NTPAC: A Distributed System for Network Forensics

Viliam Letavay, Jan Pluskal, and Ondrej Rysavy

Brno University of Technology, Bozetechova 2, Brno, Czech Republic
{iletavay, ipluskal, rysavy}@fit.vutbr.cz
http://www.fit.vutbr.cz/

Abstract. The Internet brought new opportunities for cybercrime activities. Security administrators and LEA (Law Enforcement Agency) officers call for powerful tools for high-speed network communication analysis of an enormous amount of traffic. This paper presents a novel tool capable of both real-time and post-mortem network traffic analysis. The system enables to extract information up to an application layer. The scalability is achieved by employing the distributed computation model. Moreover, the presented solution includes heuristics to analyze incomplete communication.

Keywords: Network forensics, Network traffic processing, Actor model, Distributed Computing, Tool Implementation

1 Introduction

In network forensic investigation, the network traffic is collected, and captured packets are further processed and analyzed. For investigators, the evidence is primarily located in application messages such as instant messaging, emails, voice, RTP, localization information, documents, pictures, etc. The amount of data that needs to be processed to extract evidence from the network communication is enormous. Also decoding packets up to application messages and retrieving artifacts is not a trivial task. Today computers offer tremendous computation power thanks to their multi-core architecture. However, modifying a program to use multi-core processing is relatively complex and usually require extensive modification. Often, a different computation model is required. Carl Hewitt developed the actor model of computation in 1973, and it gained its popularity only recently because of the availability of hardware and infrastructure having the necessary capabilities.

In our paper, we propose a proof of concept (PoC) of our distributed network forensic tool the NTPAC (Network Traffic Processing and Analysing Cluster), based on *actor model* that provides an efficient and scalable system for packet decoding and artifacts extraction. We overview the tool design, discuss its main functionality and evaluate its performance. We directly compare it to the VAST [1] system, which also employs the actor model, but it is intended for interactive real-time analysis and supports a limited application protocol decoding capabilities.

2 Background

2.1 Network Forensics

Network forensics has been widely neglected in recent years as Vallentin has shown [1]. Majority of authors threats it as a part of *Digital forensics* and continues to threat it as such. There is a vast difference between typical digital forensics and network.

In digital forensics, the first thing that investigator should do is reconnaissance. He needs to be assured that volatile information is not lost during the acquisition of evidence [2]. Mostly, this first part is protruded by operatives from LEA lines, and a probability of volatile information loss is high, because of the lack of expert knowledge. This state unimaginably threatens the success of an investigation as a whole. Just recently, vast sums are spent on development and teaching of digital forensics curricula to the first line operatives to be able to capture volatile information with advanced techniques to obtain an image of otherwise encrypted hard-drive or operation memory. The expert investigator works with a data source that is not volatile afterward, but backed up and properly archived with consistency ensured. Evidence obtained from such data can be validated and repeatably carved by a multiplicity of tools and techniques.

Network forensics, on the other hand, is a process that identifies, captures and analyzes network traffic [2]. Network traffic is totally volatile. Information that is not captured on the first time is lost forever. Identification of proper network line to TAP is not a trivial task because of poor network documentation from an Internet Service Provider (ISP) part, asymmetric routing, network dynamic nature, and an error rate of the actual physical connections and so on. Just theoretically abandoning legality concerns that enforce strict rules on traffic interception, unabridged analysis of all traffic of a simple home connection would require wast computational resources and investigator's time to manually analyze all extracted fragments from the communication.

This kind of *Network forensic* techniques, as we also understands it, was implemented in several network forensic frameworks [3–8] and tools. *PyFlag*, *CapAnalysis*, *Xplico* and *NetworkMiner* are examples of widely used network forensic tools. These tools are usually meant to run on investigator's machine and process a small part of filtered communication with the highest probability of evidence containment.

There is also another direction of *Network forensics* that is most concerned with the security of computer systems or network infrastructure of large organizations. This part of network forensics is building *Intrusion Detection* and *Intrusion Prevention* Systems and grows in its importance swiftly.

Majority of authors does not make this distinction and mentioning *Network forensics* they mean *IDS/IPS* tools and not *network forensic tools* that helps LEA investigators.

2.2 Actor Model of Computation

The *actor model* is one of the attractive solutions that solve the problems related to 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 [9]. It defines a primitive 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.

Actors communicate asynchronously via message passing, that is managed by the *actor system* which guarantees *at most one delivery*. This means that *any message* can get lost in *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.

Composition of *actors* in *actor system* is hierarchical. Each *actor* is responsible for any other *actor* it creates, i.e., creation of parent-child relationship. *Actor* is designed to be as simple as possible 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* failure. This concept greatly simplifies the computation model and forces programmer to focus only on the most important part that is the functionality and not the handling of exceptional states.

3 Related Work

Network forensic analysis methods were implemented in various tools. General purpose tools include network analyzers (Wireshark, TCP dump), IDS systems (Snort, Bro), fingerprinting tools (Nmap, p0f), and tools to identify and analyze security threats.

PyFlag is a general purpose forensic package which can be used as disk forensics, memory forensics, and network forensics tool. PyFlag is designed around the Virtual File System concept. For each supported data source a specific loader is implemented. PyFlag enables to reassemble the content of the communication, e.g., web pages, email conversation, etc. Network Miner is an open source tool that integrates packet sniffer and higher-layer protocol analyzers into a tool for passive network traffic monitoring and analysis. Xplico is a modular tool aimed at the reconstruction of the data content carried in the network traffic. The software 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 of these tools are very valuable for investigators, they have problems when processing capture files bigger than several gigabytes.

Lukashin et al. [10] 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 in [11], [12], and [13]. Currently, Apache Metron[1] and Apache Spot[2] projects are the most vital. They

[1] http://metron.apache.org/

[2] http://spot.incubator.apache.org/

are frameworks for security analysis of IT threats, enabling to process also firewall and application logs, emails, intrusion-detection reports, etc. Although they are primarily focusing on network security, they can be valuable as sources of forensic data.

Agent-based systems for digital forensics were considered in the literature [14–16]. 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 [17] only, which makes this node to be the bottleneck of the whole system. The VAST system builds upon Vallentin's previous work (The NIDS Cluster) [18] which distributes the workload across multiple workers running *Bro* to investigate online network traffic and extract *Bro events*. The VAST system itself goes further and distributes *Bro 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 [19] up to the speed of 3.1 Gbps (the libpcap reading speed) without no guarantee that the storage will be able to keep up with the incoming traffic of this speed.

