

# Real-time Monitoring of SIP Infrastructure Using Message Classification

Arup Acharya Xiping Wang Charles Wright  
IBM TJ Watson Research Center, Hawthorne, NY  
{arup, xiping, cpwright}@us.ibm.com

Nilanjan Banerjee Bikram Sengupta  
IBM India Research Lab, New Delhi, India  
{nilanjba, bsengupt}@in.ibm.com

## ABSTRACT

Session Initiation Protocol (SIP) is a control-plane protocol for multiple services such as VoIP, Instant Messaging and Presence, and in addition, is key to IP Multimedia Subsystem (IMS). A SIP message consists of plain-text headers and their corresponding values, which are used to route the message between one or more endpoints, resulting in a media session. These headers and values are often transformed/re-written at intermediate SIP servers ("proxies"). It is important to monitor the flow and transformation of such messages in real-time, for functional testing of a SIP overlay network containing malfunctioning or ill-configured SIP entities, or for efficient run-time SIP network operation, including problem determination and load balancing. Towards that end, we have designed and implemented a programmable in-kernel Linux SIP message classification engine. The classifier can be configured to intercept incoming and outgoing SIP messages from a server, extract appropriate message meta-data including distinguishing header-value pairs and their transformations, and forward the same to a monitoring engine. The engine collates this information from different classifiers across the network, to infer the state of a SIP call on individual servers on the call path as well as aggregated call-state.

## Categories and Subject Descriptors

C.2.3 [Network Operations]: Network Monitoring

D.4.4[Communications Management] : Network Communications

## General Terms

Management, Measurement, Performance, Design, Experimentation, Verification.

## Keywords

Session Initiation Protocol, VoIP, Classification, Monitoring.

## 1. INTRODUCTION

Session Initiation Protocol (SIP) is a control-plane protocol for

establishing, manipulating, and terminating multimedia sessions with one or more participants. SIP is media agnostic and can establish voice, text, video, and other types of sessions. SIP's basic functionality for session setup has been extended to include instant messaging and presence [4]. It has gained widespread acceptance and deployment for VoIP amongst wireline providers (e.g., Vonage), web-based providers (e.g. Yahoo, MSN, AOL) and within the enterprise. Additionally, collaboration software vendors are starting to support SIP for instant messaging, presence and collaboration. Cable companies and cellular providers are looking to embrace IMS as an underlying architecture for introducing such services [7]. SIP based network infrastructure forms the foundation of delivering these services, and these applications must maintain highest availability and superior scalability to match the voice service expectations and demands. Such QoS guarantees can be provided by deploying management technologies that can measure service quality and identify the problems that are affecting the user experiences.

Of the various management activities that are needed for efficient administration of a SIP network, in this paper, we will primarily focus on the following: (i) functional testing of SIP overlay network by monitoring the call-path of synthetic SIP dialogs or sessions; and (ii) real-time end-to-end monitoring of SIP dialogs or sessions and collection of aggregate dialog or session information for network management. The first one helps resolving issues with malfunctioning or ill-configured SIP entities (servers, proxies, and user agents), while the second helps in the deployment of appropriate network management techniques such as load balancing, exception handling etc., to ensure efficient network operation.

For both these activities, it is important to monitor the flow and transformation of SIP messages as they traverse the network. However, there are several challenges in doing this. SIP is a protocol that is both similar and different from HTTP and the web-model. A SIP message consists of plain-text headers and their corresponding values, which are used to route the message between one or more endpoints: the messages are transformed at each hop, resulting in headers being added, deleted, or modified. Additionally, a SIP message may *fork* into multiple distinct messages, which could result in parallel or sequential execution of the control flow along multiple paths. While for many of the cases, messages corresponding to a single dialog can be identified using the Call-ID in the message header, there are quite a few other cases where the Call-ID changes within a dialog and this transformation needs to be captured for effective monitoring of single dialog or session. When setting up conference calls, as yet

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MineNet'07, June 12, 2007, San Diego, California, USA  
Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

another example, all endpoints of conference call may not share a common call-ID (but share some other subset of header values that identify them as part of the same call). Also, when an intermediate SIP server (SIP proxy) processes and forwards a SIP message, it may lead to a change of state at the SIP proxy: SIP servers maintain multiple levels of call-state, ranging from transaction-state to dialog-state. In order to get an overall perspective of a SIP system, it is thus important to understand the state of a call at each of the servers in its path and combine the different states into a globally meaningful call state. When SIP is used for Presence, there is no call-state per se, but there is rather a chain of publish/subscribe events. In such situations too, it may be useful to monitor the flow of messages and deduce the causality of information.

