Media Connectivity in SIP Infrastructures: Provider Awareness, Approaches, Consequences, and Applicability

Stefan Gasterstädt
adesso AG
Berlin, Germany
stefan.gasterstaedt@adesso.de

Markus Gusowski
Center for Information Services
and High Performance Computing
Technische Universität Dresden
Dresden, Germany
markus.gusowski@tu-dresden.de

***Abstract*—In SIP-based Voice over IP infrastructures, media data is usually exchanged directly between the endpoints using RTP without provider interaction. In contrast to the Public Switched Telephone Network where the delivery of all messages is the provider's responsibility, a SIP provider is not aware of media connectivity, i. e., whether a call was successful or not. This may lead to incorrect behavior when a Voice over IP provider offers services beyond signaling (for example, payment, prevention of Spam over Internet Telephony). Most existing mechanisms relating to media connectivity only aim at increasing the chance for connectivity or are endpoint centric and cannot achieve media connectivity awareness for the provider. We present and compare several approaches solving this problem that use both implicit and explict connectivity detection and notification mechanisms. Our favoured approach uses a set of behavioral rules for the user agents and implicit connectivity notification to achieve connectivity awareness. We also suggest SCTP for media transport and as an efficient connectivity detection mechanism. Besides conforming to the existing SIP standards and minimizing protocol changes, our solution is able to tolerate "lying" user agents. Measurements with our prototype SIP proxy implementation show that the impact on provider side call processing performance is negligible.**

***Keywords*-Voice over IP (VoIP), Session Initiation Protocol (SIP), Media Connectivity, Connectivity Awareness, SCTP.**

I. Introduction

The Session Initiation Protocol (SIP) [38] has become a majorly used protocol in Voice over IP (VoIP) communication. Often, SIP is used synonymously for VoIP infrastructures but it is actually one component of many. In particular, the request/response messages of SIP provide signaling (set up, modification and tear down of multimedia sessions), whereas the media data is nearly always transported directly between the user agents (UAs) using a separate media transport protocol. Only if two non-interoperable networks need to be connected, some specific Application Layer Gateway (ALG) will be involved. In terms of SIP, a gateway is just a special type of a user agent terminating the signaling path and, in this case, terminating the media path as well.

In most cases, the Real-time Transport Protocol (RTP) [40] is used for media transport between the end-points. Usually, SIP and RTP use the User Datagram Protocol (UDP) [28] as the underlying transport protocol, while SIP sometimes utilizes the Transmission Control Protocol (TCP) [30] or the Transport Layer Security Protocol (TLS) [10] to secure the signaling. The RTP transport addresses and capabilities are specified and exchanged using the Session Description Protocol (SDP) [14] and its offer-/answer mechanism. SDP itself is carried in a SIP message body. All of these protocols belong to the application layer; their classification and the underlying, majorly used protocols are shown in the internet multimedia protocol stack, Figure 1.

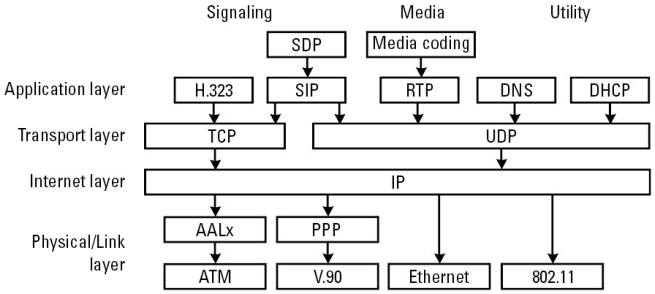

Figure 1: Internet Multimedia Protocol Stack [19, Fig. 1.1]

The SIP messages exchanged to set up and tear down a normal call (see Figure 3) and an example of a SIP invitation (see Figure 2) illustrate the separation and the interaction of these protocols. In this example figure, one can see the caller's invitation (*INVITE*) and the respective ringing/acceptance responses (*180 RINGING*, *200 OK*) send by the callee. Due to the fact that these messages contain all necessary information (e. g., current Internet Protocol (IP) adresses, port numbers, negotiated codecs and further media parameters), the subsequent acknowledgement (*ACK*), the media session itself, and the tear-down of the session (*BYE*, *OK*) can be send directly between both UAs.

SIP utilizes the Uniform Resource Identifier (URI) schema [5] to address users, single devices or end points and resolves these URIs to IP addresses [29] by using SIP proxy

```
INVITE sip:19@10.3.8.20:5060 SIP/2.0                                    % SIP request is an invitation
Via: SIP/2.0/UDP 10.3.8.18:5060;branch=z9hG4bK_000FC9022702_T664769F9    % Route of the message
Session-Expires: 1800                                                   %
From: "SIP Telefon 18" <sip:18@10.3.8.20:5060>;tag=000FC9022702_T634233581  % Caller information
To: <sip:19@10.3.8.20:5060>                                             % Callee information
Call-ID: CALL_ID11_000FC9022702_T907830378@10.3.8.18                    %
CSeq: 589933214 INVITE                                                  %
Contact: <sip:18@10.3.8.18:5060>                                        % information to potentially send
Max-Forwards: 70                                                        %           SIP messages directly
Allow: ACK,BYE,CANCEL,INVITE,NOTIFY,REFER,DO,UPDATE,OPTIONS,SUBSCRIBE,PRACK,INFO  %
Supported: 100rel,timer,replaces                                        %
User-Agent: ALL7950 02.09.31                                            %
Content-Type: application/sdp                                           % SIP body will contain a session
Content-Length: 231                                                     %                     description
                                                                        %
v=0                                                                     % The body of this message
o=18 212024437 212024437 IN IP4 10.3.8.18                               % contains the description of
s=ALL7950 02.09.31                                                      % the session offered by the
c=IN IP4 10.3.8.18                                                      % caller.
t=0 0                                                                   %
m=audio 41000 RTP/AVP 0 18 4                                            % It contains information about
a=rtpmap:0 PCMU/8000/1                                                  % the media type, codec, ip
a=rtpmap:18 G729/8000/1                                                 % address and port number and
a=fmtp:18 annexb=no                                                     % further.
a=rtpmap:4 G723/8000/1                                                  %
a=sendrecv                                                              %
```

Figure 2: Example of a SIP Invitation

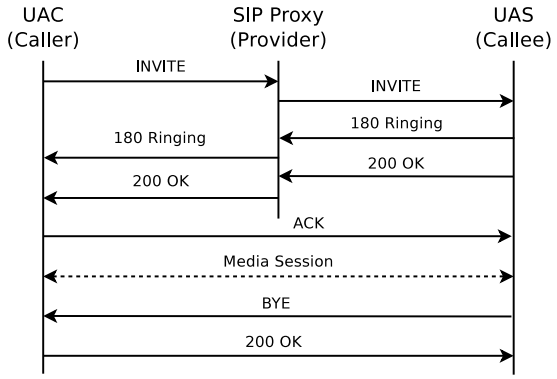

Figure 3: SIP Dialog of a Call

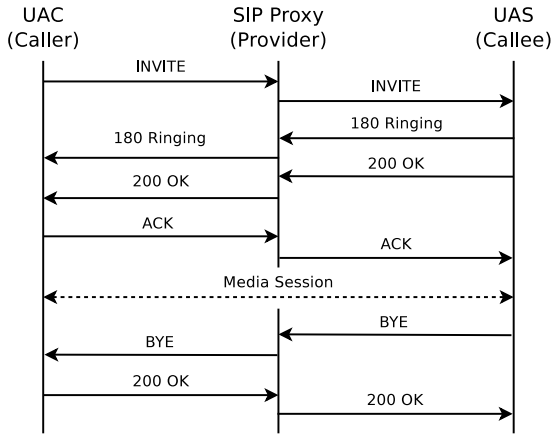

Figure 4: SIP Dialog of a Call with In-Route Proxy

servers and Domain Name Service (DNS) lookups [23], [24]. Users can call others without knowing their current IP address, because session invitations are routed to the SIP proxy that is responsible for the callee's URI domain; and as a next step, this proxy uses its location service to locate the callee and forwards the *INVITE* request to the addressed user. The location bindings can be updated by each respective user sending a *REGISTER* request to its SIP provider's registrar. Depending on its configuration, a SIP proxy may or may not request to stay in the route of any further SIP signaling (see Figures 3, 4). Independently, in most cases the media transmission is done directly between the UAs via RTP.

It is a known problem that the basic SIP infrastructure does not conform to the Network Address Translator (NAT) friendly application design guidelines described in RFC 3235 [41]. As a consequence, NATs and firewalls cause serious problems for SIP message delivery and media connectivity in conjunction with the separation of signaling and media delivery, dynamic port allocation, or RTP's "$x + 1$" port schema. In contrast to the UA-to-UA media connection, there are solutions for SIP messages; for example, by simply traversing NAT using symmetric response routing [37]. Examples of NAT and firewall traversal for SIP are given in [34].

The explicit separation between the session signaling and media delivery comes along with a significant implication: VoIP providers offering SIP services are unaware of whether or not the media stream is actually received by the endpoint(s), i. e., whether there is *connectivity* or not. SIP does not check for connectivity, and the condition is not signaled in any way. Therefore, a SIP provider cannot know if two users will actually be able to communicate,

even if a SIP session was successfully established. There are several reasons why media streams negotiated between the UAs may be blocked in one or both directions, mainly because of NATs and/or firewalls [16], [44], but other network problems like the lack of a network route, node crash, configuration problems, or codec mismatch could be responsible as well [2]. This is in contrast to the traditional Public Switched Telephone Network (PSTN), where there is always connectivity once signaling completes successfully. Admittedly, there are some rare cases where people cannot talk to each other allthough there has been a successful ringing and call acceptance before. However, the PSTN phone provider will be aware of this failure.