We were inspired by the VAST to employ agent design in our system prototype. Each actor represents an independent processing unit. The communication between actors is conducted via message passing. The actor has no shared state; thus all actors work in parallel. If actors run on the same node, the message passing has a little overhead compared to a function call or a loop. However, if actors scale over multiple nodes, messages need to be serialized. The serialization process introduces latency and adds overhead. Nevertheless, such design lacks a single point of failure and enables scalability across multiple physical nodes. Moreover, dynamic resizing of the computing cluster is possible.

Network	1 h	1 d	1 w	1 m
1 Mbps	9 GB	216 GB	1.5 TB	6.5 TB
100 Mbps	90 GB	2.2 TB	15.1 TB	64.8 TB
1000 Mbps	900 GB	21.6 TB	151.2 TB	648 TB

Table 1: Maximum size of a capture file for different network types.

4 Problem Statement

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 network speeds, an unabridged analysis would require enormous computation resources. Table 1 presents maximal sizes of capture files depending on the network speed and the data acquisition interval. Note that for 1Gbps network the duplex communication can produce up to 250MB of data each second. To get evidence from network communication, captured packets have to be thoroughly analyzed which amounts to perform several nontrivial operations such as flow identification and reconstruction, protocol decoding and artifact extractions.

The goal of network forensics is to extract evidence from network communication. The content of the communication is the ultimate evidence desired by investigators. However, in many cases, it is not possible to obtain the plain content from the communication because of encryption. Then the available evidence may be one of the following:

– identification of communicating users [20];
– identification of communicating devices [21];
– identification of communicating applications [22];
– proof of user's presence at home, work, etc.

In all these cases, we do not need to keep the full content of the communication but rather its subset carrying enough information to support the correctness of the extracted evidence. If we want to reduce the size of captured data, the necessary packet processing must be done in near real-time. It means that the system should be able to analyze about 250MB of packets every second for 1Gbps link speed. From our previous experience based on experiments and measurements, we know that to perform all required operations in real-time and over long periods of time is challenging on a standard-issue PC despite its relatively high performance. Dissection of frames and grouping packets in conversations followed by decoding protocols up to the application layer is a time-consuming operation. We wrote a tool that was able to reach processing speed of 300 Mbps [23, pp. 45-51] on the commodity machine. Several vendors offer specialized boxes for packet capturing and with limited processing capabilities for speeds above 1Gbps, and these come at a high price. Our aim is different and can be expressed as follows:

- We seek the solution for providing packet capture analysis in high-speed networks using a computing cluster developed from commodity hardware. This way, the tool will be more affordable for investigators.
- The solution should be easily extensible with new features. One should be able to write a new module and plug it to the processing pipeline of the system. This is hard to achieve in optimized and closed tools.
- We aim to provide a scalable solution capable of processing data at various speeds depending on the available hardware resources.
- We aim at providing an open interface so that the solution can be integrated with other 3rd party tools.

5 System Design

We intended to design NTPAC with as a flexible PoC solution to be deployable on a single machine as well as a cluster of commodity servers, desktop computers or any other system running Linux or Windows. In this section, we describe *processing pipeline* based on *actor model* architecture that enables feasible scaling from one to N machines. Next, we discuss possible network topologies that can be employed to interconnect computation nodes resides in one computer network or several like it is usual in data-center deployments. We conclude this section by the detailed description of particular *actor groups* to outline their functionality and intercommunication.

5.1 Processing Pipeline

Incomplete data provided by unreliable traffic interception can lead to skewed results; some information may be lost, some unconsciously fabricated by reconstruction process. Keeping these facts in mind, the processing cannot strictly follow RFCs and behave like a *kernel* network stack implementation, but it has to incorporate heuristics and additional constraints to prevent data manipulation from happening. For example, the reconstruction process needs to mark missing gaps in communication and to consider these marks during application protocol processing, or never to join multiple frames into a single conversation unless it passes more advanced heuristics and checks. Network forensic tools which we have worked with do mostly respect RFCs and thus may produce misleading results as we have shown in our previous research [24].

We propose a distributed architecture with no single point of failure, composed of commodity hardware that will be capable of linear scalability, and fine resource utilization. See Figure 1 for design details. In our solution, every message is acknowledged to ensure that no data-loss can occur. We rather slow down

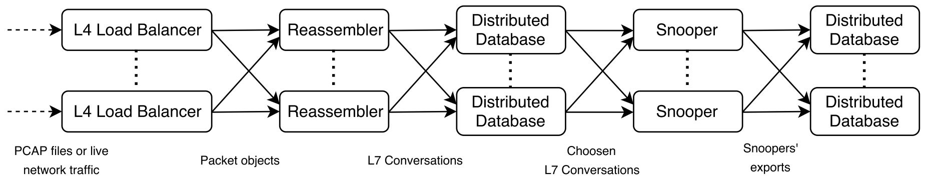


Fig. 1: Architecture diagram shows the proposed system's nodes with information flow between them. Interconnections between agents are logical, independent of underlying hardware architecture. The solution is to be deployed on a single node or scale up in a distributed environment.

the processing in favor of speed to eliminate a risk that any packet is drooped. This design decision distinguish *NTPAC* aside from *Suricata*, *VAST*, *Moloch*, and any other IDS/IPS system.

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

Data preprocessing Reconstruction of conversations at the application layer (L7 conversations) from the captured traffic. Each of these conversations holds information about the source and destination endpoints, time stamps and reassembled payloads of exchanged application messages. The outputs are meta information describing conversation and the abstract data structure containing reassembled payloads.

Data analysis Identification of application protocols in reconstructed L7 conversations and subsequent use of a proper application protocol dissector to reconstruct application events from given conversations (e.g., visited web pages, sent emails, ...). The output of this stage is a set of forensic artifacts.

In this paper, we focus mainly on the *Data preprossessing* part of our work, but we also briefly touch *Data analysis* as well. For the data reconstruction part, the same techniques described in our previous work [24] can be applied. Because data are stored in the distributed database – *Cassandra*, they are highly available, and concurrent reading/writing operations do not impact performance, as they would have if the system were running on a single machine.

