SIPPING
Internet-Draft
Expires: January 18, 2006

J. Rosenberg
Cisco Systems
H. Schulzrinne
Columbia University
July 17, 2005

Architecture and Design Principles of the Session Initiation Protocol (SIP)
draft-rosenberg-sipping-sip-arch-01

Status of this Memo

Copyright Notice

Abstract

The Session Initiation Protocol (SIP) and its many extensions and supporting technologies define a solution for multimedia communications on the Internet. Much of the design and architecture for SIP is based on a key set of architectural principles which, while commonly discussed on mailing lists and other forums, have not been explicitly captured. This document seeks to rectify that gap by

outlining the key set of architectural and design principles underlying SIP.

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) [1] and its many extensions and supporting technologies (for example, [2] [3] [4] [6]) define a solution for multimedia communications on the Internet. Much of the design and architecture for SIP is based on a key set of architectural principles which, while commonly discussed on mailing lists and other forums, have not been explicitly captured.

Section 2 of RFC 3261 briefly mentions a few of the design principles behind SIP. In particular, it mentions that SIP is not a vertically integrated system, but rather a component of an overall solution. It also mentions that SIP does not provide services, but rather, provides primitives which can be used to build up more complex services.

The guidelines for authors of SIP extensions [7] provides additional guidance in Section 3.2. In particular, it mentions session independence, signaling and media path decoupling, multi-provider and multi-hop as key design principles in SIP. It also touches on some proxy principles, such as method and body independence and transactional processing. It defines some of the characteristics of SIP methods - the full-state nature of INVITE, and the usage of of the request-URI as the key for forwarding. Finally, it mentions the importance of heterogeneity and generality over efficiency.

This document expands upon many of these principles, and mentions some of the other ones that are key to the SIP design philosophy.

2. Objectives

SIP's design is based around certain key objectives, including new features and services (and its corollary, that SIP is not a PSTN replacement), client heterogeneity, client multiplicity and multimedia.

2.1 New Features and Services

Perhaps the most important objective behind SIP is the desire to provide new communications features and services to users. SIP does more than just provide the ability to make voice calls between endpoints that look and feel like traditional telephones. It provides new features that take advantage of smart endpoints, multimedia and broadband connectivity.

One example is presence. SIP provides a generic event framework [5] on which presence is defined [8]. Presence allows SIP endpoints to obtain information on the ability, willingness and desire of another

user to communicate. In addition, SIP provides instant messaging capabilities, including a Short Message Service (SMS) style messaging [9] and a session mode [10].

Another example is the application interaction framework [11]. This framework allows for endpoints to interact with network based applications that utilize scripted user interfaces. One example if such a usage would be a web front end that allows a user to control a prepaid calling service.

Another example is the SIP caller preferences specification [12]. This specification allows a caller to direct call handling, and in particular cause the call to be routed to specific types of destination endpoints based on capabilities and characteristics.

2.2 Not a PSTN Replacement

A corollary to Section 2.1 is that SIP was not designed as a PSTN replacement. It provides many features and services the PSTN cannot provide, and it provides many services that the PSTN can provide, but in a much different manner. It has not been a goal of SIP to be able to directly map every message in ISUP or QSIG to SIP, or to be able to map each ISUP parameter into a SIP parameter. SIP was designed to facilitate communications in a way consistent with the architecture of the Internet, and the underlying network characteristics of the Internet result in a different technical solution.

2.3 Client Heterogeneity

The types of endpoints that connect to the Internet are extremely diverse, ranging from the latest gaming PCs to small appliances with barely enough memory and CPU for an IP stack. Similarly, devices vary widely in their capabilities to render media, interact with the user, and display information to the user. For this reason, SIP itself was designed to support heterogenous clients with highly variable capabilities.

SIP accomplishes this by providing an extension and capability negotiation framework covering many aspects of the protocol. Endpoints can negotiate support for new SIP option tags, new body types, new SIP methods, new codecs, and so on. That baseline framework can be leveraged for more complex situations. For example, the app interaction framework uses MIME type negotiation to indicate support for different types of scripts that control user interfaces. This, in turn, allows the framework to negotiate user interface capabilities with applications in the network.

