Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable

Viliam Letavay
Brno University of Technology
Brno, Czech republic
iletavay@fit.vutbr.cz

Jan Pluskal
Brno University of Technology
Brno, Czech republic
ipluskal@fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology
Brno, Czech republic
rysavy@fit.vutbr.cz

Abstract

Forensic analysis of intercepted network traffic focuses on finding and extracting communication evidence, such as instant messaging, email, VoIP calls, localization information, documents, images. Due to the amount of data captured, this process is time-consuming and complicated. Most commonly used forensic network analysis tools have limited capabilities for large data processing. In this paper, we are introducing a new tool that achieves better data processing performance using available computing resources through distributed processing. Thanks to the technology used, this tool can be used on commodity hardware in a local area network, in a dedicated computing cluster or cloud environment.

CCS Concepts • **Applied computing** → **Network forensics**; • **Networks** → *Network monitoring*; *Network protocols*; Transport protocols; Application layer protocols; • **Social and professional topics** → Computer crime.

Keywords Network Forensics, Network Traffic Processing, Distributed Computing

ACM Reference Format:
Viliam Letavay, Jan Pluskal, and Ondřej Ryšavý. 2019. Network Forensic Analysis for Lawful Enforcement on Steroids, Distributed and Scalable. In *6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3352700.3352720

1 Introduction

Network administrators, cyber-security analysts, and digital forensic investigators capture and analyze network communication to reveal the attack patterns or recover digital evidence. The traditional tools used to process captured communication have limited scalability. For instance, Wireshark is an excellent tool for troubleshooting and security analysis. However, performing analysis of captured files of several gigabytes is cumbersome. Current computing platforms offer tremendous computation power. It is mainly because of its multi-core architecture. The number of cores available per CPU constantly grows[1], contrary to the CPU frequency that is essential for single threaded applications.

However, modifying commonly used single-threaded network forensic tools, e.g., PyFlag, NetworkMiner, to utilize the full potential of modern processors is a complex task which would require extensive modification of their code base. Therefore, new tools for network forensic analysis are in high demand [9, 14].

Even more computing power can be obtained by distributing the workload among a cluster of machines. Availability of industrial strength technology for distributed data processing and scalable storage led to the emergence of distributed network security analysis systems, e.g., Moloch[2], Apache Spot[3], or Apache Metron[4]. Academic research also yields to implementations of scalable network security monitoring systems [26].

Regardless of the technology used, these systems aim to provide a high performance distributed computing environment for network security monitoring (NSM). These tools are especially useful for real-time data processing and complement other systems to defend against cyber threats such as IDS, firewalls, or SIEM. While these tools are also useful for network forensic analysis, forensic investigation favors the depth, accuracy, and reliability of processing over the fast response time. When investigating, it is necessary to reliably analyze any artifact that can be extracted, even though the source data may be corrupted and may not be complete.

1.1 Contribution

This paper discusses the design, performance, and properties of a new Network Forensic and Analysis Tool (NFAT) —

ECBS '19, September 2–3, 2019, Bucharest, Romania
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7636-5/19/09…$15.00
https://doi.org/10.1145/3352700.3352720

[1]Example of the state-of-the-art CPU available on the market — AMD EPYC Rome *64 cores 128 threads*, 2.35 GHz; Intel© Xeon Phi™ 7290F, *72 cores*, 1.5 GHz; Intel© Xeon© Platinum 8180M *28 cores*, *56 threads*, 2.5 GHz
[2]https://molo.ch/ (last accessed 2019-07-03).
[3]http://spot.incubator.apache.org/ (last accessed 2019-07-03).
[4]http://metron.apache.org/ (last accessed 2019-07-03).

Network Traffic Processing & Analysis Cluster (NTPAC) — that utilizes distributed computing architecture to improve the performance of network traffic analysis while being less demanding on hardware requirements than related systems. To extract the evidence from network packets, we need to thoroughly analyze them which means to perform several consecutive operations such as packet dissecting, flow identification, network stream composition, application protocol identification, and message parsing and artifact extraction (see Section 3). Contrary to the other NFAT tools (see Section 2.4), NTPAC is able to correctly process captured traffic that is malformed without yielding misleading evidence (see Section 4.2). NTPAC performs forensic network traffic analysis at high-speed networks. The system design uses a scalable approach that enables to run the tool on a single machine as well as on a computing cluster. in comparison with other NFATs tools, NTPAC is an order of magnitude faster and scales (see Section 3).

1.2 Paper Structure

Initially, background and related work are discussed presenting an overview of current network forensic and security monitoring tools. The architecture of NTPAC is introduced, and the major architectural components are outlined. The paper then provides a preliminary evaluation of the performance that focuses on demonstrating the throughput and scalability of the tool. Finally, we discuss limitations and future work.

2 Background and Related Work

This section provides a background for the paper and lists the related work. First, the actor model and network packet capture analysis are presented. Then we overview existing network forensic tools and frameworks.