5.2 Physical Topology

In order for a system to be distributed, it needs to run in multiple instances that are interconnected and creates particular network topology. These days, we recognize several network topologies that may serve our needs:

– **Full-mesh** topology would be the ideal, bandwidth maximal, but is also the least cost-efficient. Each node is connected to any node in topology, creating maximal possible network bisection which results in low utilization.
– **Star** topology uses a switching device to connect all nodes. As long as switching plane is sufficient to withstand full line speed of all connected nodes and nodes are connected to the switch with interfaces of adequate speed, the intercommunication is no longer a bottleneck, but costs are kept to a minimum. For redundancy purposes and to eliminate a single point of failure, two or more switch devices are commonly used.
– **Partial-mesh** topology is a trade-off between star and full-mesh in both performance and cost. Only selected nodes are interconnected, and routing needs to be applied.
– **Custom** topology is used when the particular logical topology is mapped to physical. Nodes that exchange the most amount of data are directly connected. Performance is similar to *star* but without a need of the *switch.*

We chose to test *star* and *custom* topologies, see details in *Preliminary Performance Evaluation* section6.1. *Full-mesh* and *Partial-mesh* topology types were considered inefficient for our use-case and not considered further.

Custom topology BPI-R2 Our custom topology was created for a Banana PI R2 cluster to test our hypothesis whether it is efficient to conduct traffic

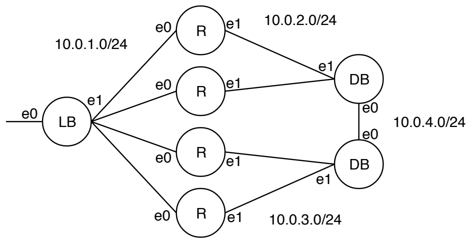


Fig. 2: The figure describes our custom topology used during the development of our solution. The network is divided into four IP networks. Interfaces *e0* are network cards, *e1* interface is a four-port physical switch with one extra virtual port connected to BPI-R2.

processing on low-cost single-board computers. Topology is shown on Figure 2. We used this topology for development purposes only, because of its scalability limits. *BPI-R2* board contains two network card. First network cards are dedicated, second is virtually shared with a four-port switch. This is a limitation, because one *LoadBalancer* can be connected up to four *Reassembling* nodes.

This topology also has benefits like a creation of multiple networks and the use of the separate interface for a particular given task. In this case, *LoadBalancer* can process *live traffic* captured on *e0* and forward it on *reassembler* nodes using interface *e1* with a full link bandwidth to its disposal. Management network is created using WLAN integrated on the board. This way, we guarantee that no peak in any network traffic can interfere or dominate over another and manageability all-the-time is ensured.

Star topology The *star topology* is the most commonly used topology interconnecting devices in TCP/IP based computer networks. In our use-case, we eliminate a single point of failure and increase throughput using two switches. If we connected this topology in a naive way, *Spanning Tree Protocol* (STP) would cut off redundant lines. We want to increase bandwidth by a factor of redundancy thus we have following options:

- **Network** per node-switch connection would require a fast dynamic routing protocol (e.g., EIGRP) to be set up on all nodes and switches and L3 switches would be required.
- **Bonding** of multiple interfaces into one logical IP network. As long as switches are not interconnected, we can shut down all switch ports in a case that one port fails to prevent data loss.
- **SDN** to achieve custom virtual topology and create end-to-end host connections with *Link Aggregation Control Protocol* (LACP). This would require *Open vSwitch* or any other switch supporting *OpenFlow*.

We have experimented with all solutions and determined that *Network* and *SDN* are most suitable for production deployment but harder to set-up. For development and testing purposes is simplest to set-up *Bonding*.

To ensure that all nodes are reachable even when a network is saturated by data traffic, we need to consider the creation of the management network. Using separate physical network would waste resources thus we are left with the choice to divide physical network using:

- **VLANs** and configure IP networks manually. This technique can be applied only on one L2 broadcast domain.
- **VXLAN** to overcome L2 broadcast boundary and scale up our topology over multiple physical networks.

We use Docker Swarm for deployment and usage of *VXLAN* that is Swarm's default configuration. See Figure 3 for topology details.

5.3 Logical Processing Phases

Using physical topologies described in Section 5.2, we will discuss logical interconnections and topology of execution units, in our case the *Actors*. The NTPAC

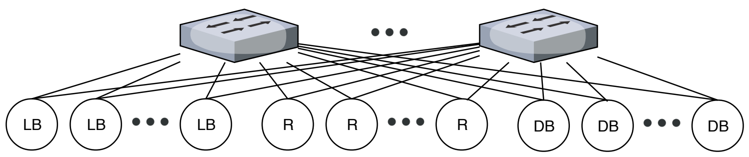


Fig. 3: The figure describes a *star topology* with a single IP network and redundancy formed by a pair of switches. Therefore, devices are interconnected with a bandwidth of factor of network interfaces.

system as a whole is visualized on Figure 4 that describes the implementation of *actors* and messages passed between them. This figure respects architecture design described in Figure 1.

We can see major components grouped into groups:

- **OnlineLoadBalancer** represents actors running on a loadbalancing node that ingest data into the cluster.
- **OfflineLoadBalancer** represents actors that similarly as *OnlineLoadBalancer* ingest data, but not into the cluster but into *CaptureTracking* actors running in the same process, thus on the same node.
- **CaptureTracking** actors dissect ingested packets, create conversations, reassemble and store them in a distributed database.
- **CaptureController** is an actor that is used in cluster mode during *online load balancing*. It receives ingested data and passes them into the same group of *CaptureTracking* actors that are used in both *cluster* (Online) and *non-cluster* (Offline) setups.

Messages passed between actors can carry data, or are used for synchronization, i.e., notification of a state change, or operation finish. Management of actor model cluster is under the control of AKKA.NET framework. AKKA takes care of fair scheduling of actors on CPU cores, memory management, message passing, data serialization and exchange between nodes, etc. *NTPAC* PoC based on

AKKA.NET framework is designed to operate on a single host (i.e., process) or multiple nodes.

Clustered solution can run directly on host OS or is deployed with Docker Swarm to ease orchestration burden and set required number of *LoadBalancers* and *Reassemblers*. It consists of *OnlineLoadBalancer*, *CaptureController*, and *CaptureTracking* actor groups, see Figure 4. The *OnlineLoadBalancer* runs as one application called *LoadBalancer*, *CaptureController* and *CaptureTracking* as another that we call *Reassembler*. Both are part of architecture design presented on Figure 1.

