Disconnection Tolerance for SIP-based Real-time Media Sessions

Jörg Ott
Helsinki University of Technology
Networking Laboratory
jo@netlab.tkk.fi

Lu Xiaojun
Helsinki University of Technology
Networking Laboratory
xlu@netlab.tkk.fi

ABSTRACT

Mobile users may experience short disruptions and premature call terminations (not just) when moving while engaged in a (multimedia) conversation. While much emphasis is put on enabling seamless mobility, recovery in case of (temporary) service failures has received little attention. In this paper, we explore the technical aspects of disconnection tolerance mechanisms for SIP-based mobile communications. We present standards-compliant detection and recovery mechanisms, provide a brief experimental evaluation based upon packet traces, and report on our prototype implementation.

Categories and Subject Descriptors

C.2.6 [**Computer Communication Networks**]: Internetworking; D.4.4 [**Operating Systems**]: Communications Management

General Terms

Design, Reliability, Experimentation

Keywords

Disconnection tolerance, SIP, Mobility, Multimedia

1. INTRODUCTION

IP-based voice and multimedia communication is widespread in use in the Internet, with the Session Initiation Protocol (SIP) [27] being the dominant open signaling protocol today. Typical deployment scenarios in the Internet assume mostly stationary users—at home, in the office, in a hotel, or in a public hot-spot. For such wireless SIP communication, e.g., using IEEE 802.11, network capacity and QoS mechanisms for voice prioritization (e.g., [13, 32]) have been investigated. But also mechanisms for SIP-based node mobility have been proposed at the link, IP, and application layer (e.g., [30, 8]). In addition to the open Internet, closed networks such as the 3GPP Internet Multimedia Subsystem (IMS) support node mobility using a combination of mechanisms at different layers, within a single link layer technology as well as across different physical networks and service providers. In this paper, we focus on the general Internet case, but the concepts presented can be adapted to work in the 3GPP context as well.

Mobility mechanisms seek to minimize the handoff delay when moving from one network point of attachment to the next in order to provide seamless communications without noticeable increasing packet delays or losses. The implicit assumption is that *there will be a next point of attachment* to perform the handoff to without service disruption. As experience tells us, e.g., when driving on a highway or riding on a train, this assumption does not necessarily hold. Even if handoffs to other points of attachments are in principle seamless, mobile phone calls may easily experience temporary outages, e.g., due to changes in the radio conditions. Or they may get disconnected altogether, e.g., because of incomplete coverage or a saturated "next" base station or wireless access point. The same applies to packet data services and thus equally to IP-based multimedia services.

Dealing with these kinds of disconnections is left to the users: they may re-dial and, if successful after one or more attempts, run a "user-to-user protocol" to mutually synchronize on the last heard statements of the conversation prior to the disconnection. Typically, it is the turn of the user who experienced the disconnection to wait until the device reports that network coverage is available again and invokes the next communication attempt (but, of course, both users will often try in parallel). Alternatively, or if they cannot get through, they give up, wait for a longer period of time to retry, or revert to voice mail or text messaging.

This entire procedure appears cumbersome and unnecessary. In this paper, we propose a set of mechanisms to detect such disconnections and deal with them as far as possible in an automated fashion. We do not make any assumptions about the underlying link layers and where the disconnection occurs but simply use end-to-end mechanisms at the transport layer and above. In section 2, we briefly review the basic SIP signaling which we build upon and review related work on mobility for SIP and on disconnection tolerance in general in section 3. We introduce our disconnection models and present our mechanisms for disconnection-tolerant SIP (DT-SIP) in section 4. We finally provide a first experimental analysis based upon packet traces and report on our prototype implementation in section 5. We conclude this paper with a brief assessment in section 6.

2. SIP CALL SIGNALING

SIP [27] is a protocol for creating, modifying and terminating sessions between two *user agents (UAs)* which are identified by their respective SIP URIs. For rendezvous purposes, SIP defines a variety of logical functions: UAs register the mapping of their persistent URI (*Address-of-Record, AoR*) to the contact addresses of one or more devices with a *registrar*. This mapping can be

MUM'07, December 12–14, 2007, Oulu, Finland.

retrieved from a *location server* function which is usually consulted by SIP *proxy* and *redirect* servers. Proxies forward incoming SIP requests on behalf of the originator towards the target whereas redirect servers simply return the next hop contact address to the originator which then re-submits the retargeted request. A typical SIP server implementation combines registrar, location, and redirect/proxy server functions and may offer additional services for media handling such as a UA for gatewaying to other networks and media server functions for announcements and voice mail. For two random users interacting, (at least) one SIP server will be involved on each side (see figure 1 top), yielding the typical SIP "trapezoid".

As depicted in figure 1, each SIP UA uses a REGISTER message to bind its AoR to its current contact address (i.e., IP address and port number) which is confirmed by a 200 OK from the registrar.[1] During the registration handshake, the validity timeout for the registration is negotiated; the UA must refresh its registration before this timeout expires to remain reachable. The UA will also re-register whenever its local address changes.

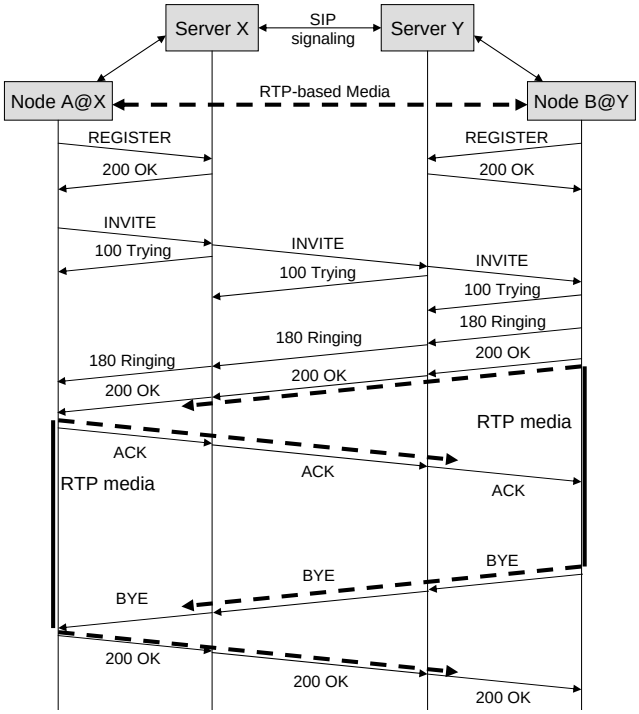

Figure 1: Basic SIP signaling

In a very basic dialog setup, the originator sends an INVITE message (via its and the remote proxy) to the target UA which usually first indicates that the user is alerted (180 Ringing) and responds with a 200 OK when the user accepts the call, which is confirmed by an ACK.[2] The two SIP proxies may decide to remain in the SIP signaling path of a dialog by performing *record-routing* or may let subsequent SIP messages flow directly between the UAs. During the course of a normal SIP dialog, no further SIP messages are exchanged until the dialog is terminated by a BYE message sent by either UA, again confirmed by a 200 OK. A SIP dialog is uniquely identified by a set of headers (simplified: To:, From:, and Call-Id:), so that incoming requests can be matched to an existing dialog.