2.1 Actor Model

Actor model offers to solve the problems related to parallel and distributed computing elegantly and efficiently. The actor model was first introduced as a theoretical computation model highly influenced by Lisp, Simula and packet switching in computer networks [7]. It defines a fundamental concept called *actor system* that is composed of tiny building blocks called *actors* that execute independently and massively in parallel. The actor is in the distributed world an abstraction of what is an *object* in Object-Oriented Programming; in other words, it bounds data with computation.

Actors communicate asynchronously via message passing. *Actor system* guarantees *at most one delivery*, which means that *any message* can get lost at *any time* but cannot be delivered twice or more. Actor's state changes only as a reaction on a received message. Actor's behavior determines how to process the incoming message by *creating another actor*, *sending a message* to another actor, *changing its state.*

The composition of *actors* in the *actor system* is hierarchical. Each *actor* is responsible for any other *actor* it creates, i.e., the creation of a parent-child relationship. An *Actor* is designed to be as simple as possible, typically without complex inner integrity checks, exception handling, etc. Thus, it can crash at any time. *Parent actor* is responsible for its children and knows how to deal with *children's* failures. This concept greatly simplifies the computation model and allows a programmer to focus only on the most important part that is the core application's functionality and frees him/her from the need of use of synchronization tools (such as mutexes).

2.2 Packet Capture Analysis

Network traffic analysis aims to reveal traces of network attacks and find answers to questions about the incident investigation. Packet analysis starts with dissecting network traffic which performs the following steps: i) *loading PCAP files*, parsing the PCAP file, and extracting individual packets, ii) *dissecting packets with low-level protocol parsers*, including Ethernet, IP, IPv6, TCP or UDP, iii) *collecting TCP packets into streams*, and iv) *applying higher level protocol parsers* to get the required information or extract artifacts.

However, in many cases, it is not possible to obtain plain content from communication because of encryption. Then at least some form of valuable forensic information can be identified, for instance, identities of users [1, 18], devices [17] or applications [16] based on extracted metadata.

Depending on the goal and available tools there are numerous analytic approaches to network packet analysis:

- The bottom-up approach is a prevalent method used by Network Security Monitoring (NSM) [24] oriented analysis that supports several tools, most notably *Wireshark.* All packets are parsed and presented to the investigator who uses filtering, querying and reassembling to identify and extract required artifacts.
- The top-down approach assumes that the Network Forensic and Analysis Tool (NFAT) [15], e.g., *NetworkMiner*, *Xplico*, *PyFlag*, *NetfoxDetective*, can extract information from packets into conversations or other higher level artifacts. These applications visualize this high-level information to the investigator that can then drill down into details if necessary.
- Search based approach considers network communication being just another data format in which it is possible to search for keywords or patterns [11, 20].

2.3 Network Security Monitoring Tools

Network forensic methods were implemented in various NSM tools, e.g., Wireshark, TCP dump, IDS systems (Snort, Zeek), fingerprinting tools (Nmap, p0f), and tools to identify and analyze security threats. As [15] observes, NSM tools are primarily used by network administrators and are intended for detailed bottom-up analysis that requires advanced skills.

Lukashin [12] presented a scalable internet traffic analysis system, which can process multi-terabytes libpcap dump files. It utilizes Apache Spark for data processing to analyze captured packets. The system performs basic analysis and lacks some advanced features required by network forensics. Other approaches to the big data network security analysis were presented by various researchers [2, 19, 30]. Currently, Apache Metron and Apache Spot projects are the most vital. They are frameworks for security analysis of IT threats, enabling to process also firewall and application logs, emails, intrusion-detection reports, and so on. Although they are primarily focusing on network security, they can be valuable as sources of forensic data.

Additionally, there are special appliances for network security monitoring based on custom made FPGA chips that can perform up to 100 Gbps deep packet analysis and export NetFlow with additional information extracted from application protocols [8].

2.4 Network Forensic Analysis Tools

The investigators of Law Enforcement Agencies deal with the enormous number of cases. They require specialized tools that perform top-down analysis and save valuable time [3]. The following list is a selection of notable open-source tools that were designed to support the investigators:

PyFlag is full-fledged NFAT which is intended for disk, memory, and network forensics. PyFlag's design incorporates the concept of a Virtual File System [4]. It implements a specific loader for each supported data source. PyFlag enables to reassemble the content of the communication, e.g., web pages, email conversation.

NetworkMiner is an open source tool that integrates packet sniffing and higher-layer protocol analyzing capabilities into a tool for passive network forensic analysis.

Xplico is a modular NFAT. It consists of the input module handling the loading source data, decoding module equipped with protocol dissectors for decoding the traffic and exporting the content, and the output module organizing decoded data and presenting them to the user. Xplico is a client-server application that can analyze PCAP files as large as several gigabytes.

While all these tools are very useful for investigators as they offer a variety of advanced features, their scalability is limited because they run on either a single computer or in a traditional client-server architecture.