Single host solution runs under one process under single *Common Language Runtime* (CLR). This is very effective, because there is no need for costly serialization of inter-node messages. Nevertheless, processing speed of single host is finite. See Figure 8 compares these two approaches. This single host application consists of *OfflineLoadBalancer* and *CaptureTracking* groups, see Figure 4.

Loadbalancing In the simplest use-case, we have one source stream (i.e., one PCAP file) which we want to analyze. Therefore to utilize more than one *Reassembler* instance, we have to split packets from this stream into a smaller sub-streams, which will be distributed among multiple *Reassembler* instances. For this split, we can not use a naive method such as *Round Robin*. *Reassembler* nodes operate independently on each other and to fully reconstruct L7 conversations (each can consist of multiple packets), they have to obtain all the pieces of the particular L7 conversation.

Using this simple method, a situation can occur when a half the packets from one L7 conversation will end up in one *Reassembler* node and a second half in some other. Thus both nodes would end up with incomplete data, and none of them would be able to reconstruct the conversation entirely.

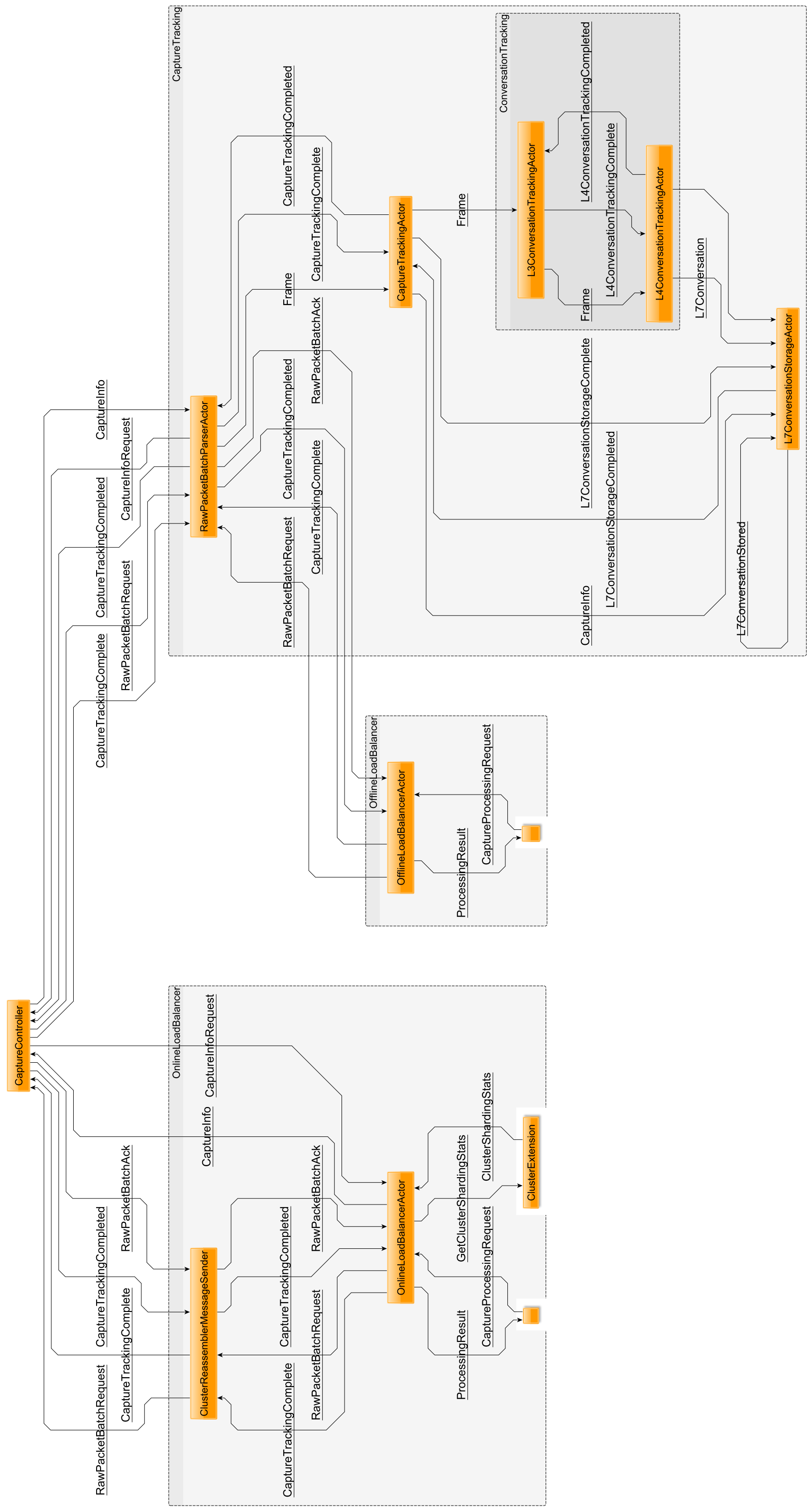


Fig. 4: The figure shows the interconnection of Actors and Passed Messages.

Solution to this problem is another type of node called *LoadBalancer*, which will be positioned in front of the *Reassembler* nodes. They will extract source and destination IP addresses, ports and transport protocol type from each packet of the source stream, and we will use them to decide to which instance of *Reassembler* should forward the packet concerning its context. This way, all packets of a particular L4 conversation will always be forwarded to precisely one *Reassembler* instance. We need have to keep in mind also a *IPv4 fragmentation* that splits originally one *transport packet* (L4 segment) to multiple *IP packets*. Consequent IP packets lack the transport protocol headers directly, therefore, IPv4 defragmentation is required on *LoadBalancer* node.

Reassembling From received packets, *Reassembler* instances can reconstruct L7 conversations in the process called *L7 reassembling*. Every reconstructed L7 conversation holds information about its endpoints (source and destination IP addresses, ports, and transport protocol), timestamp and a sequence of an L7 PDUs (Protocol Data Units). L7 PDU represents transmitted application data. The process of *reassembling* has to be designed with incomplete and damaged data capture in mind such as the capture of only unidirectional traffic or loss of particular packets (for example packets of the TCP handshake). It has to be able to cope with this situations and retrieve as much information from captured traffic as possible.

