Online Non-Intrusive Diagnosis of One-Way RTP Faults in VoIP Networks Using Cooperation

A. Amirante, S. P. Romano
Computer Science Department
University of Napoli Federico II, Napoli, Italy
{alessandro.amirante, spromano}@unina.it

K. H. Kim, H. Schulzrinne
Department of Computer Science
Columbia University, New York (NY), USA
{khkim, hgs}@cs.columbia.edu

ABSTRACT

We address the well-known issue of one-way RTP flows in VoIP communications. We investigate the main causes that usually lead to this type of fault, and we propose a methodology allowing for their automated online detection and diagnosis. The envisaged approach exploits node cooperation and is based on a more general framework for network faults diagnosis called *DYSWIS* (Do You See What I See). As most of the problems associated with one-way RTP can be ascribed to the presence of NAT elements along the communication path, one of the key features of the proposed methodology resides in the capability to detect such type of devices. Besides, another important aspect of this work is that the diagnosis is non-intrusive, meaning that the whole process is based on the passive observation of flowing packets, and on silent active probing that is transparent to the users. In this way, we also avoid the possibility of being classified as SPIT (SPam over Internet Telephony). We provide a thorough description of the various steps the diagnosing process goes through, together with some implementation details as well as the results of the validation process.

Categories and Subject Descriptors

C.2.3 [**Computer-Communications Networks**]: Network Operations—*Network management, Network monitoring*;
D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*;
D.2.5 [**Software Engineering**]: Testing and Debugging—*Diagnostics*

General Terms

Algorithms, Management, Experimentation

Keywords

One-way RTP, DYSWIS, VoIP faults diagnosis, Nodes cooperation

IPTComm 2010, 2-3 August, 2010 Munich, Germany

1. INTRODUCTION

We tackle the challenge of automatically detecting faults occurring in SIP-based Voice over IP (VoIP) networks. We first illustrate the most common fault scenarios that characterize a complex communication infrastructure comprising entities which handle end-to-end data, both in the control plane (proxies, back-to-back user agents, etc.) and in the data plane (NATs, Application Level Gateways, relays, etc.). We then focus on one of the most critical faults that can happen when trying to setup a multimedia communication in a SIP [1] network, namely the impossibility of creating a real-time bi-directional communication channel between a caller and a callee. Such fault, which is known in the literature as the "one-way RTP issue", can be due to a number of different yet often interdependent causes and represents one of the most cumbersome problems VoIP architects have to face when deploying and maintaining their networks. We deal with the above mentioned issue by leveraging a novel peer-to-peer architecture for network diagnosis, called DYSWIS (Do You See What I See) [2], which has been conceived at the outset as an extensible infrastructure for non-intrusive, cooperation-based detection of network faults. We will describe how we extended DYSWIS in order to let it support both the SIP and the RTP [3] protocol state machines. The paper embraces an engineering approach. It delves into some of the details of the most notable implementation choices characterizing our contribution. It also illustrates how the most common real-world scenarios which suffer from the one-way RTP issue can be addressed with the approach we propose. At the best of our knowledge, no other approaches addressing the one-way RTP problem have been proposed as yet. The paper is structured as follows. In Section 2 we report the main causes of the problem. In Section 3, we first introduce the DYSWIS architecture as a framework for automated network faults diagnosis; then we show how we added to it support for the SIP, SDP [4] and RTP protocols. The section explains how we devised an approach based on passive tests and silent active probing. Section 4 contains some implementation details, while in Section 5 we show the results of our validation process. Finally, Section 6 concludes the paper by summarizing the main achievements while also presenting the main directions of future work.

2. ONE-WAY MEDIA FLOWS: A WELL KNOWN ISSUE

The problem of one-way RTP flows is very common in VoIP communications. In this section, we provide a classification of the causes that lead to such kind of fault, by

splitting them into four main categories.

2.1 Configuration problems

Into this category fall all the problems that can be ascribed to some error in the configuration of the machine hosting a User Agent (UA). First of all, there are possible oversights in the configuration of the UA itself (e.g., wrong audio capture device selected). Then, we have network interface configuration errors, that are quite common especially in multi-homed systems. In fact, it can happen to see RTP packets being received and sent on two different network interfaces, for example on machines having both a wired and wireless connection up (this is not unlikely on Unix-based systems, and is usually due to the configuration stored in the `/etc/hosts` file). The presence of software firewalls not properly configured can also cause one-way media flows: for example, if we want both audio and video to be involved in the call, it would not be sufficient to open a couple of ports, since each call leg consumes two ports (one for RTP and the other for RTCP). Finally, we also classify IP address conflicts in the network as a local configuration problem.

As we will see in Section 3.3, it is easy to diagnose problems falling into this category.