2.5 Big Data Forensics

As distributed frameworks matured, new tools for big data security analysis and digital forensics were designed. Such tools are usually intended for the forensic investigation conducted by network administrators on corporate networks. They usually serve as a complement to Intrusion Detection Systems enabling to capture and analyze hi-speed communication at scale.

Agent-based systems for digital forensics were considered in the literature [21, 22, 29]. These models are more suitable for real-time network forensic analysis from multiple sources, such as logs and captured communication. In these systems, numerous agents perform data collection tasks. The extracted information is then sent to the forensic server and analyzed on this single node only [10], which makes this node to be the bottleneck of the whole system.

The VAST system builds upon Vallentin's previous work — The NIDS Cluster [28] which distributes the workload across multiple workers running *Zeek* to investigate online network traffic and extract *Zeek events*. The VAST system itself goes further and distributes *Zeek events* to workers running in a computing cluster which allows for on-line analysis and interactive queries. Distribution of raw packets is also supported as a 4-tuple with payload up to the speed of 3.1 Gb/s (the libpcap reading speed). According to Vallentin [27] the system does not guarantee that the storage will be able to keep up with the incoming traffic of this speed.

3 Traffic Processing

The goal of NTPAC is to capture and analyze network communication enabling to extract available information. Depending on the case, the forensic investigator may be interested in the content or metadata of application messages. NTPAC handles captured packets according to the following procedure in order to reassemble application messages:

- NTPAC organizes captured packets into separate network layer conversations based on their source and destination IP addresses, providing IP conversation.
- NTPAC then splits IP conversations into TCP/UDP conversations based on the source and destination port numbers and the transport protocol type, as shown in figure 2.
- NTPAC reassembles application conversations from packets separated into individual TCP/UDP conversations. This method utilizes heuristics [13] to recognize multiple application communication multiplexed into a stream of packets of one TCP/UDP conversation caused for example by port reuse.

Because application message extraction is a computationally challenging task, it is a good candidate to run on a computer cluster to improve overall system performance.

Extraction of the artifacts from application messages assumes that we correctly identified the application protocols. Methods based on known port numbers, characteristics patterns in the payload of packets, using statistical methods or machine learning [16] approach can be applied.

However, in many cases, application information cannot be extracted because the content is encrypted. In fact, approximately 76 % of HTTP traffic (at the time of writing this paper) is transmitted by SSL/TLS [5]. In this situation, we cannot extract application messages, but it is possible to get metadata from the SSL/TLS protocol itself, for example, cryptographic information, certificate data, etc. The only exception is two possible situations in which we can decrypt encrypted application data [5]:

1. We have access to the server's private key used in the initialization of an SSL/TLS session, we want to decrypt, and cipher-suites not supporting forward secrecy is used.
2. We can perform a Man-In-The-Middle attack with an SSL/TLS proxy [23] and store session keys.

Most agencies cannot use these techniques because of legal restrictions. For this reason, we did not consider implementing SSL/TLS encryption techniques in our tool.

4 System architecture

The architecture consists of multiple modules that form the processing pipeline (see Figure1). At the highest level, the NTPAC workflow can be divided into two main phases:

Data pre-processing reconstructs application layer conversations (L7 conversation). Each of these conversations is made up of source and destination endpoints, timestamps, and other information that is needed for subsequent processing.

Data analysis identifies application protocols in reconstructed conversations and uses an appropriate application protocol decoder to reconstruct application events from given conversations, such as visited web pages, sent emails, queried domains, etc. The output of this phase is a set of forensic artifacts.

These phases correspond to low-level analysis and high-level analysis. The separation of data pre-processing from the data analysis enables to use the actor-based computational model and offer the ability to distribute the computation. In the rest of the section, details will be given for each module of the processing pipeline.

4.1 Load balancing

The job of the *Load Balancer* nodes is to split the input packet stream, i.e., PCAP file or live traffic, into sub-streams that are then delivered to the reassembling nodes. To avoid the problem of sending packets from the same conversation to different reassemble nodes, the *Load Balancer* calculates the key used to select the destination node from the appropriate protocol fields.

The Eq. 1 calculates the routing key based on communication endpoints (*EP_A* and *EP_B*) and the transport protocol used. Value *n* represents the number of active *Reassembler* nodes.

$$Hash(EP_A \cdot EP_B \cdot Protocol) \bmod n \tag{1}$$

Since all packets from the same conversation (i.e. in both directions of the conversation) should produce the same routing key, we defined an ordering relation $\leq$ for the endpoints[6] and ensured that $EP_A \leq EP_B$ by swapping them if necessary.

While the *Load Balancers* process each packet individually, the data is delivered to *Reassemblers* in batches. This technique helps to decrease network and processing cost of the data distribution.

Back pressure mechanism is used to control the data flow between the nodes. To increase throughput, a *Load Balancer* can submit multiple batches in parallel to the target *Reassemblers*.