To track the flow and transformation of SIP messages, we employ a software engine called SIP message classification engine or classifier that is incorporated within the operating system (OS) of a server machine. The actual SIP software (e.g., SIP proxy) runs as an application on this server, for example, routing SIP messages. Placing our classification engine within the OS is desirable for reasons of efficiency as well as architecturally, depending on the specific scenario. The defining feature of this classifier is that it is programmable, and therefore, the same software can be used for various application scenarios, by using an appropriate set of rules. We design an efficient algorithm that takes as input a set of user-defined rules, and morphs them into suitable data structures that enable fast matching of rules against the input message stream. The rules specify both how to identify specific subsets of messages, based on a combination of message header values including complex functions such as set membership as well operations on the state amassed at the classifier from previous messages, and the actions to be executed on the matching packets. For the purposes of monitoring, one such action is to parse the incoming and outgoing message streams, belonging to a particular call, from the server and generate *footprints*. The footprints contain selected message meta-data including distinguishing headers-value pairs and their transformations, and are forwarded by the classifier to a central monitoring engine. The engine collates this information from different servers/classifiers across the network, to infer the state of a SIP session on individual servers on the call path as well as aggregated call-state.

The rest of the paper is organized as follows: In Section 2, we provide a brief background of SIP. In Section 3, we describe the design, implementation, and initial performance results of our classification engine. In Section 4, we present some challenges associated with monitoring a SIP system and outline how our classification engine may be used to generate footprints for tracking SIP dialogs/sessions. In Section 5, we describe related work. Finally, we draw conclusions in Section 6.

## 2. SIP BACKGROUD

As shown in Fig1, a SIP infrastructure consists of *user agents* and a number of *SIP servers*, such as registration servers, location servers and SIP proxies deployed across a network. A *user agent* (UA) is a SIP endpoint that controls session setup and media transfer. RFC3261 [6] describes the SIP protocol in detail.

All SIP messages are requests or responses. For example, INVITE is a request while “180 Ringing” or “200 OK” are responses. A

SIP message consists of a set of headers and values, all specified as strings, with a syntax similar to HTTP but much richer in variety, usage and semantics. For example, a header may occur multiple times, have list of strings as its value, and a number of sub-headers, called parameters each with an associated value. In the following example from [6] Alice invites Bob to begin a dialog:

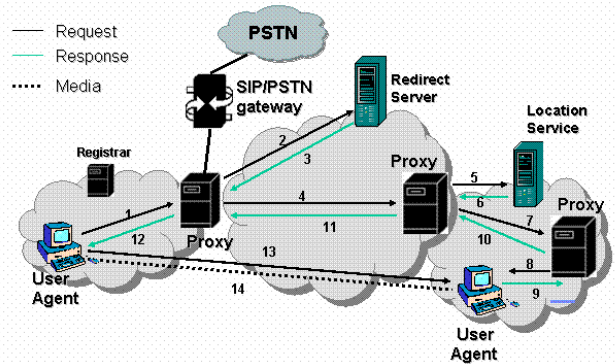


Figure 1. SIP architecture.