There are, however, important scenarios where it is desirable for the provider to know the media connectivity status between the endpoints.

Payment: In some cases, the callee or the caller request some fee in order to accept or initiate a call. Examples include duration-based fees (similar to the PSTN); (fixed) fees relating to the (voice based) service a callee is offering, such as a support hotline; fees for calls a callee subscribed for, such as severe thunderstorm warning; or, in the case of Spam over Internet Telephony (SPIT) prevention, where a caller may be confronted with a small fee if its sincerity is in doubt [18], [20].

For whatever reason a session involves payment by at least one party, it is desirable to delay finalizing the payment transaction until connectivity is assured.

Reputation: Some approaches to detect and prevent SPIT use a reputation score in order to help determine the caller's nature [4], [20], [32]. Each user's reputation is related to its behavior and is calculated from several metrics that are collected by the providers. For examples, a short call duration may indicate an unsolicited call that prompted that callee to hang up immediately. Unfortunately, it may also indicate that at least one participant could not hear the other due to a lack of (bidirectional) media connectivity. In this case, the caller's reputation would falsely be reduced.

Forensics: In the area of law enforcement, reliable evidence is crucial. Regarding the question of whether or not a call took place, SIP can only provide information about signaling – if the phone rang, if the phone was picked up, and if the phone was hung up. This may not be sufficient: The information may be required as to whether or not the two parties in a call were actually able to communicate.

Call Detail Record Analysis: Call Detail Records (CDRs) are collected and analyzed for several reasons. These records contain information about each call, for example, the caller's and callee's IDs, the invitation time, the duration, and how the call terminated. This data can be used to conduct statistical analysis, to profile users' behavior, to reduce traffic congestion or, in general, to detect any kind of anomaly. It is not sufficient if the CDRs are based on the SIP messages only, without knowing whether or not there was media connectivity. This might result in contra-productive network configuration, misinterpretation of someone's reputation or, even worse, will black-list a participant.

In this paper we present a solution for the *VoIP Media Connectivity Awareness Problem*, which fulfills the following requirements:

1) Focus: It is the *SIP Provider* who needs to obtain knowledge about the connectivity status.

2) Multiple (bi-directional) streams: It is important to consider *all media streams* negotiated between the calling parties. Any single uni-directional stream that is not established successfully might be the reason for one of the participant to end the call prematurely. Thus, the provider needs to determine at least the connectivity status for the stream aggregate. For example, if any single media stream in any direction lacks connectivity, the stream aggregate is considered to have no connectivity.

3) Genuineness: In order to prevent false conclusions (and subsequent actions), the connectivity status gathered by the provider should be *genuine*.

4) Compatibility: The number of changes introduced into the SIP message sequences should be as small as possible. Ideally, neither extra SIP messages nor additional SIP headers should be required.

Parts of this paper have earlier been published in the Proceedings of the Tenth International Conference on Networks (ICN 2011) [1]. The focus was only on a single solution, dealing with the SIP providers' media connectivity awareness. In this article, we have extended the work presented in [1] in several ways. In detail, this article considers an extended range of related work, introduces and compares alternative approaches to solving the connectivity awareness problem, contains additional and improved call scenarios depicting the SIP message flows, presents specific protocol extensions to SDP for using the Stream Control Transmission Protocol (SCTP) as media transport, shows implementation details of the SIP Proxy message routing, and elaborates on measurement results.

This paper is structured as follows: In Section II, we discuss several existing approaches that have some relation to the awareness of media connectivity. In Section III, our favoured approach is presented. The section includes detailed scenarios, a preliminary investigation of using the SCTP for connectivity detection, a description of the proposed protocol extensions to SIP and SDP, and outlines several other possible approaches. Finally, Section IV presents a prototype SIP Proxy implementation of our solution and contains measurements of the performance overhead our solution introduces.

II. Related Work

There are some approaches that relate to the awareness of media connectivity, but which are motivated by different goals.

A. Dealing with the NAT

One possibility to solve the connectivity problem is the use of an ALG in addition to the NAT. In reality, however, ALGs are deployed in the fewest scenarios, even though most users manage their own private home networks. Furthermore, an ALG might increase the chance to achieve media connectivity, but the SIP provider still does not know about it.

In contrast to the UA-to-UA media connection, there is a very high chance to deliver all SIP messages by, e. g., traversing NAT using symmetric response routing [37]. Traversing the NAT for the media streams can be done using Interactive Connectivity Establishment (ICE) [33]. ICE describes NAT traversal for multimedia signaling protocols like SIP, and it extends the SDP [14] to convey additional data. In order to operate, ICE utilizes the protocols Session Traversal Utilities for NAT (STUN) [35] and Traversal Using Relays around NAT (TURN) [22].

The goal of ICE is to *establish* connectivity, but not to require it or to inform a third party of the connectivity status. This process of connectivity establishment is in principle independent of session establishment – a SIP session is allowed to be established successfully, even if there is no media connectivity.

B. Connectivity Preconditions

UAs may use Connectivity Preconditions as defined in RFC 5898 [3] to *verify* whether there is connectivity or not. Based on the concept of a SDP precondition in SIP as specified by RFC 3312 [8] (generalized by RFC 4032 [7]), the connectivity precondition defined by RFC 5898 tries to ensure that session progress is delayed (including suppression of alerting the called party) until media stream connectivity has been verified.

This approach is motivated by the separation of signaling and media path and its implications. Similar to a part of the solution described in this paper (see Section III), it enables the UAs to delay the SIP session establishment until connectivity is ensured. RFC 5898 has been published in July 2010 and does contain similarities to this paper, which we worked on at the same time. In contrast to our approach, the provider cannot enforce the UAs to make use of this extension. In addition, it does not inform a third party (such as the provider) of the connectivity status – neither implicitly nor explicitly. In particular, the provider is not aware of the media connectivity status.

Furthermore, RFC 5898 does not guarantee that session establishment comes along with media connectivity. In RFC 3312 (which is referenced by RFC 5898), alerting the user until all the mandatory preconditions are met has a "SHOULD NOT" semantics. According to the definition in RFC 2119 [6], this means that suspending session establishment is *not* guaranteed since the UA may have "*[. . .] valid reasons in particular circumstances when the particular behavior is acceptable or even useful [. . .]*" [6, Sec. 4]. Even though the intentions of RFC 5898 and RFC 3312 are clear, the question remains if a provider interested in the connectivity status can rely on the information obtained from using Connectivity Preconditions.

C. Receiving RTCP information

RTP, the protocol used to transport the media data comes along with its own RTP Control Protocol (RTCP). This protocol is used between the RTP endpoints to send and receive statistical data about the quality of the received RTP streams. If the provider received these quality metrics, they could be used to *derive* a connectivity status for the corresponding RTP stream.

As a first possibility, a provider could "misuse" the solution suggested in [17] that tries to solve the "$x + 1$" RTCP port number problem with NATs. Due to the fact, that "it is even possible that the RTP and the RTCP ports may be mapped to different addresses" [17, p. 2] the RTCP streams could be redirected to the SIP provider who can analyze the incoming information and then forward the RTCP packets to the other endpoint.

As a second option, a provider can use the SIP Event Package for Voice Quality Reporting [27] to receive reports about the call's quality metrics. The metrics are derived from the RTCP Extended Reports [12] and are reported to an interested third party using the SIP-specific event notification [31]. Using this mechanism, a provider can subscribe to the event with a UA in order to receive metric information periodically. This is done using the SIP request messages *SUBSCRIBE* and *NOTIFY* respectively.

In contrast to ICE and Connectivity Preconditions, in both mechanisms, the media connection's information (either redirected RTCP packets or explicit signaling) is separate from the session establishment. Thus, a SIP session is established regardless of connectivity status, and the provider can then derive the connectivity status directly from the RTCP packets or event notifications. In addition, obtaining the connectivity status is not only separate from session establishment, but can only be done *after* the SIP session is established and *after* the media streams are set up. In fact, media must be sent first since RTCP packets (being a prerequisite) are not exchanged between the endpoints any earlier.

The dependence on RTCP introduces further issues. First of all, RTCP packets/RTCP Extended Reports may not arrive because there is no connectivity for the RTCP stream. However, this does not imply a lack of connectivity for the media stream since RTCP uses a different UDP port number (i. e., a different *transport address*) than the media stream. Packets may also not arrive because the other endpoint does not send them for some reason, even though there might be connectivity.

In addition, the quality metrics sent to the provider may be wrong because an endpoint deliberately falsified the information. Thus, when a provider uses the connectivity status to draw further conclusions (reputation, payment rollback, etc.), it needs to consider the trustworthiness of the information used to determine the connectivity status.

D. Disconnection Tolerance

Ott and Xiaojun [26] present mechanisms for detection of and recovery from temporary service failures for mobile SIP users.

For detection of connectivity loss, they suggest a media-based approach: Missing RTP packets, RTCP packets, or STUN packets along with some additional criteria are used as indicators that connectivity has been lost. If the connectivity loss persists longer ("call interruptions"), the UAs will automatically try to re-establish the session after locally terminating the session. For this purpose, the authors introduce the new SIP *Recovery* header field, which is set to `true` in the *INVITE* message used to re-establish the session.

By observing this header field, an in-route SIP proxy (and therefore the provider) has a way to know about connectivity loss in the previous session. However, the field contains no information about when the connectivity loss occurred. Finally, if the lack of connectivity persists even longer and automatic re-establishment fails ("call termination"), the system reverts to voice mail or instant messaging.