2.2 NAT-related problems

Most of the factors that can cause one-way media flows fall into this category and are related to the presence of NAT elements along the communication path. Several NAT traversal solutions have been proposed by the Internet Engineering Task Force (IETF), namely the STUN *(Session Traversal Utilities for NAT)* [5], TURN *(Traversal Using Relay NAT)* [6] and ICE *(Interactive Connectivity Establishment)* [7] protocols and the *Application Level Gateway* (ALG) and *RTP proxy* elements. If no such solution is employed, the User Agent is unable to receive RTP packets. Even worse, even if a NAT traversal technique is employed, it can happen that the "natted" party is anyhow unable to see incoming packets. This is the case of the most widespread NAT traversal solution: the STUN protocol. STUN is actually helpful in a number of cases; though, it is useless when a User Agent is behind a *symmetric NAT*[1], in which case it experiences one-way media flows. Furthermore, one more scenario where the STUN usage does not avoid one-way RTP flows is when both the caller and the callee happen to be in the same subnet, since a lot of NAT elements discard packets received from the private network and destined to their own public IP address. The last situation can happen also if the STUN protocol is not employed, but the NAT box has built-in SIP Application Level Gateway (ALG) functionality. This is becoming very common, as many of today's commercial routers implement such feature. Unfortunately, poorly implemented ALGs are quite common, too, and in some cases they can be the cause of the problem rather than the solution[2]. Finally, very often the same device handles both NAT and firewall functions; in these cases, port blocking issues have to be taken into account.

2.3 Node crash problems

The sudden crash of a network node also causes the inability to receive RTP packets. We remark that the crashed node could be neither the caller nor the called party, but a possible RTP proxy that belongs to the media path.

2.4 Codec mismatch

A lot of SIP clients offer the possibility to select only a subset of media codecs, among the ones supported. Unfortunately, sometimes this choice is not reflected in the capabilities offered in the SDP, so it can happen that the result of the media negotiation is a codec that has been disabled. As a consequence of this, one of the parties involved in the call would not hear the voice or see the video of the other, even if it is actually receiving the corresponding RTP packets. We report this kind of problem just for the sake of completeness, as in this case we are not experiencing one-way media flows since RTP packets flow in both directions. Consequently, our work does not address this issue.

3. DIAGNOSIS: THE DYSWIS APPROACH

As previously introduced, this work is based upon a network diagnosis architecture that is currently under development at Columbia University, called DYSWIS[3], which leverages distributed resources in the network, called *DYSWIS nodes*, as multiple vantage points from which to obtain a global view of the state of the network itself. Each DYSWIS node is capable to detect fault occurrences and perform or request diagnostic tests, and has analytical capabilities to make inferences about the corresponding causes.

3.1 Architecture overview

From a very high-level perspective, a DYSWIS node tries to isolate the cause of a failure by asking questions to peer nodes and performing active tests. The architecture is depicted in Fig. 1; in the following, we do not dwell on architectural details, since these are beyond the scope of this work. We just remark that a modular approach is adopted, in order to allow support for new protocols in an easy fashion. Specifically, each time a new protocol has to be added, protocol-specific *Detect* and *Session* modules have to be implemented, together with a representation of the fault. Furthermore, new tests and probes have to be implemented, too, when required. Finally, the rules that drive the diagnosis process have to be written. In fact, each DYSWIS node relies on a rule engine that triggers the invocation of the probes on the basis of the type of fault and of the result of previous tests.

As probing functions need to be executed on remote nodes that have specific characteristics, a criterion to identify such nodes is needed, as well as a communication protocol. For example, we could be interested in selecting a peer that has a public IP address, rather than a node that belongs to a given subnet. At the time of writing, remote peers are discovered by means of a centralized repository where each node registers all its useful information as soon as it becomes available. However, an alternative approach, exploiting a *Distributed Hash Table* (DHT), has been implemented in order to better fulfill scalability requirements.

In order to communicate among each other, as well as to convey information about detected failures and request a probe to be run, the DYSWIS nodes exploit a request-response protocol. For further details about how this func-

[1]For a thorough description of the different types of NAT, the reader can refer to [5].

[2]See `www.voip-info.org/wiki/view/Routers+SIP+ALG`.