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9h
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=192
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
(Alice's Session Description not shown)
```

SIP messages are routed through SIP proxies to setup sessions between user agents. All requests (such as an INVITE) are routed by the proxy to the appropriate destination user agent based on the destination SIP URI included in the message. A session is commonly setup between two user agents through an INVITE request, an OK response and an ACK to the response. This is shown in Figure 2 where the call setup is followed by media exchange using RTP. The session is torn down through an exchange of BYE and OK messages.

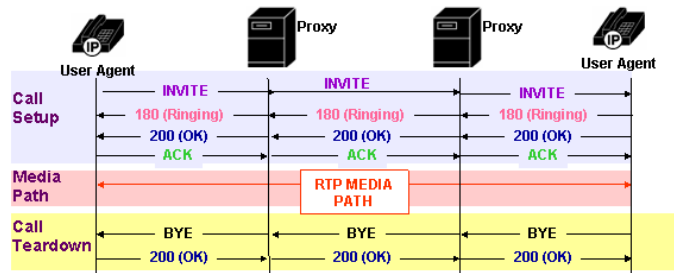


Figure 2. SIP call setup and media path.

There are many possible variations to the signaling sequence shown above, such as multiple INVITE/OK/ACK exchanges

during a call-lifetime, thus modifying the call (e.g., call handoff in cellular environments, or the addition of a video session), forking of the INVITE message to multiple targets, looping back on the request path and/or redirection. In addition, conference calls are common, where multiple endpoints signal to common control server (conference server).

To set the appropriate perspective with respect to SIP network management functionality described subsequently, let us review some of the relevant definitions [6]. There are three different types of associations that happen between the UAs (user agents) in a SIP network. They are as follows. *SIP Transaction*: A SIP transaction occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to a final (non-1xx) response sent from the server to the client. *Session*: A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session. *Dialog*: A dialog is a peer-to-peer SIP relationship between two UAs that persists for some time. A dialog is established by SIP messages, such as a 2xx response to an INVITE request. A dialog is identified by a call identifier, local tag, and a remote tag. A dialog was formerly known as a call leg.

### 3. SIP MESSAGE CLASSIFICATION ENGINE

In this section, we briefly review the key features of our SIP Message Classification engine. Our engine is flexible in that it matches SIP messages based on arbitrary headers using a programmable set of rules. It is efficient, because the static phase of our algorithm creates a set of tables that enable the runtime portion to extract only the subset of headers required, scans the message only once to do so, and rapidly compares the message to a set of rules, using bitwise operations. This is essential for scalability as SIP messages can contain dozens of headers, multiple instances of a single header, only a small subset of which may be necessary for classification. Furthermore, allowing bitwise comparisons allows for a larger number of machine generated rules.

We implemented our classification engine a Linux kernel module. Providing the classification engine as a kernel module is advantageous because some applications (e.g., overload protection) are quite performance sensitive, and classifying messages in the kernel eliminates unnecessary data copies and context switches. Moreover, kernel modules are often more readily deployable than application-level updates, as it can be changed independently of the rest of the software stack (e.g., a kernel-level modification can work with any SIP server). Details of the design/ implementation and performance are available in [18].

As the classification engine can be deployed in multiple contexts, the input is a set of rules, expressed as a conjunction of conditions followed by an action. We expect that when the classifier is embedded in other solutions (e.g., the transaction monitoring system described in this paper) that it will ship with pre-defined rules, thus obviating the need for administrators to configure rules. The conditions operate on headers, which could be one of predefined SIP headers, pseudo-headers (which we define for convenience, eg message\_type), and derived headers (composed

from other headers). For example, From.tag represents the “tag” parameter within the From header and is derived from the From header. The Dialog-ID derived header is composed of Call-Id (a SIP header), From.tag, and To.tag.

We support user-defined data types (scalars, pointers, and associative arrays) composed of basic data types (e.g., String or Integer) or other user-defined data types; we are able to maintain user-specified state that can be updated as part of rule actions. For example, in

```
Struct Session = {Dialog Dialog1, String State}
```

the element “State” stores the state of a dialog which could be “established”, “setup”, or “shutdown”. Condition evaluations could have side-effects, e.g. if the Dialog-id of the current SIP message is an element in a list of active sessions, then the following condition evaluation results in storing a pointer to the matching element (which for example, could then be used to modify any associated data structures as part of a rule action).