To achieve this, we have designed a custom heuristic to reassemble L7 conversations. This process begins by separating the packets into the corresponding L3 conversations based on their source and destination IP address and IP protocol tuple. Packets from each L3 conversation are then further separated into corresponding L4 conversations based on their source and destination port and transport protocol type tuple as shown on figure 5.

With packets separated into L4 conversations, we can reassemble (possibly multiple because of port reuse) L7 conversations from each of this L4 conversation individually. At this point the process of reassembling splits into two possible sub-processes based on used transport protocol, *UDP Conversation Tracker* and *TCP Conversation Tracker*. Every L4 conversation has a designated conversation tracker which processes only its L4 conversation's packets. Both types of conversation trackers share a common approach to reassembling as both of them process packets separately in individual directions (to and from communication initiator), creating unidirectional fragments of L7 conversations. These fragments – *flows*, store sequences of L7 PDUs in a certain direction. *Up Flow* holds packets in the direction from communication initiator and on the other hand, *Down Flow* holds those in the opposite direction – to communication initiator. Reconstructed *flows* are finally paired with their counterparts (opposite *flows*) and form L7 conversations. Pairing heuristics depends on concrete conversation tracker, whether it is by computing of *flows*' overlaps in time for *UDP Conversation Tracker*, or by comparing *flows*' identifiers which are based on TCP connections' Initial Sequence Numbers for *TCP Conversation Tracker*. In cases of processing of unidirectional captures, unpaired *flows* form unidirectional L7 conversations. This was the motivation behind the usage of the system of separate *flows*, and not processing packets in individual L4 conversations as they are, in both directions, together. For more details concerning precise reassembling algorithm see [24].

Storage The reconstructed L7 conversations are stored in a distributed database, ready to be retrieved in the second stage of the execution. We are using abstracted data access layer to stay database agnostic. Currently, NTPAC supports *MSSQL* (sharded), *ArangoDB*, and *Cassandra*. We achieved the best per-

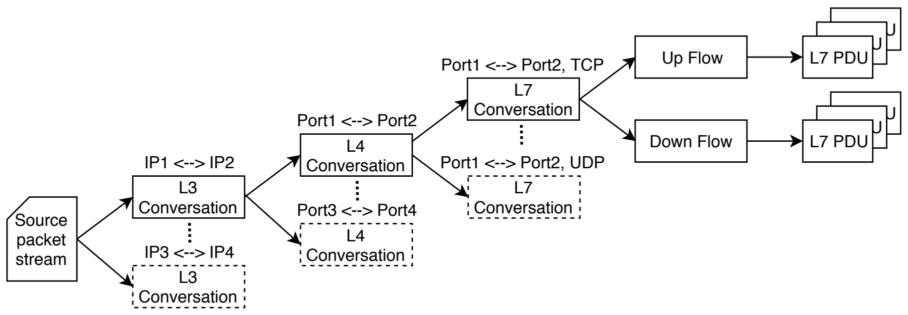


Fig. 5: Separation of packets into distinct L3 conversations, L4 conversations and finally L7 conversations. L7 conversations consists of *Up flow* and *Down flow*, which contain sequence of reconstructed *L7 PDUs*.

formance results with Cassandra, so all results are measured with the *Cassandra* database.

The database, if poorly chosen, would become a bottleneck for the whole processing. The sequential, append-only network data ideally requires to write optimized database that is capable of *tens of thousand* write operations per second. Cassandra by its distributed design, configurable *replication factor* per keyspace and *consistency factor* per query shows ideal properties.

During the capture processing, we need to store only meta information described on listing 1.1 and listing 1.2. These tables serve as the simplified example of data that are stored. In production, tables need to be optimized by queries that process them. In NoSQL (non-relational) databases, it is preferred to stored denormalized data in opposite to third normal form because of performance considerations.

Queries returning data for further analysis need to be designed in a way that ensures that only one node is queried. Thus, the whole partition key is specified in the query. In *Cassandra*, there is a support for a special type of query (ALLOW FILTERING modifier) that runs on all nodes, but its performance hurt is unpredictable.

Listing 1.1: Capture table

```
CREATE TABLE
ntpac.capture(
capture_id uuid,
first_seen timestamp,
l7_conversation_count int,
last_seen timestamp,
processed timestamp,
reassembler_addr text,
uri text,
PRIMARY KEY (capture_id)
)
```

Listing 1.2: L7Conversation table

```
CREATE TABLE
ntpac.l7conversation(
capture_id uuid,
dst ip_endpoint,
first_seen timestamp,
last_seen timestamp,
pdus *<list<*<l7pdu>>>,
protocol_type int,
reass_addr text,
src ip_endpoint,
PRIMARY KEY
((captureid, reass_addr),
  lastseen))
```

Fig. 6: This listing shows the CSQL description of data tables created to store meta information about the capture and L7 conversations. *Capture* table stores record generated by each *Reassember* nodes that participate in data processing, thus, the whole information must be retrieved as an aggregate. The *L7Conversation* table stores a record for each completely tracked L7 conversation. It also contains a list of *L7PDU* that put together contains reassembled conversation and are used for further processing and extraction of data from application protocols.

Analysis In the *second stage*, a subset of reconstructed L7 conversations is retrieved from the distributed database (by using manual or automatic selection with specified rules) and delivered to the *Snooper* nodes. They identify used application protocol and use proper application protocol dissector module to extract data from the L7 conversation. Extracted data are stored back into the distributed database.

Each instance of a particular node acts as an individual actor in the system, communicating with other actors by message passing. Thanks to this design, we can distribute the computation across multiple machines maintaining the scalability.

6 Preliminary Evaluation

This section provides a brief overview of our designated testing deployments accompanied with testing scenarios. We describe the differences to other solutions like IDS/IPS systems that may on the first glance does the similar processing but as shown by Table 5 they primary focus quite different.

6.1 Processing Performance and Scalability

We build our evaluation based on a *scalability* of our solution instead of total throughput. We have developed a proof of concept solution that is not a production-ready implementation with performance optimization at each level of the architecture, but an experimental piece of software that gives us insight into the distributed processing of network communication for forensic purposes. We do measure total performance based on total throughput, but we use it for comparison of scalability factor. We are confident that maintaining designed architecture and reimplementation in some much lover level programming language like Rust, C++, Go instead of C#, we would obtain a much faster solution. Nevertheless, we have chosen C# because it can be used for rapid development and multiplatform deployment as well in combination with .NET Core 2.1 runtime.

