuristics [29], that also reassembles TCP/UDP streams, is computationally heavier, reaching "only" 942 Mbps, and is about 4x slower than only read and parsing. With added HTTP & DNS dissection, performance slightly decreased further down to 880 Mbps.

TABLE 1. PROCESSING SPEEDS OF OUR OFFLINE TEST SCENARIO ON A SINGLE MACHINE

	Workstation [Mbps]	**Mini-cluster node** [Mbps]
PCAP file reading	5103	5719
Packet parsing	3853	1679
L7 Conversation tracking	942	380
HTTP & DNS extraction	880	358

The *CPU frequency* (performance per CPU core) plays a very important part in overall performance, that can be observed if we compare our *Workstation* with node from *Mini-cluster* — 880 Mbps vs. 358 Mbps. All other components except CPUs are otherwise roughly comparable as we can see by comparing the speed of "PCAP file reading".

The scalability is described in Table 2 that shows performance in *online* mode. The solution was deployed on *Mini-cluster*. The first node was reading the captured communication from a PCAP file and load-balancing it to the rest that reassembled L7 conversations and extracted HTTP and DNS artifacts. In the measurements, we can see an increase in the performance with each added *Reassembler*. When compared with the results in Table 1, the performance of a distributed processing at the *Mini-cluster* exceeded that of a single node running in an *offline* mode. Nevertheless, further optimization is required to achieve linear scalability as a single *L4 Load Balancer* fails to fully saturate available *Reassemblers* by distributing the packets fast enough. We have observed that serialization of messages containing the packets to process heavily contributes to the overall computational complexity and easily becomes a bottleneck of our solution.

TABLE 2. PROCESSING SPEEDS OF OUR ONLINE TEST SCENARIO MEASURED ON MINI-CLUSTER

Reassemblers count	**One** [Mbps]	**Two** [Mbps]	**Three** [Mbps]
HTTP & DNS extraction	233	407	453

We compare our solution, called Network Traffic Processing & Analysis Cluster (NTPAC), running in the *offline* mode at the *Workstation* with commonly used network forensic tools in Table 3. Our solution is an order of magnitude faster while delivering a comparable amount of results in terms of reconstructing L7 conversations and extracting HTTP and DNS artifacts.

TABLE 3. PROCESSING SPEEDS OF COMMONLY USED NETWORK FORENSIC TOOLS MEASURED ON WORKSTATION

NTPAC [Mbps]	**Netfox** [Mbps]	**Wireshark** [Mbps]	**NetworkMiner** [Mbps]
880	65.6	73.4	15.8

VI. Conclusion

In this research, we proposed a system for distributed real-time forensic network traffic analysis up to the application layer capable of large-scale communication processing. We intend to create a system based on the actor model that scales linearly and is hardware independent. The implementation environment of the .NET Core framework and C# language enables rapid development compared to C/C++ that is used by VAST and Moloch. Also, our solution is multiplatform and easily staged with Docker Swarm. Therefore, the deployment of the entire distributed application at the computation cluster is reduced to one command. The solution is distributed under the MIT License and hosted as an open-source project on GitHub here [32].

In the near future, we plan to measure the performance of our solution using data from real-world cases. Because of legal reasons, deployment to public cloud infrastructure is out of the question. Therefore, we need to build a private one that consists of nodes with high CPU frequencies and 10 Gbps network interfaces. Additionally, we need to profile and optimize processing and distribution mechanisms, to expand the set of protocols supported by application protocol dissectors and to add support for tunneling mechanisms.

VII. ACKNOWLEDGEMENT

This work was supported by BUT project ”ICT Tools, Methods and Technologies for Smart Cities” (2017-2019), no. FIT-S-17-3964.

REFERENCES

[1] N. Beebe, “Digital forensic research: The good, the bad and the unaddressed,” in IFIP International Conference on Digital Forensics. Springer, 2009, pp. 17–36.

[2] E. S. Pilli, R. C. Joshi, and R. Niyogi, “Network forensic frameworks: Survey and research challenges,” digital investigation, vol. 7, no. 1-2, 2010, pp. 14–27.

[3] M. Vallentin, “Scalable network forensics,” Ph.D. dissertation, UC Berkeley, 2016.

[4] S. Rekhis, J. Krichene, and N. Boudriga, “Digfornet: digital forensic in networking,” in IFIP International Information Security Conference. Springer, 2008, pp. 637–651.