The focus of this paper is on obtaining the connectivity status during an ongoing session *after* the session has been established. Implicitly, it assumes that there was connectivity at the beginning of the session.

E. Conclusion

In all solutions presented except for the SIP Event Package for Voice Quality Reporting, the focus is always on the endpoints. Whether the main goal is to establish connectivity, ensure connectivity, detect/monitor connectivity status, or recover from connectivity loss, the assumption is always that the *endpoints* are the entities which are interested in the goal.

Hence, the provider is not aware of the media connectivity; and even when the connectivity information can be obtained, its validity and genuineness may be questionable.

III. Implicit Connectivity Detection and Notification

One major difference between the approaches presented above is *when* information pertaining to connectivity status is obtained. Three distinct cases can be identified: before session establishment (ICE, Connectivity Preconditions), after session establishment (Disconnection Tolerance [detection only], SIP Event Package for Voice Quality Reporting, RTCP attribute in SDP), and at the end of the conversation (Disconnection Tolerance [signaled through *Recovery* header field]). In the second case, the information can also be obtained continually during the ongoing session.

Another difference is found in the direction of a media stream for which connectivity status is determined and whether media streams are considered separately or jointly on a "session level." Most mechanisms distinguish between individual streams and, as streams are usually considered uni-directional, also between receiving and sending direction. Connectivity Preconditions distinguish both direction and individual streams, but the consequence (suspension of session establishment) is affected by the aggregate of the streams for which the precondition was requested. The Disconnection Tolerance solution disregards direction as symmetric connectivity is assumed; it also disregards individual streams because the existence of only one audio stream is assumed (point-to-point audio conversation).

In our approach, connectivity notification (Section III-A) and detection (Section III-B) is done before session establishment. Further, our solution regards both, different streams and directions. In addition, it ensures the genuineness of the connectivity status by considering missbehaving UAs (Section III-C).

A. Implicit Connectivity Notification

SIP itself already offers several possibilities to modify the message routing. For example, a SIP proxy can request to stay in the route of any SIP messages beyond those belonging the first SIP request. In order to achieve this, any UA sending a new SIP request needs to insert corresponding routing information. Thus, in contrast to a "normal" SIP session establishment, a proxy can become a mandatory node of the last SIP 3-way-handshake message, i. e., the *ACK* request (compare Section I, Figures 3, 4). Furthermore, the user agent server (UAS) does not necessarily need to send a *180 Ringing* response and notify the called person. Instead, it can respond with a *183 Session Progress* message to indicate further action prior to continued call processing.

This response message and the modified message routing can be combined with a modified UA behavior. By using the *183* response's payload, the callee can answer the caller's SDP offer. Thus, both parties know the parameters of all media sessions that will usually be established *after* the SIP session invitation has been accepted. Since the *183* now contains important information about the media session (SDP answer), it is crucial to ensure message reception at the caller's side. Fortunately with the Reliability of Provisional Responses [36], a mechanism exists that enables user agents to detect lost provisional responses and ensures their delivery by using special acknowledgements (*PRACK*) and retransmissions. We will not consider *PRACK* messages during the further investigation of our solution as this is beyond the scope of this paper. However, even without using

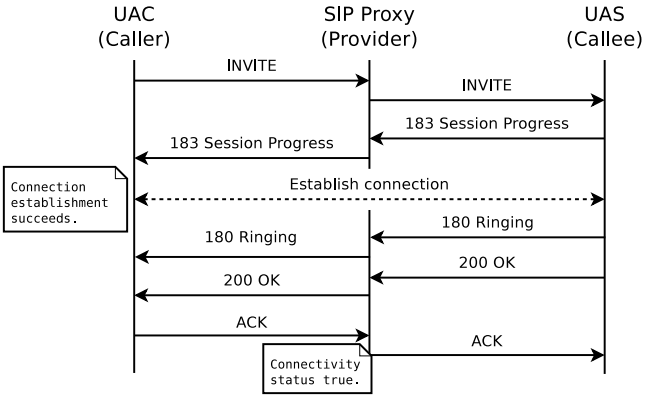

Figure 5: Accepted Call with Prechecked Media Connectivity

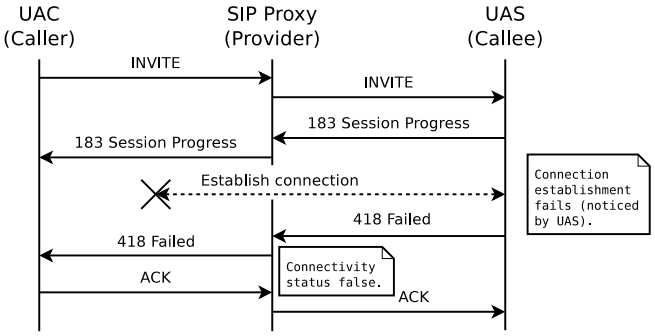

Figure 6: Call Abortion in the Case of no Media Connectivity, detected by the UAS

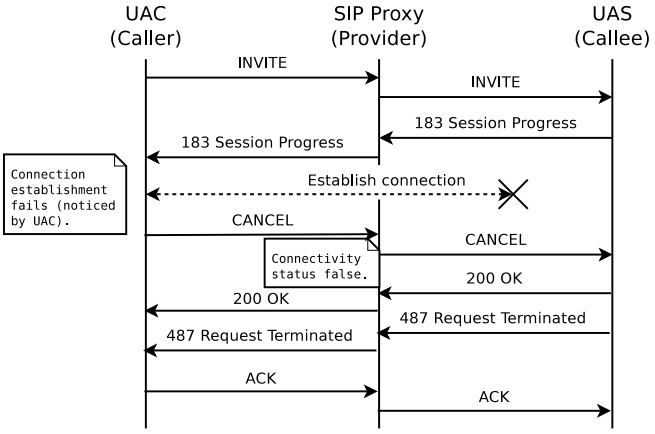

Figure 7: Call Abortion in the Case of no Media Connectivity, detected by the UAC

this mechanism, our solution will still work correctly with respect to determining the media connectivity status.

In our solution, the media sessions are established *beforehand*, and both parties **must hold back** the *180 Ringing*, *200 OK*, and the *ACK* messages until this has happened. Establishing the media sessions must involve some kind of connectivity detection mechanism, which will be considered in the next section. For simplicity, we refer to the establishment of all media sessions in their entirety as "connection establishment," where the connection establishment is considered successful when connectivity detection was positive for all media streams. With those rules, the *200 OK* and *ACK* messages act as connectivity confirmations by the UAS and user agent client (UAC), respectively. Furthermore, each UA **must ignore** any incoming media packets and **must not send** any media packets as long as the other endpoint did not confirm connectivity. This restriction enforces the connectivity status. It ensures that the reported connectivity status always matches the actual connectivity status as experienced by the user (genuineness requirement).

In result, the provider can conclude the media connectivity status by simply analyzing the messages it is routing (focus requirement). Therefore, we call the approach *implicit*. The provider will **conclude that there is connectivity if and only if** the UAS has sent a *200 OK* and then the UAC has sent an *ACK*.

In case the media connection could be established successfully, there will be a notification (*180 Ringing*), acceptance (*200 OK*) and acknowledgement (*ACK*) (see Figure 5). In result, the provider concludes that there is media connectivity.

If the UAS notices that establishing the media connection failed, it will reject the call by sending a *418* error response (see Figure 6). The latter is a new response code further explained in Section III-D. Since it is a final error response in the *4xx* category, even a UA who did not know about the new response code would consider SIP session establishment as failed and act appropriately. If the failure is detected by the UAC (see Figure 7), it will cancel the call using the *CANCEL* request causing the UAS to respond to the invitation with *487 Request Terminated*. In both cases, the provider concludes that there is no media connectivity.

In Figures 8 and 9, the media connection has been established successfully but the callee is unavailable. Thus, either the caller will *CANCEL* the call when a timeout appeared or the callee will response with a corresponding *408 Request Timeout* message. Again, the provider concludes lack of media connectivity.

B. Connectivity Detection

Due to the fact that the provider is simply analyzing the messages it is routing, it is up to the clients to verify the connectivity status. In detail, they need to check every single media stream for connectivity (multiple streams requirement), for example, by using STUN messages similar to the connectivity checks in ICE. This can be complex and time consuming.

In order to limit this overhead, we propose the use of the Stream Control Transmission Protocol (SCTP) [45] as the

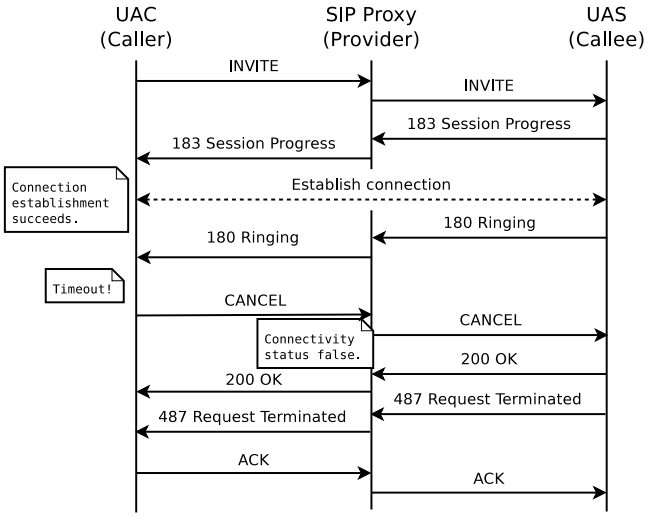

Figure 8: Timed-out Call with Prechecked Media Connectivity, detected by the UAC

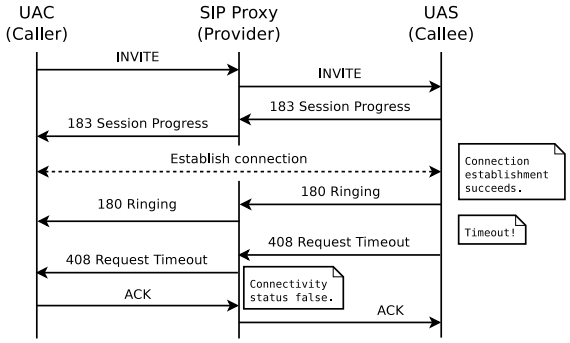

Figure 9: Timed-out Call with Prechecked Media Connectivity, detected by the UAS

media's underlying transport protocol. First of all, SCTP is connection oriented – SCTP's 4-way handshake at the beginning already ensures transport layer connectivity. In result, neither a media packet nor a notice of receipt need to be sent in order to check for connectivity.