IPv4 fragmentation is a challenge for *Load Balancers*. Fragmentation splits one IP packet into multiple IP packets so that the encapsulated transport layer segment header only occurs in the first IP fragment. The *Load Balancer* must, therefore, rebuild the IP fragments to identify the routing key for all fragments of a segment, before it can send them to an appropriate *Reassembler*.

4.2 Conversation Reassembling

Reassembler reconstructs conversations, i.e., two-way traffic layer flows, in batches of packets received from *Load Balancers*. The reassembly process is designed to reconstruct incomplete and corrupted data, using various heuristic techniques [13]. Reassembling is done in several steps until two corresponding flows are assembled, which is illustrated in Figure 2. The entire processing is mapped to actors performing individual steps. Individual L3 and L4 conversations are represented by corresponding actors, which form an actor hierarchy as shown in figure 3. *L3 Conversation* actors are managed by *Capture* actors, which stands for a source capture being analyzed. To enable an analysis of multiple captures at the same time, multiple *Capture* actors can be initiated. The *Captures Controller* actor manages all capture actors.

The packet blocks are first received by the *Captures Controller* actor, which passes them to the appropriate *Capture* actor. The *Capture* actor identifies affiliation of packets to L3 conversations by extracting the IP addresses of the packets and forwards them to appropriate *L3 Conversation* actors which, after identifying affiliation of packets to L4 conversation by extracting the transport protocol and port numbers, forwards the packets to appropriate *L4 Conversation* actors. At these actors, the process of reassembling depends on the transport protocol of the conversation and is performed by either *UDP Conversation Tracker* or *TCP Conversation Tracker*.

[5]https://letsencrypt.org/stats/

[6]The endpoint is a pair of IP address and port number. We consider that there is a suitable lexicographic ordering on a set of endpoints.

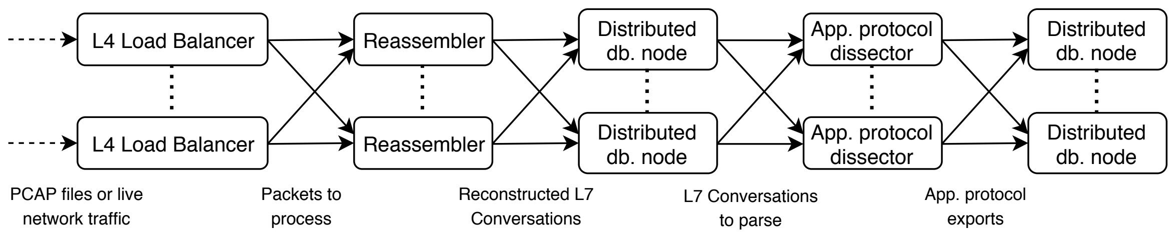

Figure 1. NTPAC's logical architecture

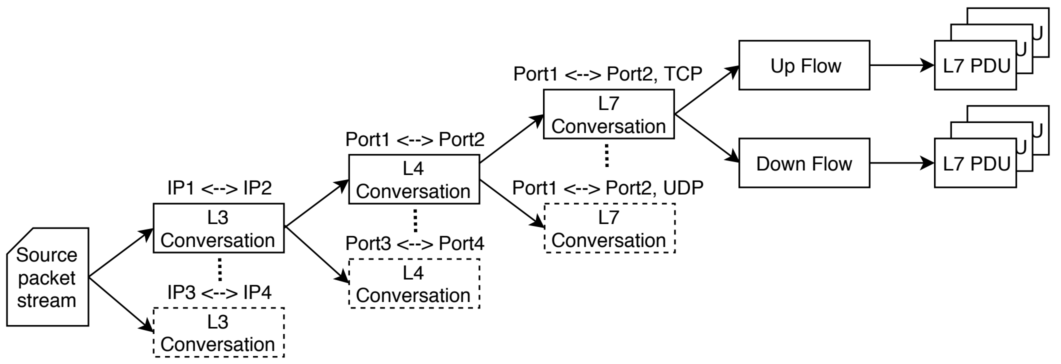

Figure 2. Separation of packets into distinct L3 conversations, L4 conversations and finally L7 conversations. L7 conversations consist of *Upflow* and *Downflow*, which contain a sequence of reconstructed *L7 PDUs*.

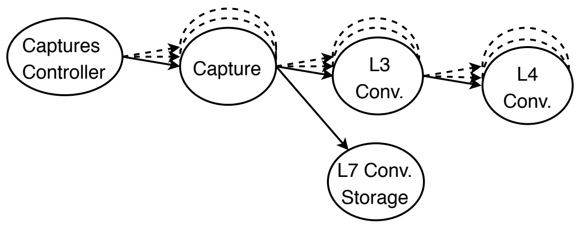

Figure 3. *Reassembler's* actor hierarchy

UDP Protocol Reassembling

UDP is transferring application data as they are, without the use of any additional control packets which implement mechanisms such as flow control or reliable data delivery. *UDP Conversation Tracker*, therefore, treats every transmitted datagram inside given L4 conversation as an individual L7 PDU (Protocol Data Unit). Another important aspect of the UDP protocol is that it is connection-less — it does not establish connections between communicating parties. To distinguish individual L7 Conversations (composed of a pair of *Upflow* and *Downflow*) inside single L4 conversation, *UDP Conversation Tracker* uses a simple heuristic based on a *time delay* between individual L7 PDUs. L7 PDUs in a given direction are considered to be part of a single *flow* if the time difference between their transmission and last recorded activity (timestamp of the last L7 PDU) of a given *flow* is less than a defined value. Experimentally we set this value to 10 minutes, but we are planning to further study UDP behavior of multiple protocols and define this threshold on application protocol bases.