[3]See `http://www.cs.columbia.edu/irt/project/dyswis/`

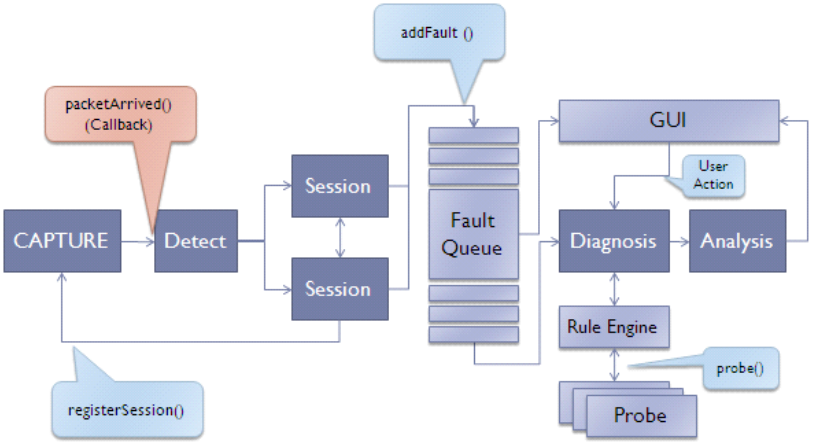

Figure 1: DYSWIS architecture

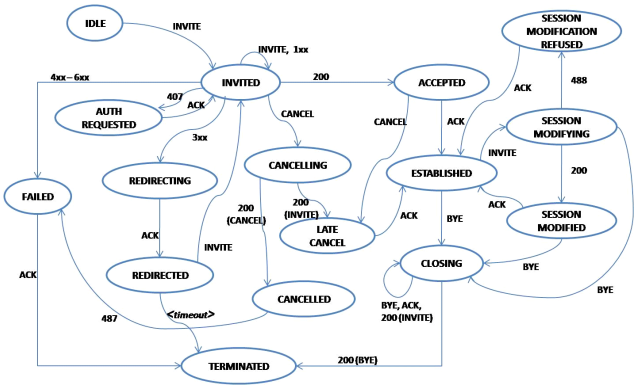

Figure 2: SIP finite state machine

tionality is provided, refer to Section 4, which discusses implementation aspects.

Finally, when the probing phase is completed, the *Analysis* module produces the final response and presents it to the user.

3.2 Adding SIP/RTP diagnosing features to the framework

For the purpose of this work, we added support for both SIP and RTP to the DYSWIS architecture. The detection part is simply performed by "sniffing" packets on the SIP standard ports 5060 and 5061, as well as on the media ports indicated by the SDP's *m-lines*. In Fig. 2, instead, we show the SIP Finite State Machine (FSM) we devised for the session module. We note that the detection process is based on the observation of packets flowing through a host's network interface, so it is a bit different from the classical SIP state machine.

The creation of a new SIP session is triggered by a new `INVITE` message and, within a SIP session, one or more RTP sessions could be created, each one representing a single medium. Specifically, the creation of an RTP session starts with the first SIP message that carries an SDP body (that could be either an `INVITE` or a `200`) and is completed as soon as the second SDP-carrying message is seen (a `200` or an `ACK`, respectively). An RTP session could also be created or modified by re-`INVITE` messages; we took into account such possibility since it is of key importance when both parties of the call make use of the ICE protocol. When the ICE negotiation ends, in fact, the caller sends a re-`INVITE` to update the media-specific IP address and port.

3.3 Proposed diagnosis flow

As already stated, the goal of this work is to diagnose one-way RTP faults by identifying the source of the problem among the ones presented in Section 2. We represent the whole process by means of a flow chart (see Fig. 3) that applies to both UAC and UAS scenarios. It takes into account all the scenarios that can lead to one-way media flows and, even if we will not thoroughly analyze all the possible branches, we provide, in Section 5, some reference scenarios that will help the reader understanding our work. In the diagram, the "local" adjective is used to identify elements or functionality that belong to the same subnet of the DYSWIS node which experienced the fault, while "remote" elements or functionality belong to the same subnet of the other party. We also make a distinction between *tests* and *probes*: the former class only exploits local information, while the latter plays an active role by introducing packets into the network. Finally, we explicitly mark the probes that need the help of a cooperating node in order to be performed.

We observe that it is not always possible to exactly identify the cause of the problem. The capability of making an accurate diagnosis, in fact, strictly depends on the complexity of the network topology under consideration and on actual availability of "remote" DYSWIS nodes, too. The ability to identify such nodes is of key importance and is far from trivial. In fact, when a remote node belongs to a private network environment (i.e., the remote party of the call is natted), its IP address is not helpful for our purpose. Even the node's *reflexive address*[4] can be not helpful in cases where hierarchies of NATs are involved, like the one depicted

[4]From RFC 5389: the *reflexive transport address* is the public IP address and port created by the NAT closest to the server (i.e., the most external NAT)