Secondly, SCTP itself offers multiplexing; so there is no need for more than one connection, as every single RTP/RTCP stream can be sent using the same unique connection. In result, the time required to check each media stream (and media control stream) is reduced to a single check only. Last but not least, in contrast to TCP, SCTP offers unordered transport, meaning a lost packet does not delay delivery of succeeding packets. In addition, the partial reliable mode (SCTP Partial Reliability Extension [46]) can be used to improve the media quality in case a lost packet can be retransmitted immediately.

To confirm our proposal, we measured the SCTP performance in comparison to UDP. The environment consists of

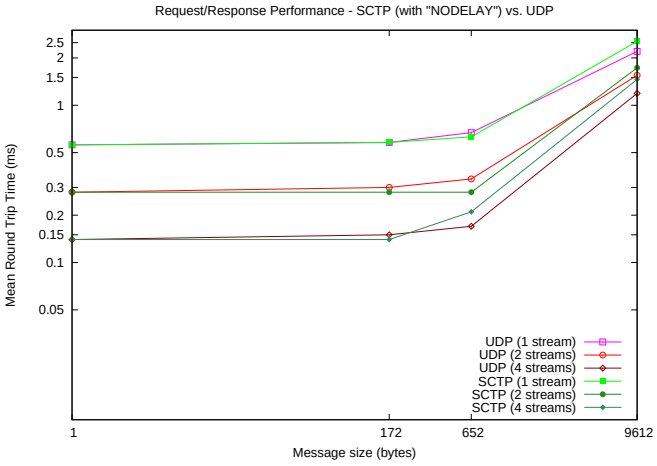

Figure 10: SCTP vs. UDP

two machines with identical hardware and software running Debian GNU/Linux 5.0.3 (lenny) with kernel version 2.6.26 (i686). Both machines are equipped with an Intel Core 2 Duo E7500 dual core CPU running at 2.93 GHz and an Intel 82567LM-3 network adapter and connected via a FastEthernet switch (100 Mbps Full Duplex). The environment also determines the choice of UDP and SCTP implementations used – those of the Linux kernel. The benchmark itself is a ping-pong application that can send multiple messages at once, approximating multiple concurrent media streams. Figure 10 shows the mean round-trip time (RTT) in relation to the size of the messages. The sizes of 172 Bytes and 652 Bytes correlate to the RTP packet sizes produced by the G.711 codec using packet transmission cycles of 20 ms and 80 ms, respectively. One can see that the values of SCTP are very close to those of UDP, and hence, we expect no performance loss due to the use of SCTP.

C. Missbehaving user agents

In some cases, either the UAS or the UAC might try to falsify the information it tells about the connectivity status. Our solution requires sending a *200 OK* (UAS) or *ACK* (UAC) to convey "connectivity" or suppress those messages to convey that there is no connectivity.

For example, a caller might falsely announce "no connectivity" to the provider in order to send SPIT calls without consequences. In our solution, the UAC would have to suppress the *ACK* message. Fortunately, this would cause the callee to ignore any incoming media packets and to terminate the call by sending a *BYE* (see Figure 11). In case the UAS sends media packets instead of responding with a *200 OK*, the UAC is able to *CANCEL* the SIP session easily (see Figure 12).

In the payment example, the callee might say "connectivity" in order to receive his fee anyway. In this case, the caller receives a *200 OK* even though there is no

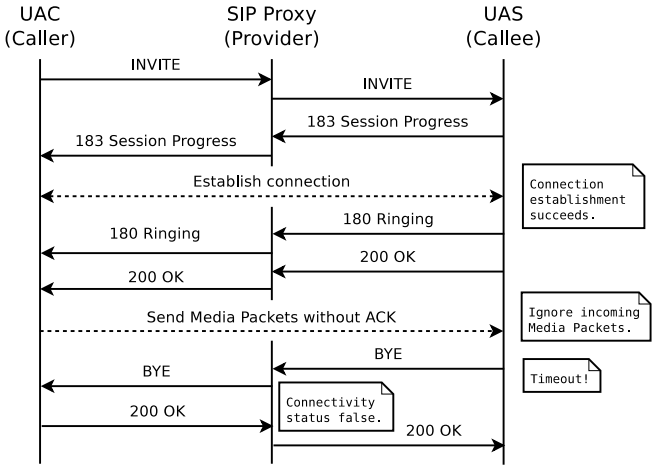

Figure 11: Media Delivery Without Prior *ACK*

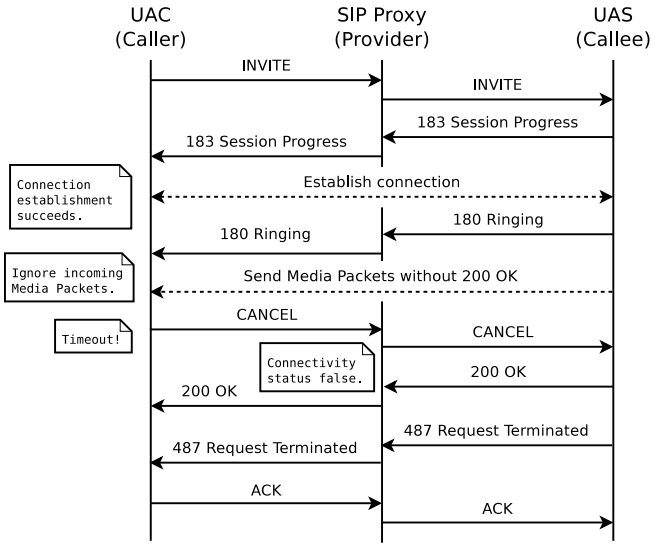

Figure 12: Media Delivery Without Prior *200 OK*

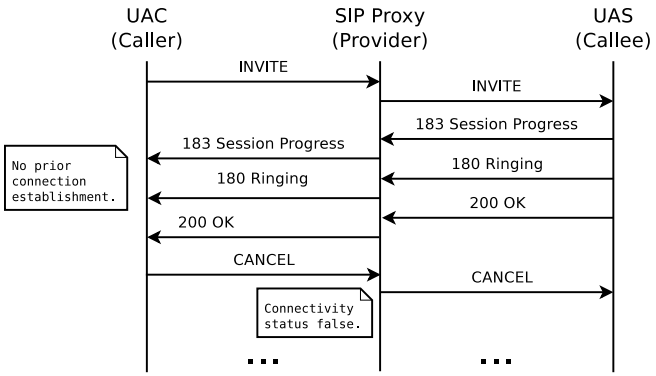

Figure 13: Immediate Session Acceptance Without Prior Connection Establishment

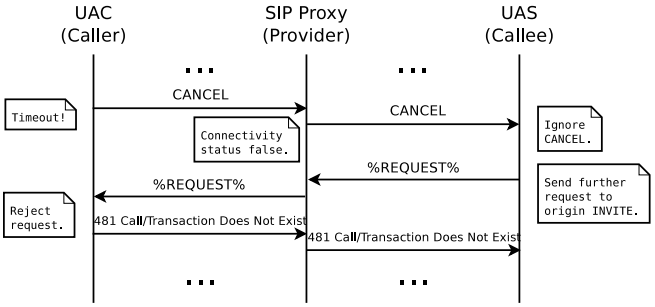

Figure 14: Ignored CANCEL, but further Requests

media connectivity. According to the SIP specification, this final response needs to be acknowledged. Unfortunately, the provider would conclude connectivity, and thus, the caller needs to terminate the call without sending an *ACK* first.This can be achieved by sending a *CANCEL* request, causing a situation that – from the viewpoint of all other SIP entities – looks like the race condition described in RFC 5407 [15, Example 3.1.2.]. In result, the provider can conclude lack of connectivity and the callee is informed that the caller will not participate in the call any longer. The rest of this SIP message sequence is not shown since it depends on the implementation details of the UAs. In case the UAS has not implemented the correction to SIP suggested in RFC 6026 [42], the UAC may receive a *481 Call/Transaction Does Not Exist* instead of a *200 OK* to the *CANCEL*. Independently, the callee will not acknowledge any retransmitted *200 OK* to the *INVITE* pretending it never received these responses. In any case (even if timeouts are provoked), the provider's conclusion about missing connectivity will be correct.

In summary, for whatever reason a UA might missbehave – our solution enables the opponent party to react appropriately, enabling the provider to know the actual connectivity status. The general rule is: **Connection establishment before sending call acknowledgment, receiving call aknowledgment before connection usage** – otherwise, the call has to be rejected by adequate SIP messages.

The case that both, caller and callee, are lying cooperatively cannot be detected with our approach, but this is only a problem in the forensics scenario. It is doubtful, however, that the calling partners would use a provider at all to communicate in a criminal scenario.