TCP Protocol Reassembling

Processing of TCP protocol is different from handling UDP flows because we can use control information carried along with the data. *TCP Conversation Tracker* is capable of identifying connection initialization and its termination, handling data retransmission and reordering. In the same way as a *UDP Conversation Tracker*, *TCP Conversation Tracker* also processes segments (TCP PDUs) in separate *flows*, which are later paired to form L7 conversations. To create this *flows*, it first stores segments in the so-called *reassembling collection*, in which segments are stored and ordered by their TCP sequence number. Both directions of communication have designated their *reassembling collection*. Before a segment is stored in *reassembling collection*, its sequence number is normalized by incrementing it by *a count of detected sequence number overflows* $\times$ 2^{32} (space of TCP sequence numbers). Sequence number overflows can be caused by a natural overflow of a 32-bit integer sequence number or by establishing a new TCP connection, with ISN (Initial Sequence Number) lower as that of a previous connection. By storing segments in *reassembling collection* and ordering them by their normalized sequence numbers, we achieve that:

1. individual segments inside L7 conversation are ordered;
2. we detect data retransmissions by comparing payloads of segments of which normalized sequence numbers are overlapping;
3. individual L7 conversations inside L4 conversation are ordered by the time they were transmitted.

```
Algorithm tcp_flow_reassembling()
    forall segment in reassembling_collection do
        if SYN flag is set then
            close_flow()
            flow ← create new flow
        else if FIN flag is set then
            close_flow()
        else if flow is nil then
            flow ← create new flow
            add_segment_to_pdu()
        else
            add_segment_to_pdu()
    end
    return flows
Procedure close_flow()
    if flow is nil then
        return
    if pdu is not nil then
        add_pdu_to_flow()
    flows.insert(flow)
    flow ← nil
    return
Procedure add_segment_to_pdu()
    if segment is retransmission then
        return
    if pdu is nil then
        pdu ← create new pdu
    pdu.segments.insert(segment)
    if PSH flag is set then
        add_pdu_to_flow()
    return
Procedure add_pdu_to_flow()
    flow.pdus.insert(pdu)
    pdu ← nil
    return
```

Algorithm 1: TCP *flow* reassembling.

After all segments of L4 conversation have been stored in an appropriate *reassembling collection* (for *Up* and *Down* direction), *TCP Conversation Tracker* iterates through both of them sequentially in order to reconstruct *Upflows* and *Downflows*. Simplified *flow* reassembling algorithm is shown in Algorithm 1. For each segment containing application data, it appends it to current L7 PDU (creates it at first, if it is not already created). After it encounters packet with TCP PSH flag set, it completes current L7 PDU and adds it to the current *flow*. Segments which do not contain application data, such as packets of TCP handshake or connection termination are used to differentiate individual TCP connections by creating appropriate *flows* with assigned created L7 PDUs. Created *Upflows* and *Downflows* are paired by their ISNs (Initial Sequence Numbers) or based on their overlap on time axis in case an ISN of a particular *flow* could not be determined (missing TCP handshake).

L7 Conversation storage

L7 conversations reconstructed by *L4 Conversation* actors are passed to *L7 Conversation Storage* actor. This actor saves contents (series of reconstructed L7 PDUs), as well as metadata (timestamps, endpoints and transport protocols) of these L7 conversations in a distributed database. Our tool uses an abstract data access layer that eliminates any dependence on one database technology. Currently, our solution is primarily based on the use of the Cassandra database engine[7], which has the appropriate features — it has a distributed design, configurable *replication factor* per keyspace and *consistency factor* per query.

4.3 Application protocol parsing

In the *second stage*, a subset of reconstructed L7 conversations is retrieved from the database and further processed to identify and extract interesting application messages:

- First, *Application protocol classifier* block identifies an application protocol of the conversation. Our solution currently implements a simple application protocol classifier based on the database of known ports. However, a more advanced classifier can be used to utilize pattern recognition or statistical methods [6, 16].
- Based on the recognized application protocol, the conversation is consumed by parsing module designed to the processing of a single application protocol such as HTTP, SMTP or DNS. The parsing module processes the entire conversation by extracting individual application protocol messages and storing them back to the distributed database.

The current implementation includes only HTTP and DNS parsers. Adding support for other application protocols requires creating an application protocol parser. Implementing the parser is time-consuming and error-prone. Another option is to generate a parser using a suitable parser generator. Depending on whether the protocol is text or binary, different types of generators can be used, for example, Spicy [25], Kaitai Struct[8], etc.