In order to provide interoperability, SIP supports (and in many cases

mandates) fallback behaviors that allow differing endpoints to find a common ground.

2.4 Client Multiplicity

SIP was built under the assumption that users would have multiple clients at their disposal - softphones, hardphones, cell phones, PDAs, and so on, and that these devices could be in use simultaneously. Users can make multiple calls at the same time, each from a different user agent. Similarly, users can receive calls that "ring" each device at the same time or in sequence. Forking, which allows multiple devices to be rung at once, is a key part of the baseline SIP specification.

2.5 Multimedia

Although much actual usage of SIP is in support of voice communications, SIP was designed to be media agnostic, and thus facilitate the deployment of multimedia. Audio, video, text and messaging are all possible with SIP. Nothing in SIP itself depends on or assumes a particular media stream type.

3. Architectural Principles

This section discusses the key SIP architectural principles - the usage of proxies for routing, the relegation of call state to endpoints, the usage of dialog models and not call models, endpoint fate sharing, component based design, logical roles, Internet-based design, generality over efficiency, and separation of signaling and media.

3.1 Proxies are for Routing

When designing a distributed system, one of the primary decisions is to determine what functionality will be assigned to which components of the system. SIP is a distributed system, composed of user agents and proxies. Proxies usually run "in the network" and their role is to faciliate rendezvous between users. The assumption in SIP is that users can be reachable at many differing devices, each of which provides a different set of features and capabilities. The problem, then, is to determine how to route a SIP message from a caller to a recipient, taking into account the desires of the caller, the recipient, and the policies of the providers in between. Each proxy on the path of a SIP request is responsible for executing the routing logic based on the desires of the entities on whose behalf it is acting. The final call routing decision is a composition of the decisions made by each of the proxies along the request path.

Because the role of the proxy is to connect users together in order to communicate, once they are connected, the proxy's task is usually done. SIP assumes that the IP network provides connectivity between the endpoints, and therefore, any additional signaling that takes place will frequently go end-to-end. This is not always the case; NATs and firewalls break the end-to-end connectivity model, and thus proxies frequently remain in the signaling path to facilitate the exchange of SIP messages.

It is important to understand that a proxy is not a switch in the traditional telephony sense. It does not maintain call state, and thus many of the capabilities switches provide that derive from management of call state are not provided by proxies. Switches do provide call routing functions, and that aspect of switch behavior is also provided by proxies. Furthermore, switch features traditionally associated with routing are also commonly placed in proxies.

There were many factors driving the decision for proxies to take on the role of rendezvous, as opposed to endpoint control and management. Some of them include:

Availability: Because proxies do not need to maintain call state, they can fail in the middle of a call without impacting calls in progress. This makes SIP systems highly available at very low cost.

Scalability: Proxies can be deployed in clusters, where any one of the proxies in the cluster can serve a request. No state sharing is needed across elements in the cluster, and proxies can be added or removed from a cluster as capacity needs require. This results in perfect linear scalability.

Flexibility: By relagating call state and many features to endpoints, they can manifest themselves in ways that are appropriate for the capabilities of user interfaces of the endpoint in question. This gives SIP the flexibility to truly deal with diverse endpoints.

3.2 Endpoint Call State and Features

Since the role of the proxy is to facilitate rendezvous, the rest of the capabilities needed in communications systems fall to the endpoints. In particular, SIP endpoints are responsible for maintaining call state, and for implementing features that require awareness and management of that call state.

As an example, consider call waiting. In SIP, call waiting functionality exists in the endpoints. This is because it is

relatively easy for endpoints to receive a new call while one is in progress, alert the user, and then select which media stream to render based on that selection. Indeed, the way in which the user is "alerted" and the mechanism by which they select the stream to render may vary widely depending on the type of endpoint. A dumb phone may do what is done today in the PSTN - provide an audio cue to alert the user, and then use the hookflash to switch between calls. However, on a PC-based softphone, a window pop can be used to alert the user to an incoming call, and then the user can use a mouse to select the call (perhaps represented as an object in a window) that they wish to hear. The interface may be different once again for a television set top box, which may render information about an incoming call on the TV screen, and then use a button on the remote to indicate that the call should be taken.