Furthermore, a missbehaving callee might ignore the cancellation of a SIP session invitation and still send further requests and/or responses. These messages, however, can be rejected or ignored by the receiving caller (see Figures 14, 15). Similarly (not shown seperately), a callee can easily deal with further incoming SIP messages relating to a session that has previously been responded to with a final *4xx* client error response.

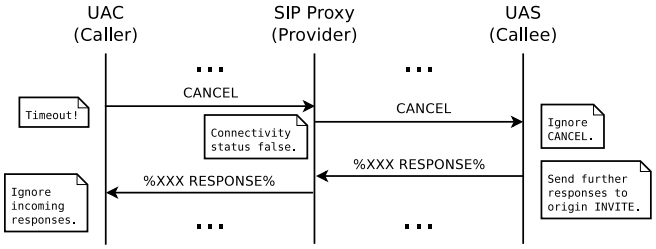

Figure 15: Ignored CANCEL, but further Responses

D. Protocol Extensions

There has been some work in the past for SCTP and SIP. Unfortunately the Internet-Draft by Fairlie-Cuninghame [11] is incomplete, inconsistent, and seems to have been abandoned. Another Internet-Draft by Loreto and Camarillo [21] tackles the same topic, but is very limited in its scope as it treats an SCTP association like a TCP connection in the sense that it completely ignores SCTP's multi-streaming feature. It is basically a one-to-one redefinition of RFC 4145 [47] for SCTP, specifying two additional protocol identifiers only. Similar to [11], it says nothing about how to use RTP over SCTP. This is different to our approach, since we do not use SCTP to deliver the SIP messages but to transport the media data. In order to simplify the media connectivity detection, all RTP streams are multiplexed by using a single SCTP connection.

In line with our compatibility requirement (see Section I), our solution only needs to slightly extend the abilities of SDP in order to specify the SCTP parameters. The use of the SCTP connection and the modified UA behavior can be indicated by identifying our extension – the SCTP Tunneling Extension for SIP – within a *Require* header by using a new option tag ("*sctp-tunnel´´*). If an incoming *INVITE* does not indicate usage of this extension the provider must reject this request by sending a *421 Extension Required* response. As described above, the extension just specifies the way the UAs must behave and the provider can draw conclusions; it does not specify any new SIP messages or headers – all of them have existed before.

We propose the following extensions to SDP in order to accomodate the use of SCTP as the transport protocol for media. First, we define the new "`proto`"-field value "`SCTP/RTP/AVP`", which must be used in every session description "`m=`" line when the SCTP Tunneling Extension is used. "`SCTP/RTP/AVP`" denotes Real-time Transport Protocol (RTP) [40] used under the RTP Profile for Audio and Video Conferences with Minimal Control [39] running over SCTP [45]. We use the "`c=`" line to specify the IP address to which the SCTP tunnel should be established, which only has to be done once at the session level. Further, we define a new mandatory session-level attribute, "`sctpPort`", which holds the port number to which the SCTP tunnel should be established. The syntax is defined in ABNF [9] as follows (cf. [11]):

```
sctpport-attribute = "sctpPort:" port
port               = 1*DIGIT
```

To accommodate multi-homed SCTP endpoints, we define a new optional session-level attribute, "`sctpAddr`", that contains a list of IP addresses. It can be used to specify IP addresses that for establishing the SCTP tunnel in addition to the one specified in the "`c=`" line. The syntax in ABNF [9] can be defined as follows:

```
sctpaddr-attribute = "sctpAddr:"
    sctpaddr-elem *("," sctpaddr-elem)
sctpaddr-elem      =  nettype SP
    addrtype SP connection-address
```

In order to be able to specify SCTP stream numbers on which the endpoints expect to receive media packets for the various media streams, we redefine the notion of a "port" in SDP to mean "SCTP stream number." The same rules how to assign port numbers for RTCP can be used for SCTP stream numbers. For example, even stream numbers are used for RTP and odd stream numbers for RTCP (cf. RFC 4566 [14, Sec. 5.14]). In general, whenever a SDP specification refers to a port number, this can simply be read as "SCTP stream number."

To manage SCTP association establishment, the mechanims for TCP connection management defined in RFC 4145 [47] can be used analogously for SCTP. However, to allow for simultaneous association establishment, we extend the "`setup`" attribute with the value "`simul`". Simultaneous association establishment – also called "initialization collision resolution" – can be useful to enable two endpoints that are behind different NATs to successfully establish an association.

The endpoints can now assume the following roles (defined in ABNF):

```
role  =  "active" / "passive" / "actpass"
       / "holdconn" / "simul"
```

where "`simul`" has the following semantics:

```
"simul": The endpoint is willing to
    accept an incoming connection, to
    initiate an outgoing connection,
    or to use simultaneous connection
    establishment (both endpoints will
    initiate the connection at the same
    time).
```

In the offer/answer model, "`simul`" gives the answering endpoint the option to choose among all options, so the answering endpoint can become active, passive, use simultaneous connection establishment, or the connection is not established for the time being.

Beyond SDP, the only syntactical extensions to SIP proposed in this paper (besides the behavioral definitions) are the additional option tag *sctp-tunnel* and the new response code number *418*.

Option tags define identifiers for SIP extensions and their use in the *Require* and *Supported* header fields. They are registered by the IANA under the "Session Initiation Protocol (SIP) Parameters" registry under the "Option Tags" sub-registry. The *sctp-tunnel* option tag can be defined as follows:

Name:

sctp-tunnel

Description:

This option tag is for tunneling all media streams between two endpoints of a SIP session through a single SCTP association as specified in the SCTP Tunneling Extension for SIP. When present in the *Supported* header field, it indicates that the UA is able to use the extension. When present in the *Require* header field, it indicates that UAC and UAS MUST use the SCTP Tunneling Extension and follow the rules specified therein.

Response codes are registered by the IANA under the "Session Initiation Protocol (SIP) Parameters" registry under the "Methods and Response Codes" sub-registry. The response code is defined as follows:

Response Code Number:

418

Default Reason Phrase:

SCTP Association Initialization Failed

The code can be used by the user agents to cause the *INVITE* request to fail when the SCTP tunnel cannot be established. Depending on which of the UAs "notices" that establishment failed (for example by using timeouts), the response code can be used directly as a failure response (by the UAS) or as a cause parameter in the *Reason* header field of a *CANCEL* request (by the UAC).

E. Further Approaches

Since it fulfills all requirements introduced at the beginning of this paper, we prefer the approach described above. Nevertheless, one might think of further approaches to solve the VoIP Media Connectivity Awareness Problem.

1) Media Gateway: Obviously, a provider could act as a media gateway, meaning all media data will be routed through a dedicated network component under its control. Since the gateway resides on the public Internet and media packets travel only between gateway and endpoint (not between endpoints directly), the typical connectivity restrictions caused by NATs and firewalls do not apply. Therefore – being an active part of all RTP streams – the provider can easily determine the media connectivity status. Setting up such a VoIP/SIP gateway can be done easily by using Asterisk [43], for example.

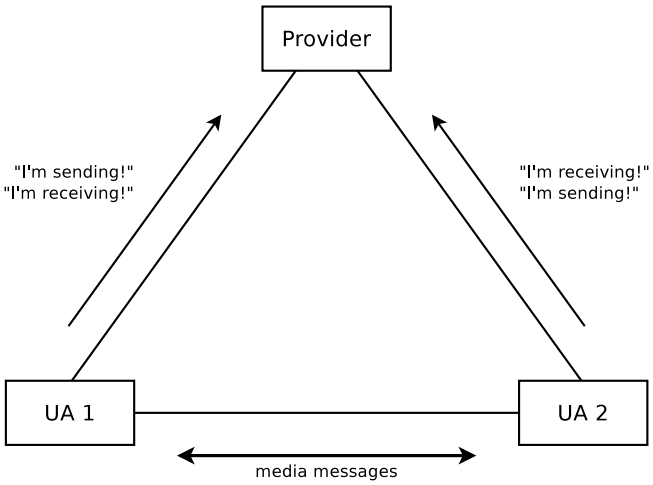

Figure 16: General View of Explicit Notification

Besides our "focus" requirement (see Section I), this solution fulfills the requirements multiple (bi-directional) streams, genuineness, and compatibility as well. On the other hand, this approach does not conform to the peer-to-peer architecture of VoIP infrastructures based on SIP. In addition, it requires a considerable amount of computational and network resources. In reality, VoIP services are often offered for free, and thus, providing these resources would not be economical.

2) Explicit Connectivity Notification: Besides an implicit notification, the UAs also can inform the provider about the connectivity status explicitly. In detail, the UAs send information about the connectivity status of every media stream they send and/or receive media packets on. A general view of this behaviour is depicted in Figure 16.

Such notifications can be implemented using *SIP-Specific Event Notification* [31], whereas, an *event package* needs to be defined that specifies the exact behavior of UAs subscribing to events and reporting events as well as syntax and semantics of the *NOTIFY* and *SUBSCRIBE* messages.

First, the provider subscribes to the *connectivity event* by sending a *SUBSCRIBE* message to the UAs. "The provider" could be the same SIP element as the SIP proxy, but it could also be a separate server belonging to the provider that communicates with the proxy. In the following section, the SIP element receiving and processing the connectivity notifications is called *connectivity server (CS)*.

For each stream where the UA is in the sender role (bi-directional and send-only streams), the UA notifies the CS as soon as it starts sending on that stream. Since this typically happens for most or all streams at the same time, the notifications for several streams can be sent in *one NOTIFY* message.