[7]Note that also MSSQL and ArangoDB are supported.
[8]https://kaitai.io/

5 Performance evaluation

We focused our preliminary assessment on determining the performance parameters of the created tool. During the experiments, we considered both the data storage scenario in the distributed database and the case where data analysis uses the output from the previous step directly. The goal is to demonstrate the scalability of the proposed solution and show the available throughput in various possible configurations. We have considered two major test scenarios:

Standalone processing tests how fast is captured traffic processed on a single machine inside one process. This test-case shows total throughput of our processing algorithms (especially reassembling and application protocol parsing) on given machine type. Because the whole processing is running under one *Common Language Runtime* (CLR), it is expected to be faster than distributed processing with a low number of processing nodes. This experiment provides a baseline to which other results are compared.

Cluster Processing shows the scalability of our solution in a computing cluster. We tested it in a distributed environment with a different number of nodes. The test scenarios considered (i) processing with a single *Load Balancer* and different numbers of *Reassembler* nodes and (ii) a different number of *Load Balancer* and *Reassembler* nodes.

For our test purposes, we have chosen multiple different computing environments described in Table 1. The E.1 environment consists of 14 workstations that are all connected to the same local network. Environment E.2 is a cluster-integrated Google Cloud Platform consisting of 12 virtual machines. E.3 is a mini-cluster of four server boards in a single chassis. Finally, E.4 is a single powerful workstation.

Table 1. Testing environments used for performance evaluation.

	E.1	**E.2**	**E.3**	**E.4**
Machine Type	Desktop computers	Google Cloud Platform (*)	Mini-cluster	Workstation
Machines count	14	12	4	1
CPU Type	Intel i5-3570K	Intel Xeon E5	Intel Xeon E5520	Intel i7-5930K
Physical Cores	4	2	4	6
Logical Cores	4	4	8	12
CPU Frequency	3.40 GHz	2.60 GHz	2.26 GHz	3.50 GHz
CPU Frequency Turbo — 1 core	3.80 GHz	2.80 GHz	3.53 GHz	4.30 GHz
RAM	8 GB	7 GB	48 GB	64 GB
Sequential disk read/write	73/67 MB/s	120/118 MB/s	282/265 MB/s	490/430 MB/s
Network Card	1 Gbps	10 Gbps	1 Gbps	1 Gbps

(*) n1-highcpu-4

As the source packet capture, we used 4.7 GB file from a well known M57-Patents Scenario[9]. It captures real-world corporate network traffic over one month, consisting of 5,707,845 frames. The size of the capture file is large enough to limit the overhead to a negligible part in the initialization phase but allows us to run all test cases in a reasonable time.

To reduce the memory consumption of tracking of all processed conversations by the *Reassembler* nodes, its actors detect and remove inactive (timed out) conversations. Thus, the memory allocation corresponds to the number of active concurrent network flows within a particular time window.

Each experiment was repeated 10 times. The calculated standard deviation was in the range of 5 − 10 %. Such a high value is due to the inherent non-deterministic behavior of the distributed system, including the effect of network communication, the garbage collection, and other operating system processes.

5.1 Single-node Environments

We measured the individual processing stages in the standalone test scenario in environments E.3 and E.4. Table 2 represents the performance achieved for each phase. Preliminary results show that it is possible to read and decode packets from a file at approximately 3.8 Gbps and 1.7 Gbps in test environments E.4 and E.3 respectively (second row of the table). The process of extracting conversations requires much more effort and therefore performance dropped to 972 Gbps and 380 Gbps respectively what represents about 75 % decrease compared to the previous phase. It suggests that this resource-intensive part could be most accelerated by distributed calculation. The last phase is the analysis of HTTP and DNS protocols, which resulted in a decrease in throughput of about 8 % compared to the previous phase. For comparison, Table 3 shows the results achieved by several commonly used network forensic tools (Wireshark, NetworkMiner) in the E.4 test environment.

Table 2. Processing speeds of individual network capture processing phases in *standalone* test scenario performed on test environments E.4 and E.3.

	Workstation E.4 [Mbps]	**Server** E.3 [Mbps]
PCAP file reading	5103	5719
Packet parsing	3853	1679
L7 Conversation reassembling	942	380
Application protocols parsing	880	358

Table 3. Processing speeds of commonly used network forensic tools measured on test environment Workstation E.4.

	NTPAC [Mbps]	**Netfox** [Mbps]	**Wireshark** [Mbps]	**NetworkMiner** [Mbps]
M57 Analysis	880	65.6	73.4	15.8

[9]https://digitalcorpora.org/corpora/scenarios/m57-patents-scenario (last accessed 2019-07-03).

5.2 Clustered Environments

Next, we compare the performance and scalability of our tool in a *clustered* test scenario executed in the test environment E.1. We have performed a series of experiments with the varying number of active *Load Balancer* and *Reassembler* nodes.