We tested two test-cases. The first one, how fast is captured traffic processed on a single machine under one process. This test-case shows total throughput of our processing algorithms 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 count of nodes.

The second test-case revels scalability of our solution. We are testing the cooperation of several nodes in the distributed computation of capture traffic processing. The first kind of node *Load balancer* is loading a PCAP file and distributing its content to *Reassembler* nodes. With the increasing count of load balancing and reassembling nodes, we show the scalability that our PoC solution is capable. To separate the impact of persistence storage, we provide one set of results that does not store its results, i.e., conversations are discarded after they processed, and the second result set that does the full processing and storage in Cassandra database.

We chose to test our solution in three different environments:

– **BPI-R2** cluster that we have built. It consists of 7 router boards equipped with Quad-core ARM Cortex-A7 CPU (4 physical cores running at 1.3GHz), 2 GB RAM, two pcs of gigabit network adapters. This cluster interconnected according to our custom topology described in Section 5.2. Operating system is custom Linux build with 4.4.70 kernel.
– **Desktop computer** in a laboratory environment. The cluster consists of 20 computers equipped with i5-3570K CPU (4 physical cores running at 3.40GHz), 8 GB RAM and two pcs of gigabit network adapters and 7200 rpm hard-drives. The operating system is CentOS 7 with 3.10 kernel.

- **Google Cloud Platform** *Compute Engine* with 12 *n1-highcpu-4* instances equipped with Xeon CPU (2 physical cores, 4 logical cores running at 2.60GHz), 8 GB RAM and 10 Gbps network adapters. The operating system is Ubuntu 16.04.2 with the 4.15.0-36 kernel.

We have used testing PCAP file with the following characteristics – file length: 1024 MB; file format: libpcap/tcpdump; encapsulation: Ethernet; capture's time span: 02:10:31; packets: $1,306,262$; average packet size: 767 B; L7 conversations: 34764. We choose this PCAP file because of its good representative features. It was captured on a live network[3]. The capture file size is sufficient to reduce communication overhead to a negligible portion in the initialization phase and allows us to run all designated testing cases in available time. Usage of a larger capture file does not make measurement more accurate but only prolongs it.

We have conducted a series of experiments described above. Each experiment was repeated 10-times, and resulting speed was averaged. Because of the non-deterministic behavior of the distributed system and internal operating system processes, the standard deviation was in the range of $5-10\%$. The standard deviation together with passing the optimal ratio of load balancing and reassembling nodes may cause a drop of performance, as seen in the Figure 2 R:6$\rightarrow$8, LB:1 617 Mbps$\rightarrow$615 Mbps.

In Table 2, we compare performance and scalability measured on the desktop computer. The raw throughput that discards processed data in the left table to be compared with the full processing and storing of L7 conversations in the right table. For visual comparison, we provide Figure 8 to show scalability trend. Performance increases with the increasing number of reassembling nodes up to a point when one load balancer cannot satisfy reassemblers' processing speed, and

[3] We are willing to share it upon a written email request.

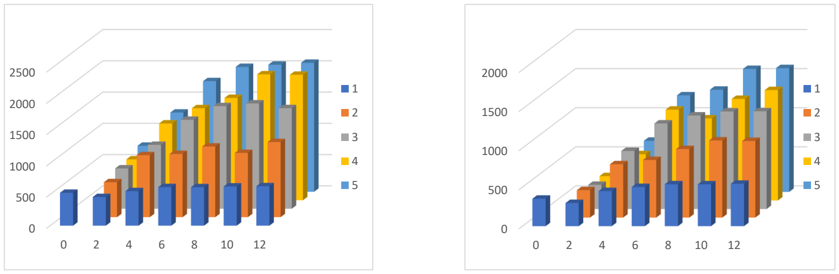
throughput stops increasing. Adding more load balancing nodes increase utilization of the whole cluster up until the reassembling nodes are fully saturated, and processing speed once again stops increasing. Incrementing both counts of load balancers and reassemblers continues to shows an increase in total throughput and scalability of our PoC solution. To compare with the performance of the only one node processing with the distributed version, see the speed on Figure 2 R:0, LB:1.

Consequently, we repeated this same series of experiments on Google Cloud Platform (GCP), Compute Engine (CE) vitalization solution. We were interested to see, whether it is the difference in performance on networks with higher throughput or performance is limited by processes in our application. See Table 3 and Figure 9b for detailed results. We can observe that with the current *load balancer* implementation, larger network throughput does not provide additional performance. We can conclude that current implementation is limited more by the number of CPU cores and frequency than network speed. The less amount of test-cases in the case of GCP occurred because Compute Engine's limitation to run more than 48 CPUs with our trial account.

To conclude our testing scenario and confirm our findings regarding bottlenecks in our implementation, we have conducted last performance measurements on low-cost ARM based devices Banana PI R2. Table 4 shows performance results but limited by cluster architecture to only one reassembling node. We can observe the same trend that occurs in both previous measurements Table 2 and Table 3.

6.2 Function Comparison

In Table 5, we compare various related tools to the presented system according to the several criteria important for the network forensics.



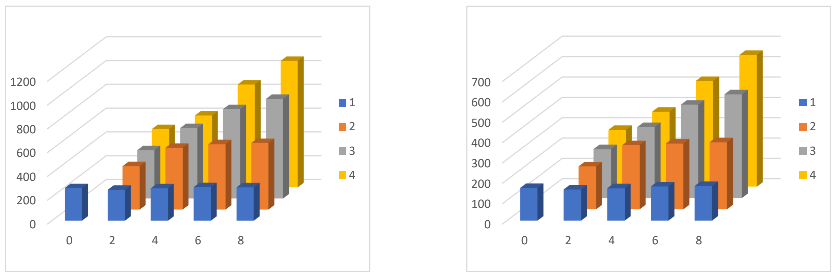
Metadata discarded   Metadata stored in Cassandra

Fig. 8: Performance measurements conducted on **laboratory computers** equipped with i5-3570K CPU (4 physical cores running at 3.40GHz), 8 GB RAM, two pcs of gigabit network adapters and 7200 rpm hard-drives. The x axis denotes count of *reassembling* nodes, z axis denotes count of *load balancing* nodes. The total processing speed represents the y axis.