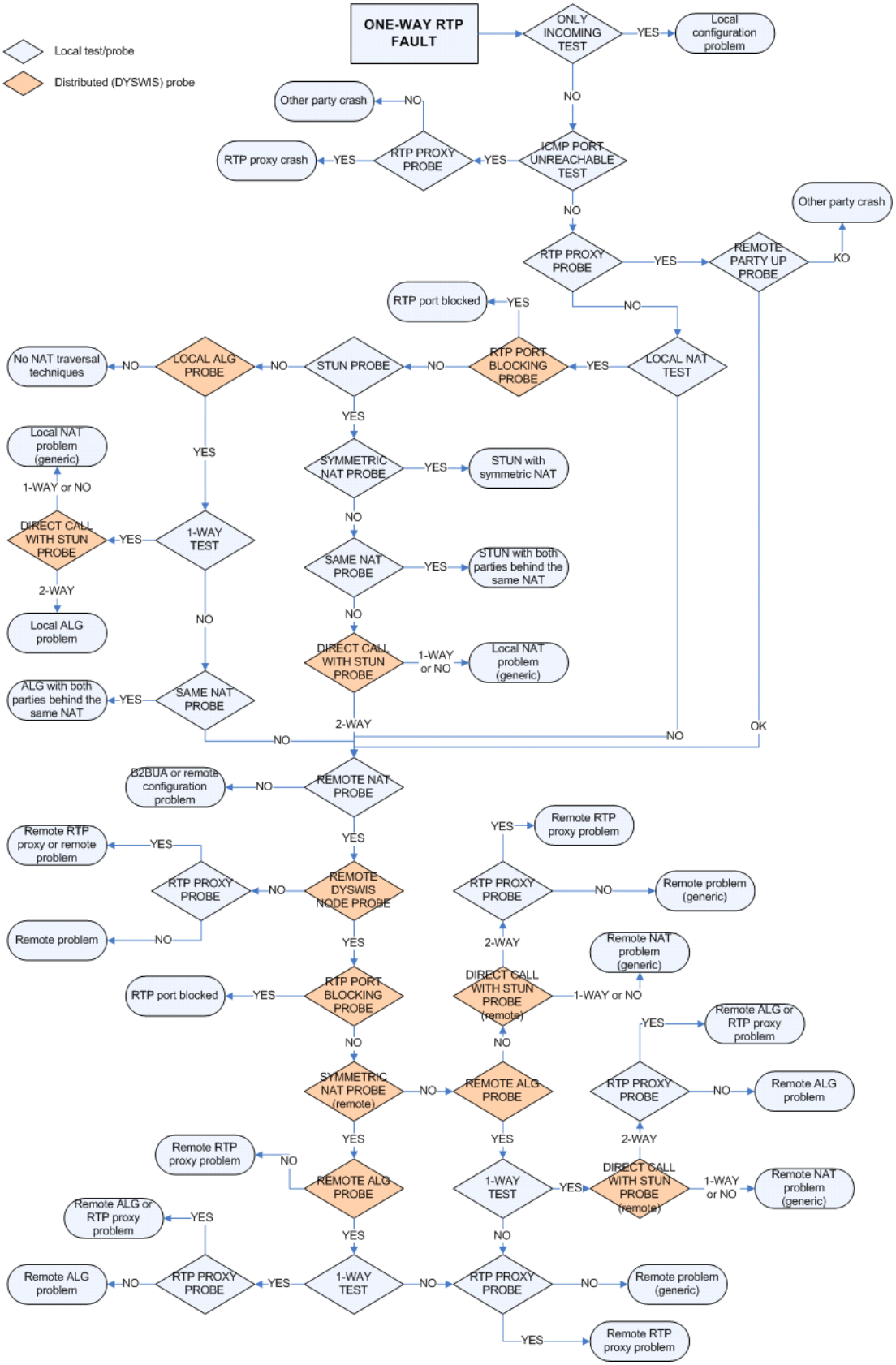

Figure 3: Flow diagram representing the whole diagnosis process

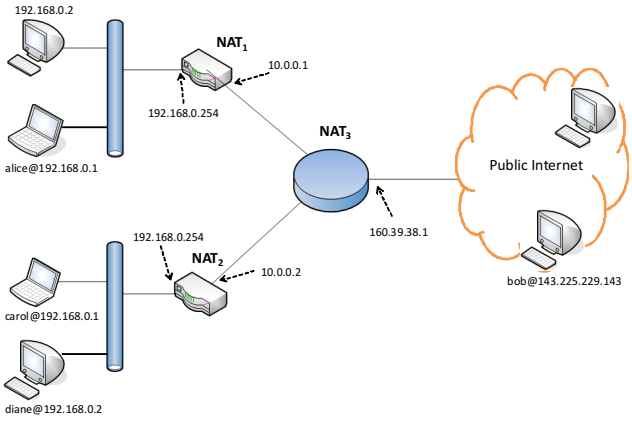

Figure 4: An example of NAT hierarchy that complicates the identification of "remote" peers

in Fig. 4. We will explain in the following subsection how we coped with this issue.

It is worth remarking that one of our goals was to carry out diagnosis in a non-intrusive way. In other words, we did not want to allocate new "real" SIP call towards the caller or the callee, because they would be annoying and could be easily classified as SPIT. Instead, a DYSWIS node tries to collect as much information as possible: (i) from the observation of flowing packets, and (ii) with silent active probes (e.g., a STUN transaction to determine its own reflexive address). When an actual SIP session needs to be set up for diagnosing purposes, it is established between two DYSWIS nodes without using the default SIP ports, so that possible softphones running on those machines would not be alerted.

3.4 Description of tests and probes

In this subsection we provide a thorough description of the probing functions we designed and implemented. These probes allow us to test the network environments close to either the caller or the callee (e.g., NATs, ALGs), as well as possible external nodes, like RTP proxies.

3.4.1 Only incoming test

This is an easy test that checks whether the detected one-way RTP flow is only incoming or only outgoing.