The INVITE transaction effects state creation at both UAs and (as part of this) causes the media exchange in the dialog to commence; the inverse happens as result of a BYE transaction. SIP itself does not define any applications to be used in the context of a dialog, but provides a mechanism (using the Session Description Protocol, SDP, [11]) to configure arbitrary media sessions during the *offer-answer exchange*, a two-way handshake of SDP messages, typically piggybacked on an INVITE and a 200 OK. The session negotiation includes parameters such as codecs, codec parameters, and particularly the transport addresses for the media streams. If the addresses of one node change, e.g., due to mobility, or if another reconfiguration of a media session shall take place, the offer-answer exchange is repeated, typically by means of a *re-INVITE*. Real-time media streams are usually carried in RTP [29] packets sent over UDP; RTCP messages sent in roughly regular interval accompany the media streams to report transmission characteristics and reception statistics.

SIP messages may be exchanged via UDP, TCP, and SCTP, optionally with an additional security layer (e.g., TLS) on top. SIP state is independent of the lower layer and hence transport connections may be torn down and re-established without affecting a SIP dialog.

3. RELATED WORK

Two areas of related work are particularly relevant to our research presented in this paper: mobility support for SIP and disruption tolerance for mobile communications.

3.1 SIP and Mobility

A variety of mechanisms have been proposed to achieve node mobility with SIP. Basic link layer mechanisms (e.g., handoffs between WLAN APs or cellular base stations) are basically transparent to the IP layer and above, yet careful engineering may be required to achieve seamless handovers [8]. Mobile IP [23] paired with numerous optimizations to reduce the impact of handoffs (e.g., [14, 12, 33]) may support mobility at the IP layer transparent to the applications.

SIP has built-in support for basic user mobility via its dynamic user registration and rendezvous mechanism as well as through re-INVITEs. In addition, many approaches have been investigated to allow user mobility during ongoing session with SIP (of which we can review only a few). The entire spectrum of pre-call and mid-call mobility for SIP and RTP has been discussed in [30], where the authors also discuss the limitations of mobile IP. An operator architecture for vertical handover of SIP dialogs across different networks is discussed in [36].

Achieving seamless multimedia session mobility with SIP requires efficient operation of the entire handover process: a mobile node needs to locate the attachment point, become authorized, obtain an IP address (e.g., via DHCP), among others [21]. Optimizations for this process have been suggested to use multiple radios (so that no or only few packets are lost) [24], to realize soft handoff techniques with support from the base stations redirecting packets so that these do not get lost [1], and to integrate SIP and mobile IP mechanisms [35]. Finally, fast handover mechanisms relying only on application layer signaling have been proposed [8] which avoids the otherwise potentially complex interdependencies and handles most aspects without network support in the application.

The above mechanisms seek to address changing a mobile node's point of attachment to the network with minimal disruptions for

[1]For simplicity of the presentation, we do not discuss the authentication mechanisms which would usually be applied.

[2]The 100 Trying messages are hop-by-hop confirmations of the message receipt and just shown for completeness. We do not discuss call setup failures.

the real-time service. But all of them assume that there will be a next attachment point available so that SIP calls will simply be disrupted (and terminate) if this is not the case. In the remainder of this paper, we present a complementary mechanism that becomes active as soon as seamless mobility fails.

3.2 Delay-/Disruption-tolerant Networking

Dealing with or preventing mobility-incurred disruptions in communications for mobile nodes has been investigated in numerous research activities which address a wide range of technical issues: from dealing with dynamically changing IP addresses when switching the network point of attachment (e.g., using *Migratory TCP* [34] or the *Host Identity Protocol, HIP* [16]) to preventing disconnections due to TCP timeouts [10, 3] to (proxy-based) architectures to support legacy and novel applications in disconnection-prone mobile environments. The latter include the MOCCA architecture as developed for mobile Internet access from cars in the FleetNet project [2], the *Drive-thru Internet* architecture and its Persistent Connection Management Protocol (PCMP) [19], and the DHARMA project architecture for mobile and nomadic computing [15].

While the above approaches primarily address the failure of a single link close to the mobile node (in most cases the last hop), the research field of *Delay-tolerant Networking (DTN)* [9] approaches communication in challenging networking environments with a more encompassing scope.[3] Challenges may include, e.g., long communication delays, low data rates, high error rates, (frequent) predictable or unpredictable disconnections, and/or only short connectivity periods. These challenges generally lead to an environment in which end-to-end paths may not exist and end-to-end interactions may take extended periods of time to complete. To cope with such conditions, DTN exclusively relies on message-based asynchronous communications, the messages being exchanged following the *store-carry-and-forward* paradigm. While originally developed for space communications, DTN finds its application in sensor networks and in mobile communications including mobile ad-hoc and infrastructure-based networks.

The aforementioned approaches to handling disconnections are primarily applicable to *elastic*—i.e., sufficiently delay-tolerant—applications. This includes applications where a temporary interruption of the data transfer, while likely annoying to the user, still does not impact the usefulness of the received media. It may take longer to send or receive an email, retrieve a web page, or synchronize a distributed calender, but the result will finally be correct. In contrast, interactive real-time media as exchange inside a SIP call suffer from disruptions so that the aforementioned approaches, which trade timeliness off for reliability are not immediately applicable [18].

4. DISCONNECTION-TOLERANT SIP

As discussed above, a SIP dialog comprises two UAs which may be connected via arbitrary access networks to the Internet—or which may operate in an ad-hoc environment without any Internet access. While SIP servers (or peer-to-peer nodes) may facilitate dialog establishment, they are not necessarily part of any subsequent call signaling so that any disconnection handling has to be performed by the UAs.

Link or path failures may occur anywhere in the network. In particular, neither of the UAs' access links needs to be impacted so that detection and recovery mechanisms cannot rely on *link layer indications* from the local interfaces.[4] The consequences of a link or path failure may differ: Considering figure 1, from node A's perspective, the path the other UA may be impaired and/or the path towards its own SIP server X. Node B may be affected by a different failure at the same time, i.e., (partly) overlapping with A's disconnection. For a real-time conversation between A and B, the reason for, the location, and the number of failures is not relevant; however, the resulting duration is. We provide a rough classification into three categories:

- Short *audio disruptions* last a couple of seconds (e.g., less than 15 s) and the SIP dialog remains established. Hence, no call signaling recovery mechanisms are needed. However, a short disruption leads to noticeable audio loss which should be recovered.

- Middle: *Call interruptions* occur if the disconnection lasts longer than a (provider-defined) threshold (e.g., 15 s) which causes the dialog to be terminated. Obviously, audio data gets lost and the users are informed about the dropped call. If the connectivity is re-established within an acceptable *reconnection period*[5] (e.g., 60 s), we consider automatic reconnection of the SIP dialog and recovery of the lost audio to be useful.