In order to allow each of these endpoints to handle call waiting in the way that is appropriate for that endpoint, the feature intelligence needs to reside in the endpoint itself. In a centralized switch type of architecture, such as those provided by the Media Gateway Control Protocol (MGCP) [13] and Megaco [14], the endpoint is a slave to the controller, and the feature intelligence resides in the controller based on an assumed model of the user interface and capabilities of the devices. Those architectures fundamentally limit endpoint innovation because they provide no ability for endpoints to differentiate themselves by providing better features or enhanced user experiences; all of that resides in the controller. In order for SIP to provide heterogeneous clients and benefit from endpoint capabilities and innovation, it makes the opposite design choice on purpose.

This aspect of SIP's design is often summarized as "smart endpoints".

3.3 Dialog Models, not Call Models

SIP does not define a "call model" in the traditional PSTN sense. SIP does define a dialog model [15]. This model defines the state associated with a communications context between a pair of endpoints. However, there is a fundamental difference between a call model and the dialog model. A call model encompasses nearly all of the fullness of the application state machine that is executing based on the underlying protocols. In SIP, this is not so. SIP presumes that it is being used by some higher layer application that is making sense of the various dialogs and SIP messages that are being exchanged. The state machine governing the operation of that application is not standardized by SIP, and purposefully so. By allowing it to be endpoint and application specific, SIP allows for numerous applications and usages not originally envisioned in the original design of the protocol.

This particular facet of SIP's design has allowed it to be applied to problems as diverse as Push-to-Talk and home automation. Though sometimes these usages are not really a good fit for SIP, they are demonstrable validations of this design goal.

A direct consequence of this design approach is that SIP doesn't generally standardize features. In many cases, features can execute within an endpoint without any involvement or awareness from other endpoints. Call waiting is a good example of that. In other cases, a feature requires some kind of communications with other elements in the network. In those cases, SIP prefers to specify a generic primitive that can support many features.

3.4 Endpoint Fate Sharing

A benefit of the smart endpoint model described in Section 3.2 is that call state and application state is co-located with the endpoints of the call itself. This means that the only way in which the call or application can fail is if the endpoints themselves fail. Of course, if the endpoints fail, the call is over in any case. This allows for fate sharing, and it allows SIP systems to be highly available.

3.5 Component Based Design

Section 3.3 alludes to another important principle behind SIP design - the use of protocol components and primitives. Generally speaking, SIP does not specify a vertical communications system. Rather, SIP itself is just a component in any complete communications system. SIP is designed to work hand-in-hand with other components, such as session descriptions like the Session Description Protocol (SDP) [6], media transport like the Real Time Transport Protocol (RTP) [17] and signaling compression like SigComp [18].

SIP itself provides capabilities through component functions that can be composed together to provide more complex functions. As an example, most of the call control capabilities SIP enables are done through a combination of two classes of components - notification and manipulation. SIP provides a generic event framework [5] that allows a user agent to learn about state elsewhere in the network. That framework is used to support notifications about registration state [19], dialog state [15], conference state [16], messaging state [20], presence state [8] and subscription state [21]. The content of the notifications across these packages allow an endpoint to gain awareness about the state of much of a SIP system (subject to authorization, of course).

With awareness about the state of parts of the SIP system, several

SIP components allow an endpoint to manipulate that state. The REFER method [22] allows endpoints to ask other endpoints to generate SIP requests. The Join [24] and Replaces [23] header fields allow an endpoint to merge and split dialogs.

As an example of building more complex features from these primitives, consider single line extension, as described in [25]. This feature is possible by having each line in the group use the dialog event package to learn about calls made to and from the other lines (the notification part), and then the Join mechanism for adding a line to an existing call (the manipulation part).