Additionally, we have tested configuration in which the results were persisted in a distributed database[10], as well as the configuration, where these results were discarded so we measured performance without the overhead associated with database operations.

Table 4. Performance measurements of *clustered* processing conducted in test environment E.1.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**4** [Mbps]	**6** [Mbps]	**8** [Mbps]	**10** [Mbps]
Load Balancers Without Persistence							
1	513	380	670	768	778	797	815
2		310	574	1093	1370	1508	1542
3		290	602	1136	1713	1945	2070
4		269	660	1258	1971	2252	2580
Load Balancers With Persistence							
1	343	273	478	729	734	740	742
2		247	482	801	1009	1123	1254
3		*	501	930	1131	1326	1438
4		*	503	949	1135	1375	1710

Table 4 shows how the performance depends on the number of *Reassembler* nodes. Columns labeled 1 to 10 represent a number of participating *Reassembler* nodes. For comparison, the column labeled S represents system performance in a stand-alone mode of the processing. First set of rows (labeled Load Balancers Without persistence) denote a varying number of participating *Load Balancer* nodes without the results being stored in a database. Similarly, the second set of rows (labeled Load Balancers With Persistence) denote a varying number of participating *Load Balancer* nodes but with results being stored in a database.

In the test results, we see that performance increases to the point where one *Load Balancer* cannot provide enough data for available *Reassembler* nodes. Adding additional *Load Balancer* nodes increases the throughput of the entire system until all *Reassemblers* are fully saturated, and the processing speed reaches its limit again. Increasing a number of both *Load Balancer* and *Reassembler* nodes allows a further increase in overall throughput until the available hardware resources are exhausted. Data points marked with asterisks (*) represent incapability to complete the test run due to the overload of the *Reassembler* nodes in a given configuration (total number of active nodes).

With the knowledge of the characteristics of the distributed system obtained from experiments in the E.1 environment, we repeated the same set of experiments in E.2 (using up to 8 *Reassemblers* and up to 4 *Load Balancers*) and E.3 (using up to 3 *Reassemblers* and single *Load Balancer*). The results shown in tables 5 and 6 show a similar trend in the rate of processing per number of individual modules. Note, that we are limited by the total number of instances that we can create in environment E.2.

[10]The number of Cassandra nodes was equal to the number of active *Reassembler* nodes.

Table 5. Performance measurements of *clustered* processing conducted in test environment E.2.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**4** [Mbps]	**6** [Mbps]	**8** [Mbps]
Load Balancers Without Persistence						
1	427	223	370	560	573	585
2		170	334	706	916	994
3		126	352	734	826	1016
4		104	271	580	618	920
Load Balancers With Persistence						
1	248	171	255	459	497	498
2		*	219	420	459	675
3		*	*	383	452	558
4		*	*	*	*	*

Table 6. Performance measurements of *clustered* processing conducted in test environment E.3.

Reassemblers	**S** [Mbps]	**1** [Mbps]	**2** [Mbps]	**3** [Mbps]
1 — Without Persistence	358	233	407	453
1 — With Persistence	210	158	301	388

When comparing results from different environments, it is interesting that the highest performance was achieved in the local network, although the Google Cloud Platform seems to have more powerful computing nodes and a faster network. This may be because GCP is a virtualized environment with shared hardware resources.

6 Conclusion

We have designed and implemented a system for forensic network analysis that can be used in high-speed networks for near real-time analysis. The distributed system is based on an actor model that, thanks to its good scalability, can run on a single machine as well as a computing cluster.

The proposed distributed system is comprised of different classes of cooperating nodes capable of distributing intercepted network traffic, processing identified network flows and storing reconstructed data into a distributed database. The resulting data consists of a description of network conversations and information from the extracted application communication. At this point, DNS and HTTP are supported.

The main goal of the system is to provide a scalable platform for network communication processing that is primarily designed to support a digital investigation. Experiments have demonstrated the feasibility of the proposed approach.

Processing throughput is scalable by adding additional processing nodes. Experiments have also shown that the proposed tool running on only one node can effectively use available resources and can offer the same or better performance than existing tools.

The NTPAC is open source, and available at https://github.com/nesfit/NTPAC under the MIT license.

References

[1] G. Alotibi, N. Clarke, Fudong Li, and S. Furnell. 2016. User profiling from network traffic via novel application-level interactions. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*. 279–285. https://doi.org/10.1109/ICITST.2016.7856712

[2] M. Aupetit, Y. Zhauniarovich, G. Vasiliadis, M. Dacier, and Y. Boshmaf. 2016. Visualization of actionable knowledge to mitigate DRDoS attacks. In *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*. 1–8. https://doi.org/10.1109/VIZSEC.2016.7739577

[3] Nicole Beebe. 2009. Digital forensic research: The good, the bad and the unaddressed. In *IFIP International Conference on Digital Forensics*. Springer, 17–36.

[4] MI Cohen. 2008. PyFlag–An advanced network forensic framework. *Digital investigation* 5 (2008), S112–S120.