[5] A. Almulhem and I. Traore, “Experience with engineering a network forensics system,” in International Conference on Information Networking. Springer, 2005, pp. 62–71.

[6] W. Wang and T. E. Daniels, “A graph based approach toward network forensics analysis,” ACM Transactions on Information and System Security (TISSEC), vol. 12, no. 1, Oct. 2008, pp. 4:1–4:33.

[7] N. L. Beebe and J. G. Clark, “A hierarchical, objectives-based framework for the digital investigations process,” Digital Investigation, vol. 2, no. 2, 2005, pp. 147–167.

[8] S. Perumal, “Digital forensic model based on malaysian investigation process,” International Journal of Computer Science and Network Security, vol. 9, no. 8, 2009, pp. 38–44.

[9] W. Halboob, R. Mahmod, M. Abulaish, H. Abbas, and K. Saleem, “Data warehousing based computer forensics investigation framework,” in 2015 12th International Conference on Information Technology-New Generations (ITNG). IEEE, 2015, pp. 163–168.

[10] Zeek, [retrieved: April, 2019]. [Online]. Available: https://www.zeek.org/

[11] Vast, [retrieved: April, 2019]. [Online]. Available: http://vast.io/

[12] Moloch, [retrieved: April, 2019]. [Online]. Available: https://molo.ch/

[13] Microsoft Network Monitor, [retrieved: April, 2019]. [Online]. Available: https://support.microsoft.com/en-us/help/933741/information-about-network-monitor-3

[14] TShark, [retrieved: April, 2019]. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[15] Wireshark, [retrieved: April, 2019]. [Online]. Available: https://www.wireshark.org/

[16] TCPDUMP, [retrieved: April, 2019]. [Online]. Available: https://www.tcpdump.org/

[17] Netfox Detective, [retrieved: April, 2019]. [Online]. Available: https://github.com/nesfit/NetfoxDetective

[18] PyFlag, [retrieved: April, 2019]. [Online]. Available: https://github.com/py4n6/pyflag

[19] NetworkMiner, [retrieved: April, 2019], https://www.netresec.com/?page=NetworkMiner.

[20] EnCase, [retrieved: April, 2019]. [Online]. Available: https://www.guidancesoftware.com/encase-forensic

[21] XPlico, [retrieved: April, 2019]. [Online]. Available: https://www.xplico.org/

[22] W. Ren and H. Jin, “Distributed agent-based real time network intrusion forensics system architecture design,” in Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on, vol. 1. IEEE, 2005, pp. 177–182.

[23] D. Wang, T. Li, S. Liu, J. Zhang, and C. Liu, “Dynamical network forensics based on immune agent,” in Natural Computation, 2007. ICNC 2007. Third International Conference on, vol. 3. IEEE, 2007, pp. 651–656.

[24] S. Khan, A. Gani, A. W. A. Wahab, M. Shiraz, and I. Ahmad, “Network forensics: Review, taxonomy, and open challenges,” Journal of Network and Computer Applications, vol. 66, 2016, pp. 214–235.

[25] M. Cohen, D. Bilby, and G. Caronni, “Distributed forensics and incident response in the enterprise,” Digital Investigation, vol. 8, 2011, pp. S101 – S110, the Proceedings of the Eleventh Annual DFRWS Conference. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287611000363

[26] C. Hewitt, P. Bishop, and R. Steiger, “A universal modular actor formalism for artificial intelligence,” in Proceedings of the 3rd International Joint Conference on Artificial Intelligence, ser. IJCAI’73. Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[27] C. A. R. Hoare, “Communicating sequential processes,” Commun. ACM, vol. 21, no. 8, Aug. 1978, pp. 666–677.

[28] J. Pluskal, “Framework for captured network communication processing,” Master’s thesis, FIT BUT, 2014.

[29] P. Matoušek et al., “Advanced techniques for reconstruction of incomplete network data,” in International Conference on Digital Forensics and Cyber Crime. Springer, 2015, pp. 69–84.

[30] S. Davidoff and J. Ham, Network Forensics: Tracking Hackers through Cyberspace. Prentice Hall, 2012.

[31] M57-Patents Scenario, [retrieved: April, 2019]. [Online]. Available: https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario

[32] NTPAC, [retrieved: April, 2019]. [Online]. Available: https://github.com/nesfit/NTPAC/