3.4.2 ICMP port unreachable test

Here, we check if there are incoming *ICMP port unreachable* packets, which would be a clear symptom that the process that was supposed to receive data is not active. Herein, we refer to this situation as a node crash.

3.4.3 RTP proxy probe

This probe determines if there is an RTP proxy along the media path. An RTP proxy could be manually configured in the SIP client (e.g., a TURN server) or its usage might have been forced by a SIP proxy by modifying the SDP payload of the messages it forwards. We take into consideration both cases. For the former, we compare the IP address contained in the `Contact` header of an **incoming** message with the SDP's *c-line* **of the same message**: if they are different, we can presume that there is an RTP proxy. As to the latter case, instead, we inspect **outgoing** SIP packets, checking if the IP address contained in the SDP's *c-line* is different from both the local interface address and the reflexive IP address that is retrieved by means of a STUN transaction.

3.4.4 Remote party up probe

Whenever an RTP proxy is employed, we are not capable to detect a possible crash of the remote node, since we would not receive any ICMP packet. In these cases, we check the availability of the remote party by sending a SIP `OPTIONS` message to it. Such message is sent through all the SIP proxies included in the signaling path, if any, in order to cross a possible remote NAT, making use of the `Record-route` and `Route` SIP headers.

3.4.5 Local NAT test

This test determines if the local node (i.e., the node which experienced the fault) is behind a NAT by checking if the local interface has a private IP address.

3.4.6 RTP port blocking

This probe verifies that the port number used for the RTP flow is not being blocked by a possible firewall running on the NAT box.

3.4.7 STUN probe

Here we determine if the local node is making use of the STUN protocol. This probe consists in a STUN transaction to learn the local reflexive IP address. The result is then checked against the address contained in the SDP's *c-line* of an **outgoing** SIP message.

3.4.8 Local/Remote ALG

This probe consists of a direct call attempt to a public DYSWIS node (i.e., a DYSWIS node that has a public IP address). As long as this call attempt is performed without exploiting any NAT-traversal technique, as well as without the SIP extension for Symmetric Response Routing [8], it lets us detect if the local or remote NAT has built-in Application Level Gateway functionality. In fact, the call attempt would succeed only if the private IP address, inserted by the client in the SIP message, is being modified by the NAT element before forwarding it. As previously said, we do not make use of the standard SIP ports for this call.

3.4.9 Direct call with STUN

This probe differs from the previously described one only because the call attempt employs the STUN protocol.

3.4.10 Same NAT probe

The public (reflexive) IP of the remote party is compared with the local reflexive address: if they match, the two parties are assumed to be behind the same NAT.

3.4.11 Symmetric NAT probe

One functionality offered by the STUN protocol is the possibility to discover which type of NAT (Full Cone, Restricted Cone, Port Restricted Cone or Symmetric) is deployed. We use such feature to determine if there is a symmetric NAT, that, as already introduced, might be the cause of the fault we are trying to diagnose.

3.4.12 Remote NAT probe

One of the main issues we had to face is the detection of remote NAT elements. In other words, we wanted to learn if

the remote party is in a private network environment. Sometimes this is easy because, parsing a received SIP message, we find a private IP address (e.g., it could be in the SIP `Contact`, `From` or `To` headers, or in the SDP's *c-line* or *o-line*). Unfortunately, this depends on the specific implementation of the SIP element: for instance, some clients, when using STUN, put their public address in the SDP's *o-line*, while others do not. Similarly, some ALGs just parse outgoing messages and substitute every occurrence of a private IP, while others perform better thought-out replacements. When we cannot find any occurrence of private IP, we exploit a modified version of the IP traceroute we developed on our own, that sends a SIP `OPTIONS` message gradually increasing the *IP Time-To-Live* value. We send such request towards the public IP address of the remote node and, if we get an ICMP *TTL exceeded* packet whose source address is the original target of our request, it is a clear indication of the presence of a remote NAT element. Otherwise, we could either receive a SIP response (e.g., a `200`) or do not receive any response at all. In the latter case, after having retried to send the message, with the same TTL value, for a couple of times (to take care of possible packet losses), we infer that there is a remote NAT box that is not a *Full Cone*. Consequently, our SIP message is being filtered. Finally, if we receive a response to the `OPTIONS` query, we cannot state there is no NAT along the path, yet. In fact, in the standard specification [9], there is no constraint for a NAT element to decrease the TTL value while forwarding packets. This topic has been discussed a lot on the BEHAVE[5] mailing list of the IETF, where both personal opinions and implementation reports were provided. It turned out that a NAT does not always decrease the TTL of packets received on the public interface, while, for diagnostic reasons, it always decreases it for packets generated in the private environment and forwarded outside. Then, in order to take into account this possibility, when we receive a response to the aforementioned SIP `OPTIONS` query, we check the TTL value of the IP packet and try to infer whether it comes from a end-host or it has been modified by a NAT. This check is performed by considering that host operating systems have distinctive values for the initial TTL. Then, if the packet did not go through a NAT, the received TTL value would be equal to one of such initial TTL values, decreased by the number of "hops" returned by the traceroute. Otherwise, we infer the presence of a NAT. Further details of these OS-specific TTL values can be found in [10].