[5] Sherri Davidoff and Jonathan Ham. 2012. *Network Forensics: Tracking Hackers through Cyberspace*. Prentice Hall.

[6] Alice Este, Francesco Gargiulo, Francesco Gringoli, Luca Salgarelli, and Carlo Sansone. 2008. Pattern recognition approaches for classifying ip flows. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 885–895. https://doi.org/10.1007/978-3-540-89689-0_92

[7] Carl Hewitt, Peter Bishop, and Richard Steiger. 1973. Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, Vol. 3. Stanford Research Institute, 235.

[8] L. Kekely, J. Kučera, V. Puš, J. Kořenek, and A. V. Vasilakos. 2016. Software Defined Monitoring of Application Protocols. *IEEE Trans. Comput.* 65, 2 (Feb 2016), 615–626. https://doi.org/10.1109/TC.2015.2423668

[9] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. 2016. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications* 66 (2016), 214 – 235. https://doi.org/10.1016/j.jnca.2016.03.005

[10] Suleman Khan, Abdullah Gani, Ainuddin Wahid Abdul Wahab, Muhammad Shiraz, and Iftikhar Ahmad. 2016. Network forensics: Review, taxonomy, and open challenges. *Journal of Network and Computer Applications* 66 (2016), 214–235.

[11] Mark Longworth, John D Abromavage, Todd A Moore, Scott V Totman, and Vince Romano. 2006. System and method for network security. US Patent 7,016,951.

[12] Alexey Lukashin, Leonid Laboshin, Vladimir Zaborovsky, and Vladimir Mulukha. 2014. Distributed Packet Trace Processing Method for Information Security Analysis. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy (Eds.). Springer International Publishing, Cham, 535–543.

[13] Petr Matoušek, Jan Pluskal, Ondřej Ryšavý, Vladimír Veselý, Martin Kmet’, Filip Karpíšek, and Martin Vymlátil. 2015. Advanced techniques for reconstruction of incomplete network data. In *International Conference on Digital Forensics and Cyber Crime*. Springer, 69–84.

[14] Erik E Northrop and Heather R Lipford. 2014. Exploring the usability of open source network forensic tools. In *Proceedings of the 2014 ACM Workshop on Security Information Workers*. ACM, 1–8.

[15] Emmanuel S Pilli, Ramesh C Joshi, and Rajdeep Niyogi. 2010. Network forensic frameworks: Survey and research challenges. *digital investigation* 7, 1-2 (2010), 14–27.

[16] Jan Pluskal, Ondrej Lichtner, and Ondřej Ryšavý. 2018. Traffic Classification and Application Identification in Network Forensics. In *IFIP International Conference on Digital Forensics*. Springer, 161–181.

[17] Libor Polčák and Barbora Franková. 2015. Clock-Skew-Based Computer Identification: Traps and Pitfalls. *J. UCS* 21, 9 (2015), 1210–1233.

[18] Libor Polčák, Radek Hranický, and Tomáš Martínek. 2014. On Identities in Modern Networks. *Journal of Digital Forensics, Security and Law* 9, 2 (2014), 2.

[19] N. Promrit and A. Mingkhwan. 2015. Traffic Flow Classification and Visualization for Network Forensic Analysis. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. 358–364. https://doi.org/10.1109/AINA.2015.207

[20] Gil Raviv. 2013. Methods for user profiling for detecting insider threats based on internet search patterns and forensics of search keywords. US Patent 8,375,452.

[21] Wei Ren. 2004. On A Reference Model of Distributed Cooperative Network, Forensics System.. In *iiWAS*.

[22] Wei Ren and Hai Jin. 2005. 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, 177–182.

[23] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[24] Rommel Sira. 2003. Network forensics analysis tools: an overview of an emerging technology. *GSEC, version* 1 (2003), 1–10.

[25] Robin Sommer, Johanna Amann, and Seth Hall. 2016. Spicy: a unified deep packet inspection framework for safely dissecting all your data. *Acsac* (2016). https://doi.org/10.1145/2991079.2991100

[26] Matthias Vallentin, Dominik Charousset, Thomas C Schmidt, Vern Paxson, and Icsi U C Berkeley. 2014. Native Actors : How to Scale Network Forensics. *Sigcomm 2014* (2014). https://doi.org/10.1145/2619239.2631471

[27] Matthias Vallentin, Vern Paxson, and Robin Sommer. 2016. VAST: A Unified Platform for Interactive Network Forensics.. In *NSDI*. 345–362.

[28] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. 2007. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 107–126.

[29] Diangang Wang, Tao Li, Sunjun Liu, Jianhua Zhang, and Caiming Liu. 2007. Dynamical network forensics based on immune agent. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, Vol. 3. IEEE, 651–656.

[30] M. Wullink, G. C. M. Moura, M. MÃijller, and C. Hesselman. 2016. ENTRADA: A high-performance network traffic data streaming warehouse. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. 913–918. https://doi.org/10.1109/NOMS.2016.7502925