For each stream where the UA is in the receiver role (bi-directional and receive-only streams), the UA notifies the CS as soon as it receives the first packet on that stream. This

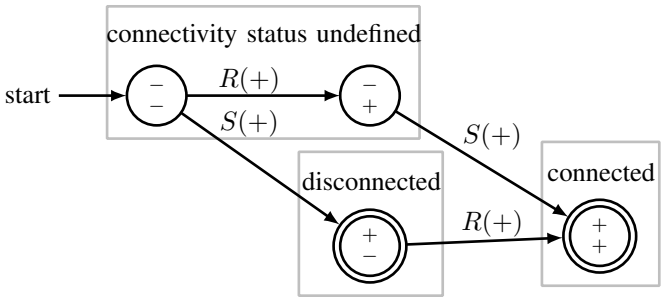

Figure 17: State Machine of Connectivity Status Calculation ($S(+)$ represents sender's sending notification, $R(+)$ represents receiver's receiving notification)

will potentially also happen roughly at the same time for most streams, so the UA can buffer the event for a short period of time in order to, again, include the notifications for several streams in *one NOTIFY* message.

The CS keeps a state table that holds the sender and receiver state (whether or not the sender is sending and whether or not the receiver is receiving) for each active media stream in the session (inactive and rejected streams are not relevant). Since the CS needs to know when a session is established and terminated, it has to communicate with the SIP proxy responsible for the call. It also needs to have a current description of the session, i. e., which streams with what directions have been successfully negotiated and which streams are inactive. In the end, the CS matches up the two notifications for each stream of the media connection and determines its connectivity status (see Figure 17).

In order to be able to determine the connectivity status, the CS needs to make sure that it will receive connectivity notifications from both endpoints involved in a call. To achieve this, the SIP proxy routing the call must delay forwarding any *INVITE* requests until it has successfully subscribed to the connectivity event with *both* UAs. If subscribing fails, the *INVITE* must be rejected. Once a subscription has been made, there should be little difficulty receiving the notifications: NATs or firewalls are unlikely to block notifications since the process of subscribing represents a two-way handshake (*SUBSCRIBE*, *200 OK*) and therefore ensures signaling connectivity between CS and UAs.

Figure 18 shows a message flow between two UAs belonging to the same provider. In this example, the connectivity server and the SIP proxy are the same SIP element. Both UAs register with the proxy first, after which the CS subscribes to the connectivity notifications. Then, a session is established successfully, after which both UAs send their connectivity notifications.

In case a callee is registered at a different SIP provider, it is considerably more complex to subscribe to the connectivity notification. In addition, provider and endpoints have a significant amount of additional work to do compared to a "regular" SIP session, especially the provider. Whereas each

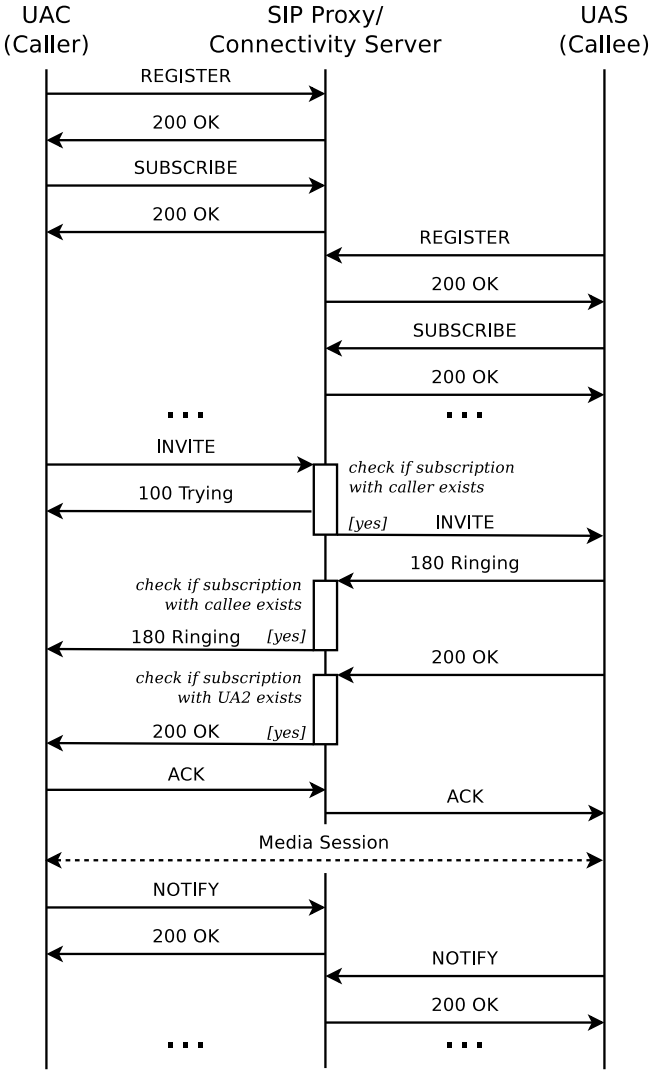

Figure 18: SIP Sequence of Explicit Notification

endpoint has to perform the subscription procedure, monitor incoming messages on all media streams, and compile and send the connectivity notification(s), the provider has to perform and manage subscriptions for a potentially great number of users, keep additional state for every SIP session (state machine), and do calculations for every media stream in every session. In addition, the provider has to keep an explicit history with the connectivity status for every completed SIP session in order to draw conclusions from the connectivity statuses later. This task is not necessary with our first approach (implicit connectivity notification), since there, every successful SIP session establishment implies that there was connectivity.

Furthermore, this approach does not fulfill the requirement of genuineness. The connectivity status is only correct under the assumption that both endpoints are generating the notifications truthfully. In contrast to the implicit approach, this

explicit approach does not include a mechanism to enforce the connectivity status. It would be difficult to introduce such mechanism for two reasons: First, there is no feedback to the UAs about the connectivity status that would enable them to act appropriately (stop sending media or ignoring incoming media). Second, the nature of the connectivity detection (watching for incoming media packets) requires the active use of the media channel *before* the connectivity status is known. So essentially, a reaction would always be too late.

In addition, the assumption is made that notifications are not blocked or altered in transit to the CS and that they are coming from the correct endpoint – additional measures would be required to verify the authenticity of endpoints and the integrity of messages.

3) Connectivity Verification with Secret Tokens: Instead of concluding the connectivity status out of UAs notifications, a provider can test each media connection indirectly by using secret tokens initially known only to the provider. Modifying the messages of the signaling channel, the provider can send these tokens to the UAs. As a second step, the provider expects the UAs to relay these tokens using the media connections. Thus, the corresponding calling partner receives the tokens in case the media connection is set up correctly. Finally, the received tokens will be transmitted to the provider in order to enable each corresponding connectivity verification.

Figure 19 shows a typical message flow for the whole process of connectivity verification of a single bi-directional media stream. The *INFO* method is used for the UA to proxy communication. Note that the *200 OK* to the *INVITE*, the corresponding *ACK*, and further additional provisional responses are not shown in the diagram. Similar to the implicit approach, a *183 Session Progress* message is required to obtain the media parameters (used to send the tokens) before connection establishment will be acknowledged via *200 OK* and *ACK*.

Unfortunately, receiving a wrong or, even worse, no token does not necessarily imply lack of connectivity since a participant might lie by falsifying or suppressing the token.

Nevertheless, this approach fulfills all requirements. However, the mechanism would become very complex, in case the participants are registered at different providers and both providers want to verify connectivity, since the tokens need to be mapped to their respective creators.

Furthermore, the provider needs to identify every media stream to be used in order to generate the corresponding amount of tokens, timeouts have to be defined, the message format needs to be specified, a new SIP header is required to carry the information, and the UAs' behavior needs to be ensured.

It is questionable if this solution fully fulfills our compatibility requirement. On the media transport level, the RTP media streams are "misused" by transporting non-media

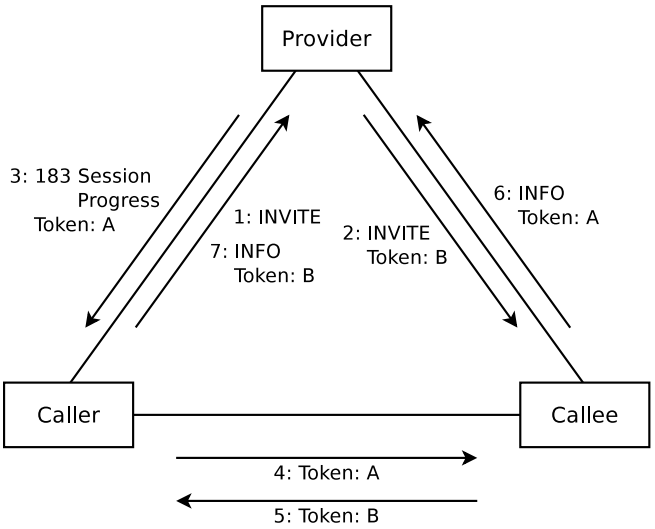

Figure 19: Connectivity Verification with Random Numbers

data. On the signaling level, the SIP *INFO* request is not used in the protocol intended way, since it is actually an end-to-end message and thus needs to be "illegally" intercepted by the proxy in order to process it (instead of forwarding it to the calling partner). If the SIP-specific event notification was used instead, similar problems to the explicit notification approach would arise.

F. Summary

Compared to all other approaches, the explicit connectivity notification approach has one major drawback: It does not fulfill the genuineness requirement because it cannot guarantee that the connectivity status seen by the provider is genuine – an endpoint can lie. All other approaches satisfy this requirement in the sense that one endpoint alone can never falsify the connectivity status seen by the provider. The key to this feature is the concept of *enforcing* the connectivity status: Each endpoint knows the connectivity status claimed by the other endpoint and only exchanges media packets if and only if this claims states that there is connectivity.