For the sake of completeness, we report a draft proposal [12] that has been recently submitted to the IETF and that might prove helpful for the NAT detection problem. It introduces a new SIP header field called `Debug` whose purpose is to convey extra debugging information.

3.4.13 Remote DYSWIS node probe

We conclude the description of the probing functions by showing how we realized the selection of a DYSWIS node that belongs to the same subnet of the remote party of the call. As we already said, a selection merely based on the public IP address would not be sufficient whenever there is a hierarchy of NATs. Then, after having selected all the DYSWIS nodes characterized by the same public IP address as the remote party, by means of the criterion described at the beginning of Section 3, we need to verify if one (or more) of them can be exploited for our purposes. We achieve this goal by sending a SIP `INFO` message in broadcast over the LAN. Such `INFO` message has to be sent within the dialog existing between caller and callee, so that, according to the INFO's RFC [11], *"A 481 Call Leg/Transaction Does Not Exist message MUST be sent by a UAS if the INFO request does not match any existing call leg"*. This is achieved by making the node aware of the `To` and `From` tags and of the `Call-ID`, so that it could be able to generate a request within a specific dialog. Therefore, each selected node would receive a non-481 response only if the remote party belongs to its same subnet.

Among all the methods envisaged by the SIP protocol, the only two that MUST[6] send an error response whenever they do not find any existing call leg are `INFO` and `UPDATE`. We chose to exploit the first one because, even if it is not mandatory, it is widely implemented in almost all the clients currently available.

4. IMPLEMENTATION DETAILS

In this section we provide some brief information about the implementation choices. Besides Java, that has been chosen at the outset as the programming language for the whole framework for its well known platform-independence characteristic, the framework exploits the Jess rule engine [13] to control the diagnosis process. Jess uses an enhanced version of the *Rete* algorithm [14] to process rules, making Java software capable to "reason" using knowledge supplied in the form of declarative rules. Consequently, we implemented the whole flow chart presented in Fig. 3 as a set of rules in the Jess scripting language. The example below shows the rules allowing for the detection of a node's crash, when incoming ICMP packets are detected:

```
(defrule MAIN::RTP_ONEWAY
 (declare (auto-focus TRUE)) => (rtp_oneway (fetch FAULT))
)

(deffunction rtp_oneway (?args)
 "one-way RTP diagnosis"

(bind ?result (LocalProbe "RtpOnlyIncomingTest" ?args))(
  if (eq ?result "ok") then
    (bind ?finalresponse "Local configuration problem")
  else then
    (bind ?result (LocalProbe "IcmpDestUnreachTest" ?args))(
    if (eq ?result "ok") then
      (bind ?result (LocalProbe "RtpProxyTest" ?args))(
      if (eq ?result "ok") then
        (bind ?finalresponse "RTP proxy crash")
      else then
        (bind ?finalresponse "Other party crash")
    )
    else then
      ...
```

As to the SIP/SDP functionality, we adopted the JAIN APIs [15] developed by the National Institute of Standards and Technology (NIST).

For the invocation of remote probes on nodes that happen to be in natted environments, we chose to make use of the *udp-invoker* library [16], slightly modifying it in order to fit our needs. More precisely, a remote natted node is contacted by means of a relay agent, as shown in Fig. 5: as

[5]BEHAVE (Behavior Engineering for Hindrance Avoidance) is the working group of the IETF which deals with the behavior of NATs

[6]In the IETF jargon, the capitalized word "MUST" represents an absolute requirement of the specification.