3.6 Logical Components, not Physical

SIP defines its functionality by defining the exchange of messages between logical components in a distributed system. These components - user agents, proxies, redirect servers and registrars, are only logical, not physical. SIP does not dictate that these components be deployed as distinct servers. The expectation is that actual products will combine many logical functions into a single "box" as defined by business needs.

Indeed, many SIP specifications define logical functions that are purely logical; in other words, they must be co-located with some other logical component, usually a proxy or a user agent. The privacy service [26] and the authentication service [27] are two examples of this.

3.7 Designed for the Internet

SIP was designed for operation on the public Internet. That is not to say that it can't also be used on private IP networks; it can. However, the public Internet introduces a number of constraints and also a number of benefits that have been taken into account in SIPs design.

As an example on the constraints, SIP signaling needs to be congestion controlled in order to run cooperatively on the Internet. The public Internet also means that SIP cannot assume a particular IP network topology, and needs to work in the face of packet loss and delays.

As an example of the benefits, SIP can make use of many of the core Internet services. SIP uses the DNS for discovery of SIP servers [3], load balancing and reliability, DHCP for client configuration [28] [29], and a certificate infrastructure for SIP over TLS [1]. SIP does not require any of these to operate, but it leverages them when SIP runs on an IP network where they are available.

3.8 Generality over Efficiency

When designing network protocols, there is often a tradeoff between efficiency (measured in terms of bandwidth, memory or processing) and generality. SIP was designed under the assumption that continuous improvements in CPU, memory and bandwidth availability, fueled by Moore's law and its related principles, would make efficiency a transient benefit, while generality is a permanent one. As a result, SIP generally prefers to build for generality at the expense of efficiency. This is not an absolute truth, but it has served as a general guideline.

As a specific example, SIP has not tried to be parsimonious in its usage of bits in message fields. Rather, in environments where message overhead is an issue (such as wireless systems), message compression is handled in a shim layer using Sigcomp so that it does not need to pervade every extension that gets defined.

3.9 Separation of Signaling and Media

It is fundamental to the design of SIP that the path followed by the media packets is independent of the path followed by the signaling packets. This separation allows for the IP network to deliver the media packets using the most direct and appropriate route it can, while the signaling packets, which are not as latency sensitive, can follow a series of proxy elements needed for the processing of the request. By separating the two, more complex proxy topologies can be utilized without concern for the impact on voice quality.

4. Design Principles

This section discusses some of the design principles used in SIP's design. They are not as fundamental as the architectural principles, but represent key design choices that were made.

4.1 Proxies are Method, Body and Header Independent

Since the role of the proxy is to provide rendezvous, the primary information from the message used by the proxy is the request URI, Via, Route and Record-Route header fields. Extensions can also define new header fields that are relevant for proxy processing, such as the Accept-Contact and Request-Disposition header fields [12].

However, except for the notable exceptions of ACK and CANCEL, proxy routing of a request does not normally depend on the request method. This allows SIP's rendezvous functions to be provided to other functions besides session initiation. The same is true for SIP bodies, which are of interest end-to-end and not used by proxies.

That said, the specifications do not prohibit proxies from invoking special routing logic based on method or other information in the message. Such behaviors are typical of distributed proxy networks where the overall processing of a SIP user agent is distributed across multiple components, and a proxy is used to route messages to the appropriate component.

4.2 Full State Nature of INVITE

One of the important design choices made by SIP is that each INVITE request within a dialog conveys the full state of the session. For example, instead of a re-INVITE indicating that a video session should be added, a re-INVITE would state that the session is being updated and now includes audio and video. This characteristic of the INVITE method is useful for systems that perform inspection of the INVITE message in order to manipulate some underlying network state. An example of this is the MIDCOM framework [30].

4.3 SIP URIs Identify Resources

SIP URIs are an important part of SIPs design. The SIP URI is an identifier for a communications resource, and that resource can be a user, a device, a service or some combination thereof. Traditional SIP addresses-of-records (AOR) identify users, but URIs have been defined that identify user agent instances [31], and voicemail services [32], for example.