All other requirements, on the other hand, are satisfied by all the approaches presented.

However, the alternative approaches in general require much more resources and efforts to put these solutions into practice. Considering each approach's costs added by the additional behavior, the way SIP and other involved protocols are extended or altered, the effort required for implementing the mechanism, and each applicability, the implicit notification approach presents itself as the most promising solution. Thus, this approach is the focus of our research.

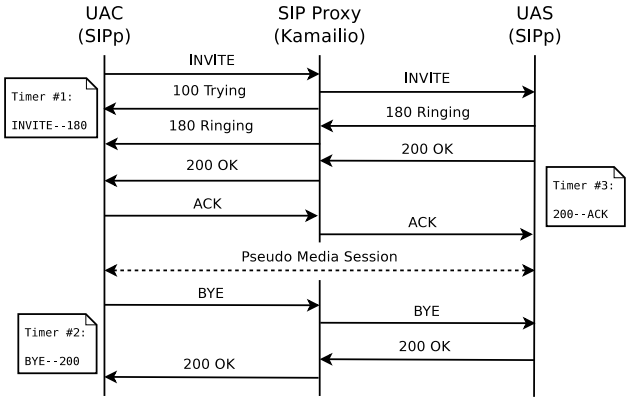

Figure 20: Measurement Scenario

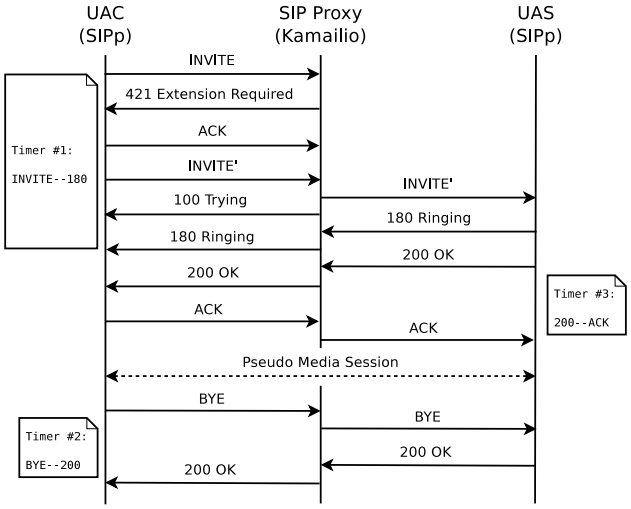

Figure 21: Measurement Scenario with Enforced Use of Extension

IV. Measurements

Although we minimized the changes to the existing VoIP infrastructure, the provider still has to be aware of the *sctp-tunnel* extension indicated within the SIP messages. Whether the extension is stated or not, the provider has to use different message handling and routing. It is thus important to know how much network and computational overhead our extension creates on the provider's side and how much more UA-to-UA time the message routing takes.

Note that the following measurements do not consider the impact of SCTP. SCTP is used as the underlying transport protocol of the UA-to-UA media session only, whereas SIP messages still use UDP. In result, a SIP proxy does not need to be adapted to use another transport protocol. On the other hand, media gateways (not considered by the following measurements) need to be altered to conform to our approach.

A. Testbed, Scenarios

We used three nodes (each with 2 x AMD Opteron 244 CPU (1.8 GHz), 4 GB RAM, Gigabit Ethernet Interconnection) to setup one SIP proxy (Kamailio [25], v3.0.3) and two UAs (SIPp [13], v3.1) that generated and processed a various number of SIP calls. Kamailio has been configured to use 1024 MB of memory, to create four processes, and its log level was set to zero.

In general, we measured three scenarios: a) default behavior of the proxy, b) modified behavior of the proxy where the UAs already indicated the use of the *sctp-tunnel* extension, and c) the modified behavior of the proxy without initial indication by the UAs. The third scenario is the most expensive one since the provider needs to reject incoming invitations first, and then has to deal with the reformulated ones. In addition, we measured d) the SIPp-SIPp-interconnectivity to determine the overhead of Kamailio in general.

In scenarios a) and b), the UAC and the UAS send and receive SIP messages according to Figure 20. According to Figure 4, the proxy stays in the route for the whole call. Scenario c) requires three more messages at the beginning: the first *INVITE* will be rejected with a *421* response that has to be *ACK*ed (see Figure 21).

In order to implement the modified proxy behavior, we used a prototypical approach that only required modification of the Kamilio routing logic that is defined in the Kamilio configuration file (`kamailio.cfg`). The relevant excerpts from the file are shown in Listing 1. The implementation checks if the *Require* header is present and searches for the option tag that identifies the SCTP Tunneling Extension. If it is not present (Figure 21), it sends the *421* response attaching the *Require* header with the appropriate option tag indicating which extension is required. If the *Require* header with the correct option tag is present, routing proceeds as usual (Figure 20; see the first **return** statement in Listing 1).

Each call generates three round-trip time values: RTT #1 represents the delay of a UAS's response including Kamailio action (such as lookup and extension verification); RTT #2 represents the delay of a UAS's response in case the request can be forwarded immediately; RTT #3 represents the delay of a UAC's feedback. Each series lasted five minutes, using a constant call frequency (between 1 and 1000 calls per second). The proxy and the UAs were restarted for each frequency.

B. Results

For all scenarios and each frequency, we calculated the corresponding median and quartile values for each RTT. As expected, in scenarios a)–c), the values of RTT #2 and RTT #3 are nearly the same (see Figures 23, 24). The SIPp-SIPp interconnection's second and third RTT are ∼0.25–0.55 ms lower only.

The comparison of RTT #1 is shown in Figure 22. Again, one can see the additional time required (∼0.5–0.6 ms) when

```
#!KAMAILIO
#!define WITH_SCTP_TUNNELING

/* other defines, parameters,
 * and module configuration */

####### Routing Logic ########

# main request routing logic

route{
  /* processing of related requests */

  # make sure UAC is using the
  # SCTP tunneling extension
  if (is_method("INVITE")) {
        route(REQUIRE);
  }

  /* processing of initial requests */

  /* other processing */
}

/* other route blocks */

route[REQUIRE] {
#!ifdef WITH_SCTP_TUNNELING
  if (is_present_hf("Require")) {
    if (search("^Require:.*sctp-tunnel.*")) {
      return;
    }
  }
  append_to_reply("Require: sctp-tunnel\r\n");
  send_reply("421", "Extension Required");
  exit;
#!endif
  return;
}
```

Listing 1: SIP Proxy Implementation: Kamailio Configuration File (kamailio.cfg)

Kamailio is put between the SIPp instances. Furthermore, we expected the overhead of the header verification to be very small since we only slightly modified the routing logic of Kamailio. This small RTT increase can be seen when comparing the values of scenarios a) and b).

In scenario c), where the proxy had to enforce the use of the SIP extension, RTT #1 increases a little more. This happens because Kamailio is involved one more time and three more messages are sent until the first callee's response is received by the inviting caller.

V. Conclusion and Future Work

In this paper, we presented several scenarios motivating the need for media connectivity awareness for SIP providers, including payment, reputation, forensics, and call detail record analysis. We identified specific requirements a solution must fulfill. Two requirements should be emphasized. The derived connectivity status must be genuine. This is important because the provider uses the obtained information

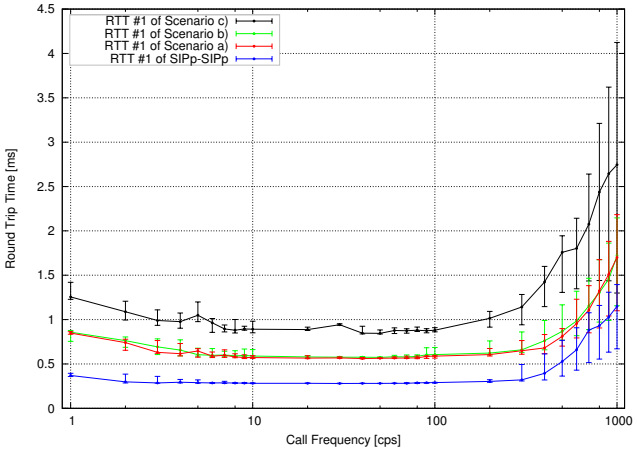

Figure 22: Comparison of RTT #1

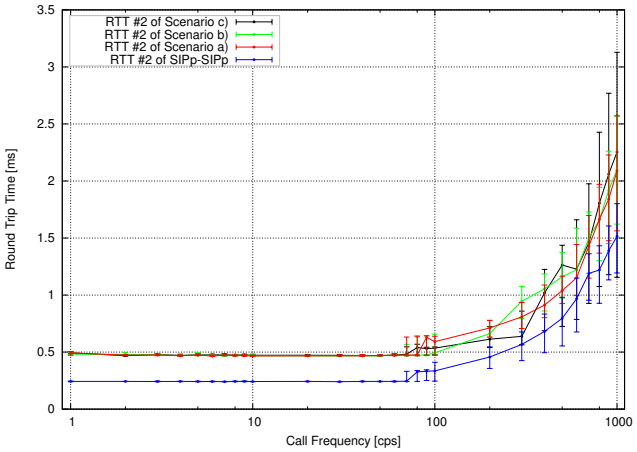

Figure 23: Comparison of RTT #2

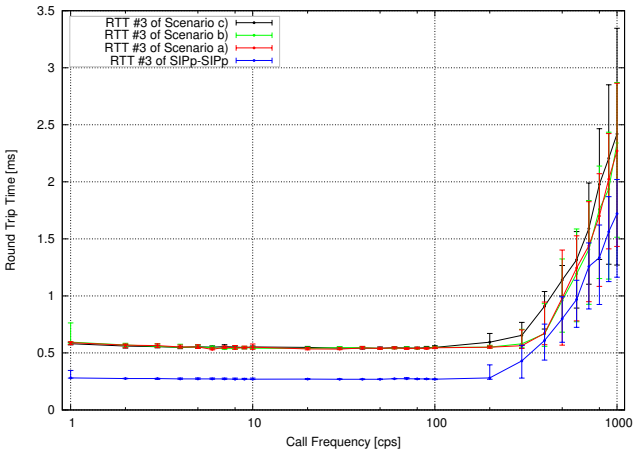

Figure 24: Comparison of RTT #3

to draw futher conclusions that potentially impact the user directly, like demoting a user's reputation regarding SPIT. It is similarily important that the solution is compatible with existing protocols as it would otherwise not be applicable in existing VoIP infrastructures.