- Long disconnections lead to *call termination* and connectivity is not re-gained within the reconnection period so that no automated re-establishment should occur. In such a case, the synchronous conversation can be automatically turned into an asynchronous interaction (similar to voice mail) without requiring explicit action by the users.

To assist users in case of disconnections, the communication within a SIP dialog needs to be monitored by the UAs and the latter must be prepared at any time to invoke the corresponding recovery actions. Invocation must work incrementally as they cannot tell up front how long a failure will persist.

4.1 Connectivity Loss Detection

The first step towards disconnection-tolerant SIP is detecting the loss of connectivity. A SIP UA has typically (a) a SIP signaling relationship directly to the peer or indirectly via a SIP server and (b) an RTP-based media stream (we assume audio only). In addition, as noted above, link layer indications may speed up the detection of link (non-)availability at the last hop.

4.1.1 SIP-based Detection

In a simple SIP call, the involved UAs do not perform any SIP transactions beyond the initial INVITE and the final BYE so that the basic SIP signaling cannot be relied upon. As the transport connection used for signaling has no relevance to the SIP state machine, relying on lower layer keep-alive mechanisms is also questionable. Even if re-INVITEs are necessary, e.g., to re-configure transport addresses used with *Interactive Connectivity Establishment (ICE)* [25], see below, these cannot be predicted and would rather occur at the beginning of a session. Only SIP session timers [7] can be used to generate SIP messages in regular and predictable intervals. However, SIP servers have a say in the timer intervals and may prevent too short timeout values. Timeout values in the

[3]The Delay-tolerant Networking Research Group (DTNRG) in the Internet Research Task Force (IRTF), http://www.dtnrg.org/, defines protocol specifications for DTN-based communications such as [5, 31].

[4]Nevertheless, such hints may be useful for optimizations, e.g., to efficiently decide when to attempt a recovery.

[5]Before the users mentally switch to another task.

order of minutes are, however, unsuitable for efficient disconnection detection.

Finally, note that the SIP signaling path may follow an overlay and thus a different path than the media stream. Therefore, the (non-)availability of the signaling path does not allow inferring the status of the media path. In summary, we consider SIP signaling unsuitable for the purpose of disconnection detection and rely on the media session(s) instead. However, the SIP signaling and particularly the SDP-based media configuration provide context information—e.g., whether a call is placed on hold (and thus media streams are paused) and if silence suppression is enabled for the chosen codec—which are relevant for properly interpreting the media streams.

4.1.2 Media-based Detection

SIP audio calls use RTP sessions for the media exchange. An RTP session consists of the RTP-based media stream and an associated RTCP-based control channel. Furthermore, if NAT traversal techniques are employed, additional STUN [26] control messages may be multiplexed on both the RTP and the RTCP channels:

- *RTP:* During an audio conversation, both nodes send continuous audio streams as a sequence of RTP packets. However, if silence suppression is enabled, this packet stream is only generated while the respective user is talking. Depending on the codec, specific silence packets may occasionally be sent during silence periods (e.g., once per second) to allow the receiver to fill in background noise. If no silence suppression is used (which the receiver can only detect heuristically) or if silence packets are sent (which can be inferred from the SDP codec parameters), monitoring the media stream for gaps in the otherwise regular and frequent packet reception is the most efficient cue that a disconnection occurred: as audio and silence packets are sent frequently[6], loss of connectivity may be detected within a few seconds (already accounting for IP packet loss). However, in the general case, a continuous reception of media packets cannot be guaranteed (e.g., when using the G.711 PCM audio codec with silence suppression but without the comfort noise according to RFC 3389 [37]). Therefore, additional mechanisms are needed.

- *RTCP:* RTCP packets are exchanged between the endpoints to report transmission and reception statistics and allow monitoring the path quality. RTCP packets are sent independently of silence suppression and a call put on hold. Therefore, the absence of several consecutive expected RTCP packets can be taken as an indication of a media path failure. In a two-party call, RTCP packets are sent in deterministically calculated intervals Td so that they use 2.5% of the session bandwidth per peer, dithered between $0.5 \times Td$ and $1.5 \times Td$. With $Td \geq 5\,s$, this usually yields a maximum distance of 22.5 s for consecutive RTCP intervals. This defines failure detection latency for RTCP monitoring if three missed RTCP packets are taken as the threshold. The RTCP interval can be decreased by using a different RTP profile [22] which eliminates the 5 s minimum: to about 3 s for an 8 kbps media stream (assuming 80 byte RTCP packets) and even further by explicitly allowing for a higher RTCP bitrate [4], so that the detection latency can be reduced.[7]

- *STUN:* Both RTP and RTCP communication may require STUN packets to be exchanged for NAT traversal to detect the externally visible transport addresses and to maintain the NAT bindings alive (during silence periods). When used stand-alone STUN may operate between an endpoint and a STUN server (the latter of which is usually run by the SIP service provider) so that these STUN exchanges do not relate to the end-to-end media path. However, if ICE [25] is used (which can be observed from the SDP offer/answer exchange), STUN packets are also sent between the two peers as continuous connectivity checks and to maintain NAT bindings active even during silence periods. STUN packets regularly exchanged between the peers thus support disconnection detection on both the RTP and RTCP channels and, for RTP, particularly irrespective of the chosen codec and the silence suppression behavior.

The above considerations can be summarized in the resulting detection algorithm depicted in figure 2. The detection algorithm relies on three independent detectors, two of which are enabled or disabled depending on negotiation during the offer/answer exchange: if a codec is negotiated together with a suitable comfort noise payload or a comfort-noise supporting codec is chosen (e.g., G.729), RTP-based detection is enabled and if ICE is negotiated, STUN tracking is enabled. In addition, the RTCP channel is monitored and if RTCP packets are received from the peer, RTCP-based detection is activated. If none of the mechanisms is enabled—e.g., for a G.711 call without comfort noise to a non-RTP-compliant peer which does not support RTCP—the disconnection detection is disabled.

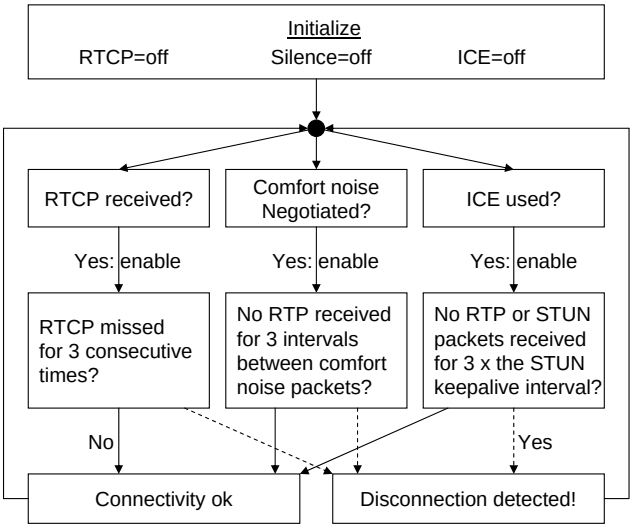

Figure 2: Disconnection detection algorithm

Once a disconnection is detected, its duration is monitored and the actions for the respective duration class are taken as discussed in the following subsection. To detect regaining connectivity, RTCP packets continue to be sent while disconnected and may be augmented by STUN probe packets or RTP no-op packets if STUN is not available: as soon as packets are received again, the failure is over. Disconnections and regaining connectivity may be indicated to the user by audible signals.