Generally speaking, it is not (and should not) be possible to inspect a URI and conclude whether or not the URI identifies a user, device, or a specific type of service. Only the owner of the domain on the right hand side of the @ sign can interpret or define the meaning of the resource identified on the left hand side. The owner of a domain must be able to, at will, change the nature of the resource identified by any specific token on the left hand side of the @ sign.

It is quite appropriate for a SIP URI to identify a fairly complex resource, and to use the URI to parameterize the service that gets invoked [33]. Ideally, a SIP URI is handed out and discovered through other means, such as presence or on a business card. However, it is not desirable for a SIP URI to be constructed based on pre-determined awareness of the type of resources provided by a domain.

4.4 Extensibility and Compatibility

Extensibility has been a key design goal for SIP. SIP assumed that many usages would arise which could not be forseen at the time SIP was specified. SIP needed to be able to support those future needs,

yet still be interoperable with older implementations. It is for this reason that SIP provides a fairly complex extensibility framework. New methods can be defined, new header fields, new body types, and new parameters. Several header fields, including the Accept, Accept-Language, Allow, Supported, Require, Proxy-Require, Accept-Encoding and Unsupported header fields, have been defined to facilitate negotiation of support for extensions.

SIP provides two general modes for usage of extensions - "asking permission" and "asking forgiveness". In the former modality, an endpoint first discovers the capabilities of a peer, either through an OPTIONS message or through capability header fields exchanged during dialog establishment. Once they are discovered, those extensions can be safely used. In the latter modality, an initial request makes use of an extension, and then through either the Require or Proxy-Require header field, tells the peer to reject the request if the extension is not supported.

SIP also differentiates between extensions that are specific to user agents, and those that are specific to proxies. This is due to the differing roles of proxies and user agents in the SIP network.

In all cases, interoperability is only achieved by being able to fall back to or support baseline behavior. This is discussed extensively in [7]. An implementation that uses an extension but does not provide fallback is not compliant to the SIP specifications, and should be considered no better than proprietary.

4.5 Internationalization

SIP is meant to be used in an international setting. It supports UTF-8 encoding of freeform text and declaration and negotiation of languages.

4.6 Explicit Intermediaries

Proxies represent a form of intermediary that operates on requests on behalf of users. However, unlike other intermediaries like NATs and firewalls, which are invisible to participants in the protocol, proxies are visible. They declare themselves through Via and Record-Route header fields, their roles are well defined and bounded, and they are meant to act in concert with user agents. A proxy is involved in request processing only by the explicit request of a user agent or another proxy. An element which intercepts a SIP message not addressed to can not ever be considered a compliant proxy.

Furthermore, when proxies need to provide additional functions on behalf of user agents, this is always best done by explicit

communications with the endpoints, rather than implicit behaviors. Explicit cooperation with endpoints guarantess that SIP can continue to provide the end-to-end security features that are important to its design. As an example of this, if a proxy needs to restrict the set of audio codecs used by a user agent, it is far better to use the session policy mechanisms [34] to ask the client to discard the codec, than for the proxy to attempt to modify the SDP in an INVITE message as it goes by.

4.7 Guided Proxy Routing

Many SIP capabilities are provided by having an entity, either a user agent or a proxy server, predetermine a set of downstream proxy resource that must be visited prior to completion of the request. This is known as loose routing, and is a key part of SIPs design. The main task in loose routing is the discovery of the URI to use. SIP provides several techniques for that, including the Record-Route mechanism in RFC 3261, the Path header field [35], and the Service-Route header field [36].

4.8 Transport Protocol Independence

Another design choice made by SIP is its ability to run over several different transport protocols. These include, at the moment, UDP, TCP and SCTP. The transport protocol must be capable of providing just a minimal set of capabilities - the transport of 8-bit messages of at least around 1500 bytes. [[EDITORS NOTE: In hindsight its not clear that this flexibility has been worth the cost of complexity. Mention this?]]

4.9 Protocol Reuse

SIP has attempted to reuse many other protocol components as part of its design. SIP uses MIME [37] for transport and encoding of body parts, reuses many of the HTTP [38] header fields and semantics, makes use of the URI framework [39], including non-SIP URIs like the tel URI [40], uses standardized transport protocols like SCTP [41] and existing security protocols, like TLS [42] and SMIME [43].