In our solution, the provider is *implicitly* informed about the media connectivity: The SIP provider can draw genuine conclusions by simply analyzing the messages it is routing. The UA, however, needs to alter its behavior. This behavior is specified by way of a new SIP extension and its usage can be enforced by the provider.

To reduce the overhead of media connectivity detection, we propose to use SCTP for media transport. This requires a slight extension of SDP.

The measurements showed that the overhead introduced by our solution is negligible, as long as the UAs indicate the use of our extension from the beginning. In addition, our approach can easily be integrated into existing VoIP infrastructures as it fully conforms to existing protocols. If a UA is not aware of our extension it is at the discretion of the provider to proceed with the call (without the ability to conclude media connectivity) or to reject it.

Several other approaches to the connectivity awareness problem are presented and contrasted in this paper. We favour the implicit approach as it requires the least changes to the involved protocols, minimizes protocol overhead, and ensures that the connectivity status is genuine even if a user agent lies.

Even though none of the related work presented in Section II by itself enables a SIP provider to gain awareness of the media connectivity status, one work – Connectivity Preconditions [3] – could achieve this with a slight modification and combination with parts of our approach. The semantics of the Connectivity Preconditions SIP extension would have to be altered from "SHOULD" to "MUST" and the SIP Proxy would have to be able to enforce the use of the precondition. In addition, the behavioral rules for the user agents introduced in Section III would have to be observed. Lastly, each UA would have to verify connectivity of every single media stream in both directions.

Future work will deal with Quality of Service (QoS) aspects. Besides a lack of connectivity, low quality can also cause a call to be aborted prematurely by one of the participants. We therefore need to conduct further investigation in order to deal with this problem.

In addition, as the use of reliable provisional responses would be beneficial to our solution (cf. Section III-A), protocol interactions and possible implications need to be investigated.

REFERENCES

[1] Stefan Gasterstädt, Markus Gusowski, and Bettina Schnor. SIP Providers' Awareness of Media Connectivity. In Pascal Lorenz, Tibor Gyires, and Iwona Pozniak-Koszalka, editors, *10th International Conference on Networks (ICN 2011)*, pages 157–163. IARIA, January 23rd–28th, 2011.

[2] Alessandro Amirante, Simon Pietro Romano, Kyung Hwa Kim, and Henning Schulzrinne. Online Non-Intrusive Diagnosis of One-Way RTP Faults in VoIP Networks Using Cooperation. In Georg Carle, Helmut Reiser, Gonzallo Camarillo, and Vijay K. Gurbani, editors, *Proceedings of the 4^{th} International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm 2010)*, pages 153–160, Munich, Germany, August 2^{nd}–3^{rd}, 2010. Technical University Munich.

[3] F. Andreasen, G. Camarillo, D. Oran, and D. Wing. Connectivity Preconditions for Session Description Protocol (SDP) Media Streams. RFC 5898 (Proposed Standard), July 2010.

[4] Vijay A. Balasubramaniyan, Mustaque Ahamad, and Haesun Park. CallRank: Combating SPIT Using Call Duration, Social Networks and Global Reputation. In *Proceedings of the 4^{th} Conference on Email and AntiSpam, CEAS 2007*, August 2^{nd}–3^{rd}, 2007.

[5] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005.

[6] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997.

[7] G. Camarillo and P. Kyzivat. Update to the Session Initiation Protocol (SIP) Preconditions Framework. RFC 4032 (Proposed Standard), March 2005.

[8] G. Camarillo, W. Marshall, and J. Rosenberg. Integration of Resource Management and Session Initiation Protocol (SIP). RFC 3312 (Proposed Standard), October 2002. Updated by RFCs 4032, 5027.

[9] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (Standard), January 2008.

[10] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[11] R. Fairlie-Cuninghame. Guidelines for specifying SCTP-based media transport using SDP. Internet-Draft draft-fairlie-mmusic-sdp-sctp-00, Internet Engineering Task Force, May 2001. Work in progress.

[12] T. Friedman, R. Caceres, and A. Clark. RTP Control Protocol Extended Reports (RTCP XR). RFC 3611 (Proposed Standard), November 2003.

[13] Richard Gayraud, Olivier Jacques, et al. SIPp: An Open Source Performance Testing Tool for SIP [v3.1]. http://sipp.sourceforge.net, retrieved: January 22^{nd}, 2012.

[14] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[15] M. Hasebe, J. Koshiko, Y. Suzuki, T. Yoshikawa, and P. Kyzivat. Example Call Flows of Race Conditions in the Session Initiation Protocol (SIP). RFC 5407 (Best Current Practice), December 2008.

[16] M. Holdrege and P. Srisuresh. Protocol Complications with the IP Network Address Translator. RFC 3027 (Informational), January 2001.

[17] C. Huitema. Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP). RFC 3605 (Proposed Standard), October 2003.

[18] C. Jennings, J. Fischl, H. Tschofenig, and G. Jun. Payment for Services in Session Initiation Protocol (SIP). Internet-Draft draft-jennings-sipping-pay-06, Internet Engineering Task Force, July 2007. Work in progress.

[19] Alan B. Johnston. *SIP: Understanding the Session Initiation Protocol*. Artech House, 2^{nd} edition, 2004.

[20] Stefan Liske, Klaus Rebensburg, and Bettina Schnor. SPIT-Erkennung, -Bekanntgabe und -Abwehr in SIP-Netzwerken. In David Buchmann and Ulrich Ultes-Nitsche, editors, *15. ITG/GI-Fachtagung "Kommunikation in Verteilten Systemen" (KiVS 2007) – Workshop "Secure Network Configuration" (NetSec 2007)*, pages 33–38, Fribourg, February 26th– March 2nd, 2007. DIUF, Universität Fribourg.

[21] S. Loreto and G. Camarillo. Stream Control Transmission Protocol (SCTP)-Based Media Transport in the Session Description Protocol (SDP). Internet-Draft draft-loreto-mmusic-sctp-sdp-05, Internet Engineering Task Force, February 2010. Work in progress.

[22] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), April 2010.

[23] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[24] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966.

[25] Ramona-Elena Modroiu, Bogdan Andrei Iancu, Daniel-Constantin Mierla, et al. Kamailio (OpenSER) [v3.0.3]. http://www.kamailio.org/, retrieved: January 22^{nd}, 2012.

[26] Jörg Ott and Lu Xiaojun. Disconnection tolerance for SIP-based real-time media sessions. In *MUM '07: Proceedings of the 6^{th} international conference on mobile and ubiquitous multimedia*, pages 14–23, New York, NY, USA, 2007. ACM.

[27] A. Pendleton, A. Clark, A. Johnston, and H. Sinnreich. Session Initiation Protocol Event Package for Voice Quality Reporting. Internet-draft, Internet Engineering Task Force, Mar. 2010. Work in progress.

[28] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[29] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[30] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093.

[31] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265 (Proposed Standard), June 2002. Updated by RFCs 5367, 5727.

[32] J. Rosenberg. The Session Initiation Protocol (SIP) and Spam. Internet-Draft draft-ietf-sipping-spam-03, Internet Engineering Task Force, October 2006. Work in progress.

[33] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), April 2010.

[34] J. Rosenberg and G. Camarillo. Examples of Network Address Translation (NAT) and Firewall Traversal for the Session Initiation Protocol (SIP). Internet-Draft draft-rosenberg-sipping-nat-scenarios-03, Internet Engineering Task Force, July 2004. Work in progress.

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

[36] J. Rosenberg and H. Schulzrinne. Reliability of Provisional Responses in Session Initiation Protocol (SIP). RFC 3262 (Proposed Standard), June 2002.

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

[38] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.

[39] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard), July 2003. Updated by RFC 5761.

[40] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. Updated by RFCs 5506, 5761, 6051, 6222.

[41] D. Senie. Network Address Translator (NAT)-Friendly Application Design Guidelines. RFC 3235 (Informational), January 2002.

[42] R. Sparks and T. Zourzouvillys. Correct Transaction Handling for 2xx Responses to Session Initiation Protocol (SIP) INVITE Requests. RFC 6026 (Proposed Standard), September 2010.

[43] Mark Spencer et al. Asterisk – The Open Source Telephony Projects. http://www.asterisk.org/, retrieved: January 22^{nd}, 2012.

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

[45] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFC 6096.

[46] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad. Stream Control Transmission Protocol (SCTP) Partial Reliability Extension. RFC 3758 (Proposed Standard), May 2004.

[47] D. Yon and G. Camarillo. TCP-Based Media Transport in the Session Description Protocol (SDP). RFC 4145 (Proposed Standard), September 2005. Updated by RFC 4572.