[6]Note that for some silence suppression mechanism, the interval is implementation specific [37] and may thus not be easy to predict.

[7]One major issue is still missing RTCP support in various SIP clients.

4.1.3 Detection Accuracy and False Positives

The three detection mechanisms work passively and are complementary. The observable incoming packet flows—RTP (media vs. silence), RTCP, and STUN—exhibit different characteristics as discussed above, so that 1) the detectors operate at different timescales and 2) have different susceptibility to packet losses. Therefore, if multiple detectors are active, one detector may indicate a disconnection event while another one does not (yet). In general, RTP-based detection has the finest granularity (as long as audio packets are sent) whereas the relative reactivity of silence-, STUN-, and RTCP-based detection may vary depending on the precise media stream configuration. With this in mind, we suggest the following resolution mechanism:

Whenever RTP-based detection is enabled, it takes precedence and is used to infer disconnection events. Regardless of whether or not RTP-based detection is enabled: even if other detectors suggest a disconnection, such an event is ignored as long as RTP packets or silence packets continue to be received. This matches the semantics of a conversation: as long as the user hears something, the call should continue. Only if no RTP packets (media or silence) are received, STUN- or RTCP-based detection events are considered. In this case, both are treated equal and the first indication is handled as a disconnection event. This procedure should prevent false positives due to detectors contradicting each other.

However, false positives are still possible if the detectors are too sensitive given the conditions of the underlying network paths. Particularly RTP-based detection can allow for fairly fine-grained tuning as many packets may arrive per second under normal circumstances. A careful trade-off needs to be found between, on one hand, not unnecessarily extending the period of uncertainty (while the user continues to speak) and, on the other hand, prematurely invoking repair mechanisms which may then disturb more by further interrupting the conversation than they help. While upper bounds representing a user's patience are probably needed (should both users agree?), adaptive timeout calculation mechanisms are probably needed within this bound (see also section 5.1). We leave further considerations on dynamic adaptation for future study.

Finally, note that all three detectors presented above implicitly assume symmetric connectivity and infer from a interruption of packet reception that their own media transmission is affected, too. Since our aim is support of conversational media, this assumption appears justified.

4.2 Handling Disconnections

Disconnections lead to a loss of segments of a voice conversation and may lead to call termination. According to the above classification, we choose to:

1. *Record the audio* sent by each endpoint so that it can be delivered after the disconnection is over; this is independent of the type of disconnection.
2. *Recover the audio* saved before; the mechanisms chosen vary depending on the duration of the outage.

Dealing with disconnections has to be progressive since, at the time a path failure is detected, it is unknown how long it will persist and hence which measures need to be taken. Because detection naturally incurs some delay, basic recovery measures need to be taken pro-actively in all calls. The recovery method chosen is then depending on the duration of the disconnection.

4.2.1 Pro-active Audio Recording

We record the audio data for the outgoing media stream in addition to generating the RTP packets of the encoded audio. Because of the basic RTCP detection latency of some 20 s, we use a ring buffer for 30 s of audio and also maintain sequence number and timestamp information along with the media. The extra space allows conveying more context information. This recording can grow up to two minutes to cover the case that the local user wants to continue speaking even after the call disconnection has been signaled.

The recorded audio can be streamed as a sequence of RTP packets, to a human in an ongoing SIP session or to a voice mail box. Or it can be delivered in a single message, e.g., to be used as an email attachment or as an instant message.

4.2.2 Recovery Mechanisms

The recovery capabilities and the resulting disruption observed by the user differ depending on the disconnection duration.

Short disconnections are handled by replaying the RTP stream as soon as the connectivity is re-gained (figure 3). Each client starts at a suitable point of the recording, being conservative since it cannot know that the peer received (the latter may discard older duplicates based upon the RTP sequence numbers). For short and quickly detected disruptions, this results essentially in a few seconds of silence, followed by the continuation of the previously heard sentence. During playback, the speech continues to be recorded and played back subsequently; the resulting delay is reduced over time by removing silence periods. We assume that users will notice the silence (and latest the disconnection signal), finish their sentence, and do not continue speaking much longer until recovery has succeeded.

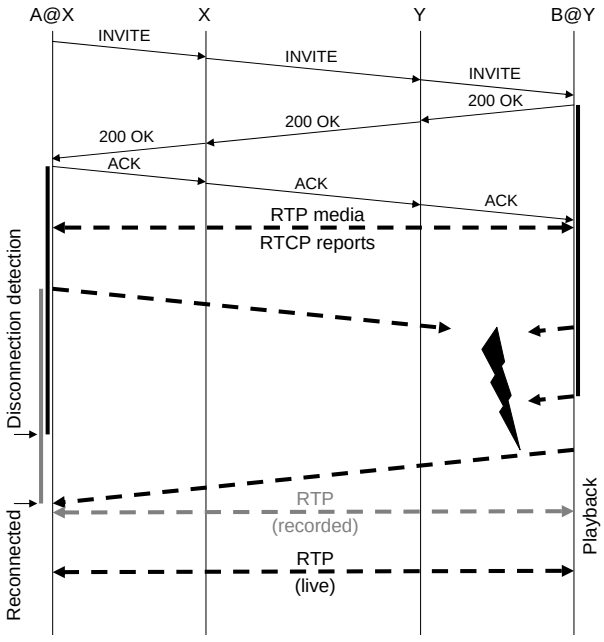

Figure 3: RTP-based recovery

Call interruptions are handled slightly differently because the media channel is lost and the call and media channel need to be re-established. Assume that user B got disconnected (figure 4): After disruption detection and (local) call termination, both UAs

attempt to re-establish the call using SIP INVITE messages.[8] Once this succeeds, the recorded voice is played back after an indication tone and, subsequently, the users can continue their conversation. The recovering UA should always be the one who established the original call in order to maintain the charging characteristics of the original call. If charging is not an issue, both UAs should try to recover and use randomized timers and a tie-breaking mechanism to deal with the possibility that accidentally two dialogs in opposite directions get established simultaneously (see section 4.2.3).