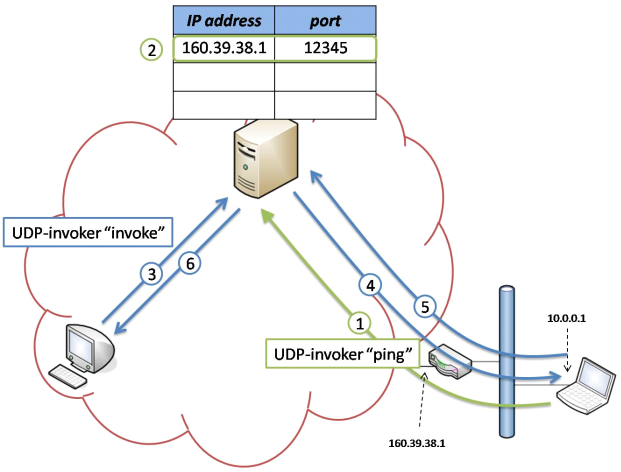

Figure 5: Remote probing functionality of natted nodes leveraging a relay agent

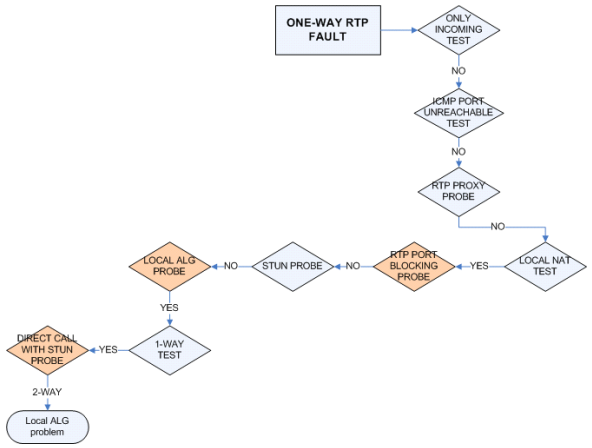

Figure 6: Local ALG problem

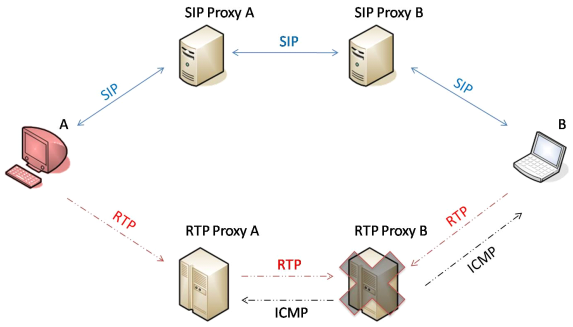

Figure 7: Remote RTP proxy crash: network topology

soon as a DYSWIS node belonging to a private environment becomes available, it sends a *udp-invoker ping* message to the relay agent, which in turn stores the related public IP address and port. Such message is sent periodically, in order to properly refresh the binding in the NAT table. Then, if the probing functionality provided by a private node needs to be exploited, the *invoke* message is sent through the relay agent. We remark that, in such way, we managed to cross any type of NATs. On the other hand, when the peer has a public IP address, the XML-RPC protocol [17] is exploited. Since it uses HTTP as the transport mechanism, it is more reliable than udp-invoker and, in some cases, it helps crossing restrictive local NATs.

Finally, the Jpcap library [18] allowed us to "sniff" packets from the network interfaces and send ad-hoc formatted packets, as well.

5. VALIDATION

In this section we provide the results of our validation. We tested our work with several different SIP clients. Specifically, we exploited the following softphones: *X-Lite* [19] (Windows), *SJPhone* [20] (Windows and Linux), *Ekiga* [21] (Linux) and *PJSIP-UA* [22] (Linux). As SIP and RTP proxies, we used *OpenSIPS* [23] and its *RTPproxy* [24] component, respectively. Finally, we developed our own implementation of a basic SIP ALG, since we could not find any suitable open-source library. With all these components, we set up a distributed testbed between the IRT lab at Columbia University and the COMICS lab at the University of Napoli. For the sake of conciseness, we do not present all the possible diagnosis paths that result from the flow chart in Fig. 3, which nonetheless have all been tested. Instead, we just provide a couple of representative scenarios, which show how the diagnosis process takes place.

5.1 Scenario 1: problem with the local ALG

The first scenario we examine is characterized by the use of an ALG in the local network. We deliberately modified our ALG library in order to induce the one-way RTP fault. Specifically, we let our ALG function modify the *c-line* in the session-level section of the SDP message, without changing the same parameter in the media description section. So, since the session-level parameter is overridden by an analogous one in the media description, if present, the remote party will send its RTP packets to a private, non-routable, IP address.

In Fig. 6 we show a snippet of the whole flow diagram that applies to this situation, whose understanding is quite straightforward. We just clarify the last steps. The call attempted by the *Local ALG probe* can take place, thus revealing the presence of an ALG. Though, the resulting RTP flow is still one-way and this definitely represents a clue that the source of the problem might be the ALG itself. Such conjecture is confirmed by the *Direct call with STUN probe*. In fact, as long as we employ the STUN protocol before placing the call, the ALG does not come into play, since there would be no private IP addresses to replace.

5.2 Scenario 2: remote RTP proxy crash

In this scenario, we suppose that both caller and callee use an RTP proxy. If the proxy used by the remote party crashes, the local DYSWIS node will experience a one-way RTP fault. Furthermore, it will not see any incoming ICMP packet (see Fig. 7).