LB \ R	**0**	**2**	**4**	**6**	**8**	**10**	**12**
1	526	458	552	617	615	627	631
2		560	992	1011	1130	1028	1202
3		645	1021	1422	1641	1684	1609
4		654	1228	1472	1637	2014	2007
5		736	1267	1769	1997	2033	2063

LB \ R	**0**	**2**	**4**	**6**	**8**	**10**	**12**
1	351	296	449	501	536	535	542
2		351	682	737	877	987	980
3		309	743	1094	1196	1247	1250
4		311	592	1162	1050	1299	1412
5		305	654	1235	1307	1573	1583

Table 2: Performance measurements conducted on **laboratory computers** equipped with i5-3570K CPU (4 physical cores running at 3.40GHz), 8 GB RAM, two pcs of gigabit network adapters and 7200 rpm hard-drives. The left table shows the absolute throughput of NTPAC PoC solution for benchmarking purposes to be compared with the right table that shows throughput when metadata is stored in the distributed Cassandra database (using R Cassandra nodes or 1 in the case of single node processing). Values are denoted in *Mbps*. The combination of one *load balancer* with zero *reassemblers* denotes single node processing inside one CLR.



Metadata discarded Metadata stored in Cassandra

Fig. 10: Performance measurements conducted on **Google Cloud Platform** – Compute Engine with 12 *n1-highcpu-4* instances equipped with Xeon CPU (2 physical cores, 4 logical cores running at 2.60GHz), 8 GB RAM and 10 Gbps network adapters. The operating system is Ubuntu 16.04.2 with the 4.15.0-36 kernel. The x axis denotes count of *reassembling* nodes, z axis denotes count of *load balancing* nodes. The total processing speed represents the y axis.

LB \ R	**0**	**2**	**4**	**6**	**8**
1	274	260	272	282	280
2		364	519	549	559
3		405	591	750	836
4		489	603	864	1062

LB \ R	**0**	**2**	**4**	**6**	**8**
1	160	153	159	168	170
2		211	315	323	329
3		240	348	458	509
4		279	368	519	647

Table 3: Performance measurements conducted on **Google Cloud Platform** – Compute Engine with 12 *n1-highcpu-4* instances equipped with Xeon CPU (2 physical cores, 4 logical cores running at 2.60GHz), 8 GB RAM and 10 Gbps network adapters. The operating system is Ubuntu 16.04.2 with the 4.15.0-36 kernel. The left table shows the absolute throughput of NTPAC PoC solution for benchmarking purposes to be compared with the right table that shows throughput when metadata is stored in the distributed Cassandra database (using R Cassandra nodes or 1 in the case of single node processing). Values are denoted in *Mbps*. The combination of one *load balancer* with zero *reassemblers* denotes single node processing inside one CLR.

LB \ R	**0**	**1**	**2**	**3**	**4**
1	49	32	65	68	72

LB \ R	**0**	**1**	**2**	**3**	**4**
1	32	19	36	41	45

Table 4: Performance measurements conducted **Banana PI R2 cluster** equipped with Quad-core ARM Cortex-A7 CPU (4 physical cores running at 1.3GHz), 2 GB RAM, two pcs of gigabit network adapters. Router boards were interconnected according to our custom topology described in Section 5.2. The left table shows the throughput of NTPAC PoC solution for benchmarking purposes to be compared with the right table that shows throughput when metadata is stored in the distributed Cassandra database (using exactly 2 Cassandra nodes). Values are denoted in *Mbps*. The combination of one *load balancer* with zero *reassemblers* denotes single node processing inside one CLR.

- **Live data capture** is an ability to process network packets directly captured on network interface instead of PCAP file.
- **Supported capture files** is a range of PCAP file formats supported by an application.
- **IPv6** defines whether the application can process IPv6 traffic.
- **Encapsulation protocols** defines a range of encapsulation protocols that the application can decapsulate data.
- **OS Fingerprinting** defines an ability to deduce used operating system and version.
- **Application Protocol Identification** is a range of mechanisms used to identify application protocol used in tracked conversation.
- **Applications Identification** is an ability to identify the application that communicated, e.g., Firefox vs. Chrome.
- **Exported application protocols** is a range of application protocols that the application can extract information.
- **Credentials Extraction** is an ability to obtain credentials from application traffic if it contains them.

- **Incomplete or malformed communication** is an ability to process even malformed communication and recover from failure when some part of traffic is missing or is malformed.
- **Processing speed** defines an approximate speed of traffic processing.
- **Multiple data sources** marks an application able to process multiple data sources at once.
- **Advanced analytic views** shows whether the application supports some advanced visualization of processed data.
- **Persistent storage** defines a range of possible persistent storage providers implemented in the application.
- **Querying** defines how the data are queried after processing.
- **PCAP-over-IP** defines an ability to process data that are captured on remote point and encapsulated in IP tunnel and send to the processing server.

We continue our work on *Netfox Detective* and utilize experience gained to create a tool with its broad functionality but in the distributed environment. With the *Netfox Detective*, we have reached limits what we can extract from the captured communication on a single machine. In time, we will extend the *NTPAC* PoC solution with all of the *Netfox Detective* functionality and many more new features.

7 Conclusion

We have proposed the system for distributed real-time forensic network traffic analysis at high-speed networks. The system design follows an actor model that scales nicely and can run on a single machine as well as on a computing cluster. The primary goal of the system is to provide a scalable platform for on-line processing of network communication, but it can also be used as an off-line tool for processing large PCAP files. We compared the prototype implementation