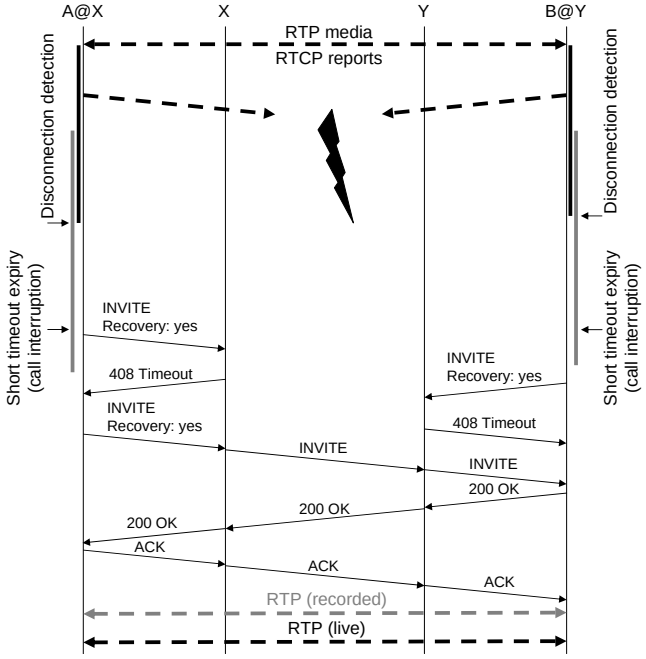

Figure 4: SIP-based recovery

Call terminations occurs if the automated re-establishment fails for a certain period of time. In this case, a UA indicates this to the user and allows the user to record additional voice and transforms the entire recording into a voice mail—subject to the user's consent. The UA which stays connected can either send an INVITE and play the media stream via RTP to the voice mail server. Or the UA may open a messaging session with an instant message store of user B and deliver the voice contents as an instant message—which gets delivered to B once its UA becomes reachable again. If the user does not stay connected, it has to wait until connectivity is regained and then send the voice mail using either option. RTP-based streaming has the advantage that this approach will work with arbitrary SIP servers supporting voice mail. However, the streaming may take more time (and is thus, again, potentially prone to further disconnections). And the UA may need to explicitly indicate its preference to talk to voice mail using SIP Caller Preferences [28] (so that the other user is not alerted again).

Given suitable UAs, the originally synchronous voice communication could be continued in an asynchronous fashion, similarly to a walkie-talkie or push-to-talk application, except with potentially longer delays between interactions—using voice messaging in analogy to instant text messaging [20]. User interface controls could allow users easily to transition back to synchronous communication if a need arises.

[8]If the dialog still exists at the remote peer, the *Replaces:* header can be used to indicate substitution of a previous dialog and thus avoid ringing the phone again. Preferably, however, a semantically distinct *Recovery:* header should be used which allows the receiver UA to handle the call accordingly.

4.2.3 Backwards Compatibility

If only one of the UAs supports the extensions described above, the automatic session re-establishment will still work (yet the other UA may "ring"), but media recording and replay will only work in one direction. It may be awkward for the user of the non-extended UA to answer the phone and immediately hear the continuation of a previous voice call. Hence, it seems sensible to have a capability indication exchanged during the INVITE transaction so that the automatically recovering UA can insert some delay (and maybe an announcement) before continuing playback. Such a capability is also useful to decide whether or not session re-establishment should be attempted at all and which node has to become active for SIP-based recovery.

For this purpose, the aforementioned *Recovery* header with two values can be included in every INVITE message and the corresponding 200 OK response. For any dialog setup initiated by a user, the first value is "false", for automated recovery, it is "true". The second value indicates a random 32-bit value as a tiebreaker. In case of simultaneous dialog re-establishment, the dialog from the entity with the larger value will remain active. In case a UA does not want to actively recover, it indicates this by using a value of zero in the original call.

5. EXPERIMENTAL VALIDATION

We have evaluated our design in two complementary ways: We have performed a first set of packet-based measurements passively observing (in UMTS) and actively causing (in WLAN) variable path performance including disconnections and analyzed the behavior of our algorithms based upon the resulting reception characteristics. And we have implemented a prototype integrating the DT-SIP functionality into a SIP UA.

5.1 Packet Trace Analysis

All the mechanisms introduced above rely on timeouts for detecting disconnections. To understand the effectiveness of our algorithms, we provide an analysis of the packet loss, delay, and packet inter-arrival characteristics for RTP traffic in two different scenarios. From the spacing between successively received packets, we can understand where our algorithms would perceive a connectivity loss (assuming certain delay parameters) and gain insights into proper (possibly adaptive) parameterization. For this purpose, we collected traces for UMTS data communication[9] to a mobile user on a train and WLAN traces to a pedestrian user in our lab.

We carried out six measurements using a UMTS data connection while aboard an Intercity train of Finnish Rail (VR) between Helsinki and Tampere in both directions. We used a server in our lab as the fixed endpoint and an IBM T41p laptop running Windows XP—connected to UMTS via Bluetooth and a Nokia E70 mobile phone—as the mobile node. A VPN tunnel was needed [10] and set up between the mobile node and the University of Bremen, so that we obtained a relatively long Internet path (about 30 hops).

The fixed node generated a steady stream RTP/RTCP packets using *rtpsend* [17]. The media stream consisted of 52 byte RTP

[9]Assuming that non-3GPP users would (be forced to) run VoIP applications using the data channel.

[10]Otherwise UDP traffic was apparently filtered and not a single RTP packet was received in either direction.

packets (12 bytes header + 40 bytes payload) sent every 20 ms resulting in an application data rate of some 20 kbit/s and 100 byte RTCP packets sent once per second resulting in a data rate 800 bit/s (modestly above the 2.5% fraction targeted for RTCP). The incoming packets were recorded at the mobile node using *rtpspy* [17]. Offline processing used scripting to produce the results (visualized using *gnuplot*). Based upon these measurements, we investigate the effectiveness of our RTP-based and RTCP-based detection mechanisms below.[11]

The measurements were mostly taken while the train was moving but also included stops in train stations along the way. Out of the six runs, only runs 1 and 2 lasted as long as expected and were manually terminated after some 15 minutes. For the other four runs, UMTS connectivity broke earlier, yielding trace durations between 60 s and 6 min. Durations and the RTP packet count are indicated in figure 9 below.

Figures 5 and 6 show the packet arrival characteristics for runs 1 and 2: the light shaded areas indicate the per-packet RTP delays, the crosses indicate RTCP delays, and the vertical bars the length of loss runs (at log scale) for RTP packets sent at a rate of 50 packets/s. Both delays are measured relative to the first received packet. For both runs, we observe quite some variation in delay (up to some 13 s) and repeated loss runs of several 100 packets (equivalent to several seconds). We also observe that losses equally affect RTP and RTCP packets (as expected if both share the same VPN tunnel).

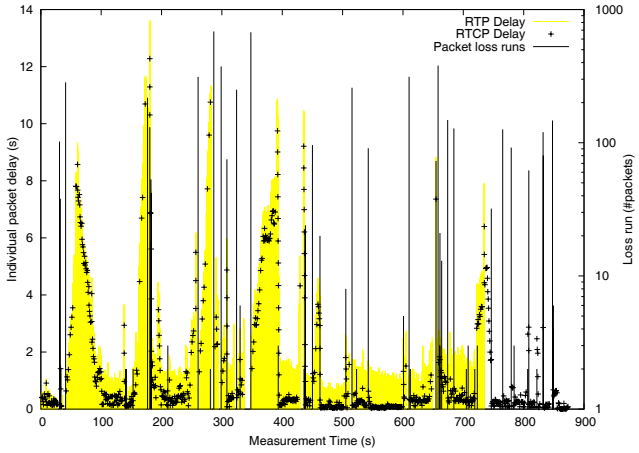

Figure 5: Packet arrival characteristics for UMTS run 1

For timeout-based detection, the inter-arrival delay of packets is the primary indicator when our detectors would kick in. Furthermore, the (increasing) per-packet delay may be an early indicator as may be the packet reception frequency. The latter two, however, depend on queuing delays along the path and may or may not be directly correlated with upcoming outages. Figures 7 and 8 show the per-packet delay for RTP (again relative to the first packet received) as the light-shaded area and the packet inter-arrival delays as crosses. Both figures show many gradual and steep delay increases due to queuing (which may or may not be caused by a worsening wireless link requiring more retransmissions) where the inter-arrival delays remain relatively low, packets keep flowing, and no connectivity loss occurs.