In Fig. 8 we show the diagnosis steps in this scenario. We are supposing that the remote node is behind a non-symmetric NAT that has no built-in ALG functionality. However, even changing such hypotheses, we are still able to

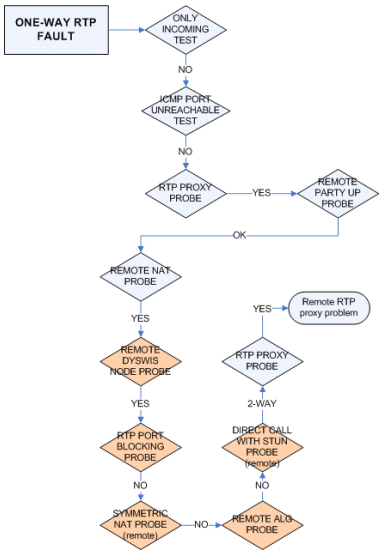

Figure 8: Remote RTP proxy crash: diagnosis flow

identify the cause of the fault. In general, when the diagnosis process involves the remote subnet, the results of the various probing functions allow us to narrow down the set of possible sources of the problem. In this case, we first get ensured that the problem cannot be ascribed to a remote ALG; then, we exclude that it could be somehow related to the remote NAT's behavior, since the SIP+STUN call involves two-way media flows. This brings us to the final verdict. We observe that, in this lucky case, we are able to detect the exact cause of the fault, while in other cases, when the network topology is particularly complex, we are able to narrow down the fault space to two possible choices.

6. CONCLUSIONS AND FUTURE WORK

In this work we dealt with RTP faults in VoIP networks. Specifically, we addressed the well-known problem of one-way media flows, by first introducing the main causes and, then, by proposing a methodology allowing for its online detection and diagnosis. The proposed approach leverages distributed resources in the network that cooperate in order to isolate the source of the fault, as envisaged by the wider framework for network fault diagnosis, called DYSWIS, it is based upon. The diagnosis process is completely transparent to the users and does not generate any unsolicited calls. We showed that most of the times we are able to exactly identify the source of the problem, while, in the worst cases, we manage to narrow down the fault space to two possible choices. We provided the reader with a thorough description of the diagnosis process, also presenting some reference real-world scenarios, in order to ease its understanding. Finally, implementation details about the prototype we realized have been provided, too, together with the results of the validation we conducted.

The framework described in this paper paves the ground to future research challenges. Besides its enrichment with new protocols and new fault scenarios, we see a big potential in the exploitation of the DYSWIS framework for security purposes. For example, as long as we consider an intrusion as a type of network fault, we might follow the DYSWIS approach in order to build a distributed IDS (Intrusion Detection System). In such context, nodes cooperation is also helpful in the reaction/remediation process. Finally, security issues must be faced in order to avoid that the active probing functionality is exploited for bad purposes by malicious users. Then, it is worth providing the framework with intrinsic mechanisms that guarantee its robustness against possible attacks.

7. ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme INSPIRE (FP7/2007-2013) under grant agreement no. 225553.

This work has been carried out with the financial support of Intel Corporation.

8. REFERENCES

[1] J. Rosenberg, H. Schulzrinne et al., *SIP: Session Initiation Protocol*, RFC 3261, June 2002.

[2] V. K. Singh, H. Schulzrinne and K. Miao, *DYSWIS: An Architecture for Automated Diagnosis of Networks*, Network Operations and Management Symposium 2008, April 2008, 851-854.

[3] H. Schulzrinne et al., *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, July 2003.

[4] M. Handley, V. Jacobson and C. Perkins, *SDP: Session Description Protocol*, RFC 4566, July 2006.

[5] J. Rosenberg, R. Mahy, P. Matthews and D. Wing, *Session Traversal Utilities for NAT (STUN)*, RFC 5389, October 2008.

[6] J. Rosenberg, R. Mahy and P. Matthews, *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*, RFC-to-be 5766, February 2010.

[7] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, RFC-to-be 5245, February 2010.

[8] J. Rosenberg and H. Schulzrinne, *An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing*, RFC 3581, August 2003.

[9] P. Srisuresh and K. Egevang, *Traditional IP Network Address Translator (Traditional NAT)*, RFC 3022, January 2001.

[10] T. Miller, *Passive OS Fingerprinting: Details and Techniques*, `http://www.ouah.org/incosfingerp.htm`.

[11] S. Donovan, *The SIP INFO Method*, RFC 2976, October 2000.

[12] V. Pascual et al., *A SIP Flight Data Recorder Extension*, work in progress, July 2009.

[13] Jess rule engine's web site: `http://www.jessrules.com/`

[14] C. L. Forgy, *Rete: a fast algorithm for the many pattern/many object pattern match problem.* In Expert Systems: A Software Methodology For Modern Applications, P. G. Raeth, Ed. Ieee Computer Society Reprint Collection. IEEE Computer Society Press, Los Alamitos, CA, 324-341

[15] Jain project's web site:
https://jain-sip.dev.java.net/
[16] UDP-Invoker project's web site:
http://code.google.com/p/udp-invoker/
[17] XML-RPC project's web site:
http://www.xmlrpc.com/
[18] Jpcap's web site:
http://netresearch.ics.uci.edu/kfujii/jpcap/doc/
[19] X-Lite's web site:
http://www.counterpath.com/x-lite.html
[20] SJPhone's web site: http://www.sjphone.org/
[21] Ekiga's web site: http://ekiga.org/
[22] PJSIP's web site: http://www.pjsip.org/
[23] OpenSIPS' web site: http://www.opensips.org/
[24] Sippy RTPproxy's web site:
http://www.rtpproxy.org/