Tool / Feature	**NTPAC**	**Netfox Detective**	**Network Miner**	**XPlico**	**PyFlag**	**Moloch**	**VAST**
Live data capture	YES	NO	YES	YES	NO	YES	YES
Supported capture files	libPcap, pcap-ng	libPcap, pcap-ng, mnm	libPcap, pcap-ng	libPcap	libPcap	libpcap	libpcap
IPv6	YES	YES	YES	YES	NO	YES	YES
Encapsulation protocols	Not implemented yet	GRE, GSE	GRE, 802.1Q, PPPoE, LLMNR, VXLAN, OpenFlow, SOCKS, MPLS and EoMPLS	L2TP, VLAN, PPP	NO	NO	NO
OS Fingerprinting	Not implemented yet	YES (using typical applications)	YES	NO	NO	NO	Bro
Application Protocol Identification	Not implemented yet	SPID, NBAR, ESPI, Bayessian, Random Forests	SPID, PIPI	PIPI	NO	NO	Bro
Applications Identification	Not implemented yet	YES	NO	NO	NO	NO	NO
Exported application protocols	Not implemented yet	HTTP, SSL/TLS, MAFF. XMPP, YMSG, OSCAR, Facebook Messanger, Hangouts, Twitter, XChat, IMAP, POP3, SMTP, Gmail, Yahoo, RTP, SIP, Minecraft, Warcraft, Facebook, Stratum, DNS, FTP, SPDY, MQTT	FTP, TFTP, HTTP, SMB, SMB2, SMTP, POP3, IMAP, YouTube	HTTP, POP3, SMTP, IMAP, SIP, RTP, SDP, FTP, DNS, IRC, IPP, PJL, MMS, SLL	DNS, HTTP, MSN, Gmail	DHCP, DNS, HTTP, IRC, KRB5, LDAP, MYSQL, QUIC, RADIUS, SMB, SOCKS, SSH, TLS	NO
Credentials Extraction	Not implemented yet	Facebook, IMAP, SMTP, POP3	SMTP, HTTP Digest Autentization	NO	NO	HTTP	NO
Incomplete or malformed communication	TCP data loss, IPv4 fragmentation	TCP data loss, IPv4 fragmentation	?	NO	NO	?	NO
Processing speed	Scales – Gbps, see figures 7a, 7b	264 Mbps	11.92-18.48 Mbps	?	?	Scales – Gbps	Scales – Gbps
Multiple data sources	YES	YES	NO	YES	NO	YES	YES
Advanced analytic views	NO	YES	NO	YES	NO	YES	NO
Persistent storage	SQL, Cassandra, ArangoDB	SQL	CSV / Excel / XML / CASE / JSON-LD	SQL	VFS	ElasticSearch	Hash and Tree Indexes
Querying	3-rd party tools on database	3-rd party tools on SQL database	keyword search	3-rd party tools on SQL database	YES	YES	YES
PCAP-over-IP	YES	NO	YES	YES	NO	NO	NO

Table 5: Network forensic tools capability overview. Open-source or freeware tools were selected for comparison. Details were gathered from propagation materials, documentation, and study of source code. It is probable that some features will change over time.

of the system with relevant existing tools and showed the results of the performance evaluation. Prototype evaluation was done in different environments, e.g., cloud environment, laboratory environment, ARM-based low power environment, demonstrating that the proposed solution can be deployed in multiple ways according to the specific needs. In our settings, we were able to achieve a processing speed necessary to capture and analyze full duplex 1 Gbps network communication. Conducted experiments demonstrated the feasibility of the proposed approach although much work remains for implementation optimization and further tuning of various parameters that influence the achievable processing speed.

References

1. M. Vallentin, *Scalable Network Forensics.* PhD thesis, UC Berkeley, 2016.
2. E. S. Pilli, R. C. Joshi, and R. Niyogi, “Network forensic frameworks: Survey and research challenges,” *digital investigation*, vol. 7, no. 1-2, pp. 14–27, 2010.
3. S. Rekhis, J. Krichene, and N. Boudriga, “Digfornet: digital forensic in networking,” in *IFIP International Information Security Conference*, pp. 637–651, Springer, 2008.
4. A. Almulhem and I. Traore, “Experience with engineering a network forensics system,” in *International Conference on Information Networking*, pp. 62–71, Springer, 2005.
5. 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, p. 4, 2008.
6. N. L. Beebe and J. G. Clark, “A hierarchical, objectives-based framework for the digital investigations process,” *Digital Investigation*, vol. 2, no. 2, pp. 147–167, 2005.

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

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

9. C. Hewitt, P. Bishop, and R. Steiger, “Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence,” in *Advance Papers of the Conference*, vol. 3, p. 235, Stanford Research Institute, 1973.

10. A. Lukashin, L. Laboshin, V. Zaborovsky, and V. Mulukha, “Distributed packet trace processing method for information security analysis,” in *Internet of Things, Smart Spaces, and Next Generation Networks and Systems* (S. Balandin, S. Andreev, and Y. Koucheryavy, eds.), (Cham), pp. 535–543, Springer International Publishing, 2014.

11. M. Wullink, G. C. M. Moura, M. Mller, and C. Hesselman, “Entrada: A high-performance network traffic data streaming warehouse,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 913–918, April 2016.

12. M. Aupetit, Y. Zhauniarovich, G. Vasiliadis, M. Dacier, and Y. Boshmaf, “Visualization of actionable knowledge to mitigate drdos attacks,” in *2016 IEEE Symposium on Visualization for Cyber Security (VizSec)*, pp. 1–8, Oct 2016.

13. N. Promrit and A. Mingkhwan, “Traffic flow classification and visualization for network forensic analysis,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pp. 358–364, March 2015.

14. 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, pp. 177–182, IEEE, 2005.

15. W. Ren, "On a reference model of distributed cooperative network, forensics system.," in *iiWAS*, 2004.

16. 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, pp. 651–656, IEEE, 2007.

17. 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, pp. 214–235, 2016.

18. M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The nids cluster: Scalable, stateful network intrusion detection on commodity hardware," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 107–126, Springer, 2007.

19. M. Vallentin, V. Paxson, and R. Sommer, "Vast: A unified platform for interactive network forensics.," in *NSDI*, pp. 345–362, 2016.

20. L. Polčák, R. Hranický, and T. Martínek, "On identities in modern networks," *The Journal of Digital Forensics, Security and Law*, vol. 2014, no. 2, pp. 9–22, 2014.

21. L. Polčák and B. Franková, "Clock-skew-based computer identification: Traps and pitfalls," *Journal of Universal Computer Science*, vol. 21, no. 9, pp. 1210–1233, 2015.

22. J. Pluskal, O. Lichtner, and O. Rysavy, "Traffic classification and application identification in network forensics," in *IFIP International Conference on Digital Forensics*, pp. 161–181, Springer, 2018.

23. J. Pluskal, "Framework for captured network communication processing," Master's thesis, Vysoké učení technické v Brně, Fakulta informačních technologií, 2014.

24. P. Matousek, J. Pluskal, O. Rysavy, V. Vesely, M. Kmet, F. Karpisek, and M. Vymlatil, "Advanced techniques for reconstruction of incomplete network data," in *International Conference on Digital Forensics and Cyber Crime*, pp. 69–84, Springer, 2015.