[11]Note that RTCP is representative also for detection based upon STUN or silence packets as the packet transmission frequency may be roughly similar.

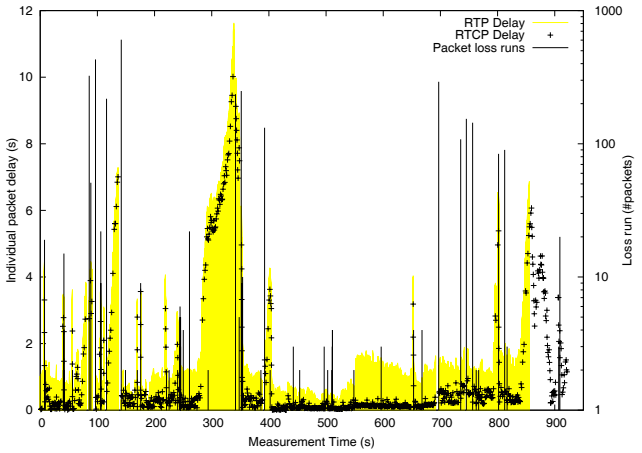

Figure 6: Packet arrival characteristics for UMTS run 2

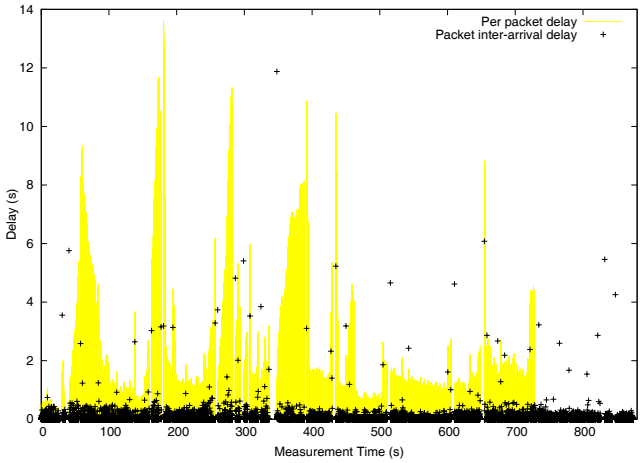

Figure 7: Per packet and inter-arrival delays for UMTS run 1

Assuming adaptive applications on the endpoints (and sufficiently patient users), detection mechanisms should hence rely on frequent packet samplings (as provided only in the RTP stream). If silence suppression is enabled or STUN or RTCP-based detection are in use, their timeouts should be chosen carefully (and may need to be dynamically adapted according to the observed jitter) in order to avoid false positives arising simply due to significant propagation delay variations. It seems preferable to have the user manually terminate a deteriorating call when she find it unbearable rather than invoking recovery procedures too early.[12]

Figures 9 and 10 show the complementary cumulative distribution functions for inter-packet arrival times for RTP (log-log scale) and RTCP packets (log scale).[13] For the RTP traces, we observe

[12]Experience from experiments with real users is required to determine what is acceptable in a real conversation. An analysis of skype conversations has found that the RTT has only little impact on the user satisfactions (hence, longer delay would be acceptable) where jitter and bit rate are dominant (hence, delay variations are not acceptable) [6].

[13]For the RTP packets in our six runs, the curve does not start at 1 as many of the packets arrive in "batches" with less than 10 ms

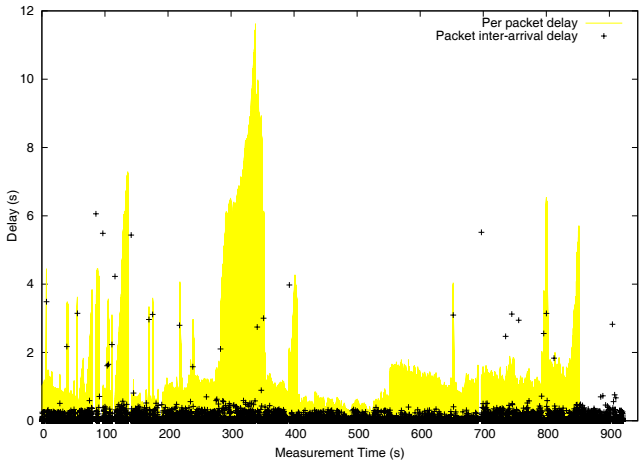

Figure 8: Per packet and inter-arrival delays for UMTS run 2

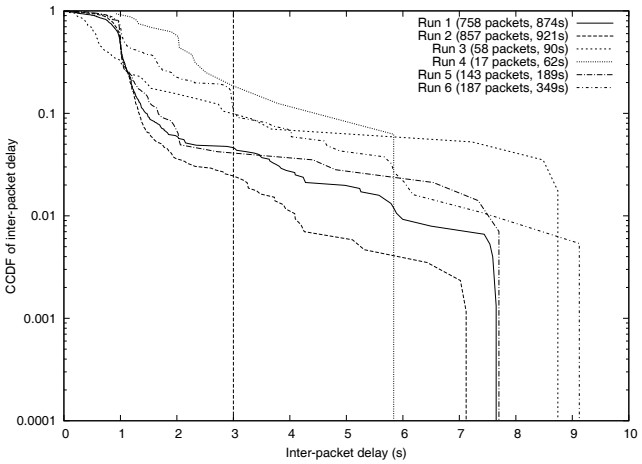

Figure 10: Inter-arrival times of RTCP packets

that 0.065–0.3% of adjacent packets are more than one second apart (dashed lines in the figure). Even for the lowest value (run 2), this yields a mean of one occurrence every 1,500 packets (=30 s) and hence a timeout of 1 s occurred 27 times in our longest trace; for run 1, we obtain about 0.13% and 47 timeouts. RTCP-based mechanisms using three successive missing packets as indicator (RTCP sent at 1 s intervals) result in 36 detections for run 1 and 21 detections in run 2. A slightly lower detection rate for RTCP is expected since occasional short-term outages are not covered by this mechanism.