This reuse flattens the learning curve for SIP and eases implementation by allowing developers to use off-the-shelf implementations of its component parts.

5. Security Considerations

This specification does not introduce any new security considerations for the Internet. However, SIP does introduce new considerations, and those are discussed in the SIP specification and its extensions.

6. IANA Considerations

This specification does not introduce any IANA considerations.

7. Acknowledgements

The authors would like to thank Cary Fitzgerald for his "Tao of SIP" list.

8. Informative References

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

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

[3] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[4] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[5] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

[6] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[7] Rosenberg, J., "Guidelines for Authors of Extensions to the Session Initiation Protocol (SIP)", draft-ietf-sip-guidelines-09 (work in progress), February 2005.

[8] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", RFC 3856, August 2004.

[9] Campbell, B., Rosenberg, J., Schulzrinne, H., Huitema, C., and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", RFC 3428, December 2002.

[10] Campbell, B., "The Message Session Relay Protocol", draft-ietf-simple-message-sessions-10 (work in progress), February 2005.

[11] Rosenberg, J., "A Framework for Application Interaction in the Session Initiation Protocol (SIP)",

draft-ietf-sipping-app-interaction-framework-04 (work in progress), February 2005.

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

[13] Andreasen, F. and B. Foster, "Media Gateway Control Protocol (MGCP) Version 1.0", RFC 3435, January 2003.

[14] Groves, C., Pantaleo, M., Anderson, T., and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[15] Rosenberg, J., "An INVITE Inititiated Dialog Event Package for the Session Initiation Protocol (SIP)", draft-ietf-sipping-dialog-package-06 (work in progress), April 2005.

[16] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Conference State", draft-ietf-sipping-conference-package-12 (work in progress), July 2005.

[17] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[18] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, "Signaling Compression (SigComp)", RFC 3320, January 2003.

[19] Rosenberg, J., "A Session Initiation Protocol (SIP) Event Package for Registrations", RFC 3680, March 2004.

[20] Mahy, R., "A Message Summary and Message Waiting Indication Event Package for the Session Initiation Protocol (SIP)", RFC 3842, August 2004.

[21] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", RFC 3857, August 2004.

[22] Sparks, R., "The Session Initiation Protocol (SIP) Refer Method", RFC 3515, April 2003.

[23] Mahy, R., Biggs, B., and R. Dean, "The Session Initiation Protocol (SIP) "Replaces" Header", RFC 3891, September 2004.

[24] Mahy, R. and D. Petrie, "The Session Initiation Protocol (SIP) "Join" Header", RFC 3911, October 2004.

[25] Johnston, A. and R. Sparks, "Session Initiation Protocol Service Examples", draft-ietf-sipping-service-examples-08 (work in progress), February 2005.

[26] Jennings, C., Peterson, J., and M. Watson, "Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks", RFC 3325, November 2002.

[27] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", draft-ietf-sip-identity-05 (work in progress), May 2005.

[28] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, August 2002.

[29] Schulzrinne, H. and B. Volz, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", RFC 3319, July 2003.

[30] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[31] Rosenberg, J., "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", draft-ietf-sip-gruu-03 (work in progress), February 2005.

[32] Jennings, C., "SIP Conventions for Voicemail URIs", draft-jennings-sip-voicemail-uri-03 (work in progress), October 2004.

[33] Campbell, B. and R. Sparks, "Control of Service Context using SIP Request-URI", RFC 3087, April 2001.

[34] Hilt, V., "Session Initiation Protocol (SIP) Session Policies - Document Format and Session-Independent Delivery Mechanism", draft-ietf-sipping-session-indep-policy-02 (work in progress), February 2005.

[35] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[36] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration", RFC 3608, October 2003.

[37] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[38] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[39] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[40] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

[41] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[42] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[43] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

Authors' Addresses

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027
US

Email: schulzrinne@cs.columbia.edu
URI: http://www.cs.columbia.edu/~hgs

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.