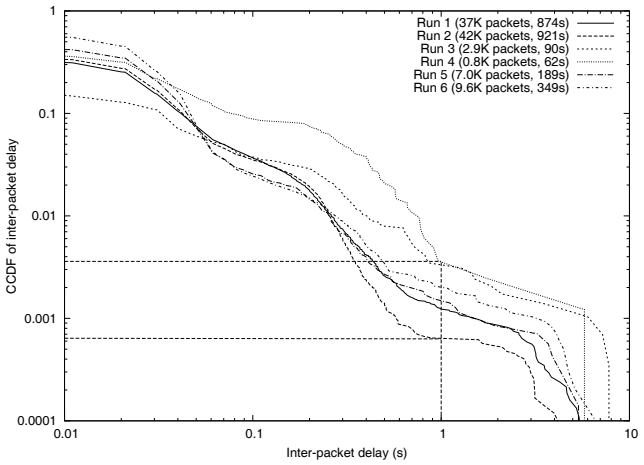

Figure 9: Inter-arrival times of RTP packets

To determine whether the two detection mechanisms operate in line with each other, we run a combined algorithm over all the traces and observe at which point in time the RTP- and RTCP-based detectors notice a disconnection. Table 1 summarizes the number of detections per method. We observed that in most cases, both detection mechanisms agree and that, naturally, RTP provides faster outage detection (which also explains the generally higher number of events). However, we observed one event in runs 4 and six events in run 6 in which three successive RTCP packets got lost, but RTP packets continued to flow—which emphasizes that the mechanisms should be applied together. Finally, not all of the disconnection events reported by the RTP detector were real disconnections, as can be seen from the lower RTCP reporting rate. This calls for a cautious choice of the timeouts. As noted earlier, which timeouts to apply before declaring a disconnection and take a repair action (and how to adapt these timeouts to the networking conditions) is subject to future study.

Test run	1	2	3	4	5	6
RTP-based detection	47	27	10	3	11	20
RTCP-based detection	36	21	6	3	6	20

Table 1: Number of detected connection loss events

We have also performed a first WLAN experiment in the lab to collect and analyze packet traces as above. We have set up a dedicated access point and used a fixed local node to generate an RTP packet stream to a laptop serving as a mobile node (using the same *rtpsend* and *rtpspy* setup as above, but 68.8 kbit/s as RTP data rate with 160 bytes audio payloads). Our experimental WLAN was unloaded but there was interference from production WLANs spread throughout the building. We walked through the corridors in our building starting close to the access point until the connectivity was lost and returned in order to re-gain connectivity. We repeated this experiment five times and applied our detection algorithms to the resulting RTP trace files. As expected, we find much less variation of delay and inter-arrival times in a WLAN, typically less than a hundred milliseconds and some 10–20 ms, respectively. These values increase significantly as we move closer to the edge of the coverage area of our WLAN, reaching up to 1 s and 2 s, respectively. Under these conditions, the RTP-based detection mechanism worked reliably and no false positives were observed.

5.2 Prototype Implementation

We have done a proof-of-concept implementation which is based upon the RTP/RTCP-based detection mechanism and supports all three recovery steps described above. We build on the SIP communicator open source implementation[14]. The SIP communicator

inter-arrival time despite their transmission spacing of 20 ms.

[14]https://sip-communicator.dev.java.net/

can be subdivided into three major modules: the SIP module, the media module and the GUI module. The SIP module provides all functions of SIP signaling: it processes all of the SIP requests and responses, and keeps the state of the session transactions.

The multimedia functions are implemented in the media module, currently restricted to voice. The voice data flow is depicted in figure 11. The entire process of handling voice data is implemented in this module, including recording voice data to and retrieving recordings from the local file system. The media module also manages RTP sessions. The network failure detection mechanism is implemented on top of the RTP session management. Usually, the SDP would be considered part the SIP signaling. In this implementation, however, SDP messages are parsed by the media module to simplify the interfaces between different modules. The SIP module pushes the entire SDP messages to the media module which has the best knowledge to understand the SDP message and format and act upon it. This also simplifies extending the detection to cover silence suppression and ICE in the future.

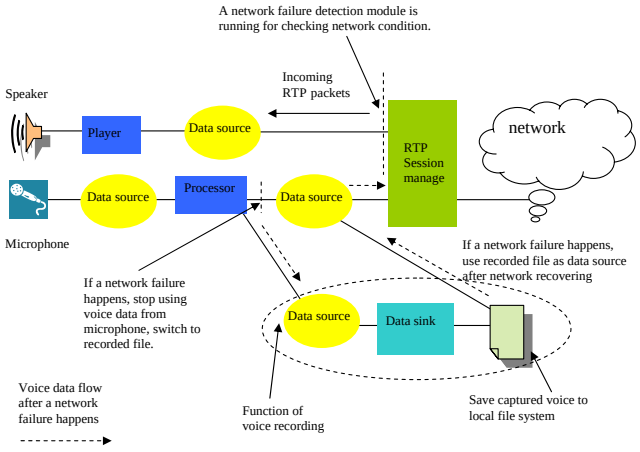

Figure 11: Implementation overview

The GUI module is the implementation of user interfaces. All of the GUI components are based on swing components. In addition, a sound alert component is also implemented in the GUI module. We have not yet implemented additional user interface elements provide enhanced control to the users to deal with disconnections and voice messaging.

To test the implementation of the disconnection-tolerant SIP (DT-SIP) client, we use a small SIP network comprising a SIP Express Router (SER) running on a Linux PC with an additional voice mail server module to store the voice mail from SIP clients. None of these components has been modified specifically for our experiments. Two DT-SIP clients are running on two independent machines as users A and B.

We have experimented with the three above disconnection scenarios. In the first scenario, client A calls clients B, and then we switch off the network interface of client A for 3 seconds in the middle of the call. After the network connection of client A is recovered, both clients A and B hear the delayed voice from each other. If both keep on talking to each other, the voice data continues to have a large delay which dependent to the connectivity loss duration—the implementation of silence compression is still ongoing. We observed that this recovery mechanism fails if either SIP client changed its IP address after the network recovery because the implementation assumes that the old RTP session will be continued. In case of such a failure, either or both nodes observe a prolonged outage period which will lead to a session re-establishment (second scenario) to rebuild the entire RTP session.

In the second scenario, we cause a longer path failure in the network and recover the connection after it. Both clients are trying to contact each other in randomized intervals after detecting the network break. In this scenario, either client A or client B will be called as soon as network is recovered. After the call is rebuilt, both clients are staring to replay the recorded voice. While the recovery works fine, we have yet to implement the user interface controls to allow users to switch off the auto-callback function or disable the playback after automated re-dial.

For the the third scenario we disconnect client A for a long period of time. Again, both clients try to contact each other but without success. In this case, client B reaches the voice mail server first because its connectivity remained. Client A connects to the voice mail server only after a long period without connectivity but can finally deliver the voice mail. While there is currently no interactive or automated retrieval of voice messages in the user interface, the basic voice mail retrieval functionality through an explicit call or a separate web interface proves the core functionality to work.

Finally, we have repeated the WLAN walking experiment in our lab described in the end of the previous subsection: we ran our prototype implementation on the mobile node and used Ethereal to monitor the wireless traffic to and from our mobile node. We moved out of WLAN coverage and returned repeatedly causing short and middle disconnection durations—as determined from the Ethereal traces. The voice conversation in the SIP dialog was recovered without additional SIP signaling for short durations of up to 15 s and by using automated SIP call re-establishment for longer disconnections.

6. CONCLUSION

In this paper, we have addressed the—largely neglected—case when seamless mobility with ongoing multimedia sessions fails and recovery is usually left to the users. We have proposed end-to-end disconnection detection mechanisms that do not make any assumptions about the location of failures and do not rely on link or network layer indications. We have proposed three recovery schemes to deal with outages of different durations, the use of which is basically customizable by the end user. All our mechanisms can be implemented using standard-compliant SIP and RTP. Yet, we propose one optional minor extension to SIP for dealing with non-DT-SIP peers and use standardized RTP and RTCP extensions to improve detection efficiency.

We have performed a first validation using RTP/RTCP measurements over UMTS and WLAN to assess the basic feasibility of the detection mechanisms. We have also provided a prototype implementation of the basic functionality, which clearly shows that the *technical* aspects of disconnection handling are well addressed. Our experiments have confirmed that further work is needed to properly address the *usability* aspects, ranging from adaptive timeout mechanisms to determining acceptable upper bounds for timeouts to silence compression to UI cues about disconnections to support for managing disconnected calls to an efficient integration of asynchronous and synchronous communication.

7. REFERENCES

[1] N. Banerjee, A. Acharya, and S. K. Das. Seamless SIP-Based Mobility for Multimedia Applications. *IEEE Network*, pages 6–13, March/April 2006.

[2] M. Bechler, W. J. Franz, and L. Wolf. Mobile Internet Access

in FleetNet. In *13. Fachtagung Kommunikation in verteilten Systemen, Leipzig, Germany*, April 2003.

[3] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM Computer Communication Review*, 27(5), 1997.

[4] S. L. Casner. Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth. RFC 3556, July 2003.

[5] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H.Weiss. Delay-Tolerant Network Architecture. RFC 4838, April 2007.

[6] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei. Quantifying skype user satisfaction. In *Proceedings of the ACM SIGCOMM*, 2006.

[7] S. Donovan and J. Rosenberg. Session Timers in the Session Initiation Protocol (SIP). RFC 4028, April 2005.

[8] A. Dutta, S. Madhani, W. Chen, O. Altintas, and H. Schulzrinne. Fast-handoff Schemes for Application Layer Mobility Management. In *Proceedings of IEEE Infocom*, May 2007.

[9] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of ACM SIGCOMM 2003*, pages 27–36, August 2003.

[10] T. Goff, J. Moronski, and D. Phatak. Freeze-TCP: A True End-to-end TCP Enhancement Mechanism for Mobile Environments. In *Proceedings of IEEE Infocom*, 2000.

[11] M. Handley, V. Jacobsen, and C. Perkins. Session Description Protocol. RFC 4566, July 2006.

[12] R. Koodli. Fast Handovers for Mobile IP. RFC 4068, July 2005.

[13] A. Lindgren, A. Almquist, and O. Schelén. Evaluation of Quality of Service Schemes for IEEE 802.11 Wireless LANs. In *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, 2001.

[14] K. E. Malki. (ed.) Low Latency Handoffs in Mobile IPv4. Internet Draft draft-ietf-mobileip-lowlatency-handoffs-v4-09.txt, Work in Progress, June 2004.

[15] Y. Mao, B. Knutsson, H. Lu, and J. Smith. DHARMA: Distributed Home Agent for Robust Mobile Access. In *Proceedings of the IEEE Infocom, Miami*, March 2005.

[16] R. Moskowitz, P. Nikander, P. Jokela, and T. R. Henderson. Host Identity Protocol. Internet Draft draft-ietf-hip-base-10.txt, Work in progress, October 2007.

[17] J. Ott and D. Kutscher. Drive-thru Internet: IEEE 802.11b for "Automobile" Users. In *Proceedings of IEEE Infocom, Hong Kong*, March 2004.

[18] J. Ott and D. Kutscher. Why Seamless? Towards Exploiting WLAN-based Intermittent Connectivity on the Road. In *Proceedings of the TERENA Networking Conference, TNC 2004, Rhodes*, June 2004.

[19] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *Proceedings of IEEE Infocom, Miami*, March 2005.

[20] J. Ott, D. Kutscher, and O. Bergmann. SIP Voice Services for Intermittently Connected Users in Wireless Networks. In *Upperside Conference Wi-Fi Voice 2004*, May 2004. available at http://www.informatik.uni-bremen.de/-jo/presentations/2004-upperside-wifi-voice-sip-intermittent.pdf.

[21] J. Ott, D. Kutscher, and M. Koch. Towards Automated Authentication for Mobile Users in WLAN Hot-Spots. In *Proceedings of VTC Fall 2005*, September 2005.

[22] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-based Feedback (RTP/AVPF). RFC 4585, July 2006.

[23] C. E. Perkins. (ed.) IP Mobility Support for IPv4. RFC 3344, 2002.

[24] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratty, and S. Banerjee. MAR: A Commuter Router Infrastructure for the Mobile Internet. In *Proceedings of the ACM Mobile Systems, Applications and Services Conference (ACM Mobisys 2004)*, June 2004.

[25] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. Internet Draft draft-ietf-mmusic-ice-19.txt, Work in progress, October 2007.

[26] J. Rosenberg, C. Huitema, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for (NAT) (STUN). Internet Draft, Work in Progress, October 2007.

[27] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[28] J. Rosenberg, H. Schulzrinne, and P. Kyzivat. Caller Preferences for the Session Initiation Protocol (SIP). RFC 3841, August 2004.

[29] H. Schulzrine, S. Casner, R. Frederick, and V. Jacobsen. *RTP: A Transport Protocol for Real-Time Applications*, July 2003. RFC 3550.

[30] H. Schulzrinne and E. Wedlund. Application-Layer Mobility Using SIP. *ACM Mobile Computing and Communications Review*, 4(3), July 2000.

[31] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[32] S. Shin and H. Schulzrinne. Multimedia: Balancing uplink and downlink delay of VoIP traffic in WLANs using Adaptive Priority Control (APC). In *Proceedings of the 3rd International Conference on Quality of service in Heterogeneous Wired/Wireless Networks (QShine)*, 2006.

[33] H. Soliman, C. Castelluccia, K. E. Malki, and L. Bellier. Hierarchical Mobile IPv6 Mobility Management (HMIPv6). RFC 4140, August 2005.

[34] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Highly Available Internet Services Using Connection Migration. Technical Report DCS-TR-264, Rutgers University, December 2001.

[35] Q. Wang and M. A. Abu-Rgheff. Mobility Management Architectures based on Joint Mobile ip and SIP Protocols. *IEEE Wireless Communications*, pages 68–76, December 2006.

[36] W. Wu, N. Banerjee, K. Basu, and S. K. Das. SIP-based Vertical Handoff between WWANs And WLANs. *IEEE Wireless Communications*, pages 66–72, June 2005.

[37] R. Zopf. Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN). RFC 3389, September 2002.