```
*CurrentSession=(Dialog-ID belongs-to %ActiveSessions)
```

Rules consists of a conjunction of conditions and an associated list of actions, which could further change variable values. Depending on the context in which the classifier is being used, actions could further result in placing a message in a queue, e.g. drop queue if the context is overload control.

Our classification algorithm has a static and a runtime component. The static component consists of rule parsing and creating several tables and bitmaps that allow the runtime portion to operate efficiently. The key to generating these efficient structures is that most redundancy is eliminated from the rule set, so that the runtime phase does not need to repetitively extract headers or evaluate conditions. The runtime component consists of parsing individual SIP messages, comparing them to the rules, and performing a set of actions when a message matches a rule.

The SIP classifier has been tested for its efficiency in handling other situations like overload control. Details of the performance results can be obtained in [18]. However, to give a sense of the performance we briefly present some of the results here. When faced with loads beyond their capacity (e.g., during catastrophic situations and major network outages), SIP servers must drop messages from the input stream. It is therefore desirable that the server process high-value messages in preference to dropping lower-value messages. We evaluated our in-kernel classifier implementation with a commonly-used open source SIP server (SER) [10] for such an overload scenario. The workload consists of a mix of call setup and call handoff SIP messages and the classifier is programmed with rules that prioritize handoffs over call setups (reflecting typical message prioritization used by mobile service providers). We show that, while SER can process about 40K messages/sec (in a FIFO manner), our classifier can examine and prioritize about 105K messages/sec during overload. With the classifier operating at peak throughput, SER’s processing rate drops to about 31.6 K messages/sec, but it should be noted that the processed messages reflect as much of the high-

value messages as available in the input stream. [We ran our classifier and SIP Express Router (SER) 0.9.6 [10] on a dual 3.0 GHz Xeon with 4.5GB of RAM. We used two identical 1.7 GHz Pentium IVs with 512MB of RAM to send and receive the messages. All machines were connected via a 1Gbps Ethernet network. ]

While overload control is one of the many possible applications of our rule-based classifier, in this paper we want to motivate its use in tracking the end-to-end message flow path of SIP dialogs and sessions, whether synthetic (for testing purposes) or actual. In the following section, we explain how the classifier may be used to generate footprint records from SIP messages, and outline how these footprint records may be centrally collated by a monitoring engine to track the progress of SIP dialogs/sessions. It should be noted that we are not monitoring the media part, but rather the control part (SIP) of a session.

#### 4. CHALLENGES OF MANAGING A SIP NETWORK

Let us first motivate the challenges through a common problem. When a phone call fails to go through, especially in early stages of a deployment, users contact a help desk. In such situations, it may not always be possible to retrace the user's original call context, by retrieving the logs from all possible servers and trying to retrace the path of the call and detecting the root cause by inspecting logs. For one, every server may not maintain a call log (SIP systems, unlike phone systems, contain a distributed set of servers and a service is obtained by routing a call between a subset of these servers, which may vary between call to call); it is usually only a few key servers that maintain call-detail records, e.g., outbound SIP proxy that forwards calls outside the domain. In such a situation, it is often useful if the help desk can retry the call and retrace the path of a call within the network in real-time. Besides, real-time monitoring of calls is useful in tracking individual ongoing calls as well as aggregate network information for efficient network management operations.

A management solution is typically expected to monitor functionally and operationally the various SIP based associations between the endpoints. SIP transactions are typically very short-lived. Hence, they may not be suitable for real-time operational monitoring. But, the functional validation of SIP network with the monitoring system can be done using SIP transaction model and a synthetic SIP transaction. SIP dialogs and sessions, on the other hand, are quite enduring. Therefore, functional monitoring and real-time operational monitoring of dialogs and sessions for online mining of individual dialog and session information as well as the aggregate information, are quite feasible. Central to our approach for monitoring SIP dialogs or sessions, is the notion of a "model" of the dialog/session. Here, however, we will only discuss about the state model for SIP dialogs. The model, derived from the SIP specification, captures the various *states* the dialog can go through, along with the *footprint patterns* expected at each state. States correspond to the progress of the dialog, starting from an invitation being sent out, intermediate routing through different proxies, forking of messages if needed, and arrival at the destination(s). To mention some other possibilities, calls may also be put on hold, forwarded or transferred. Examples of such call-flow can be found in [23] which show possible paths along which a dialog may progress. These can be represented by

corresponding state-machine models. State transitions in these models are triggered by the sending and receiving of SIP messages (simplified into *footprint records* in our approach). A footprint record is obtained by intelligently extracting relevant portions of a SIP message by a classifier attached to a SIP proxy. For example, if Call-Id is invariant for a particular dialogue, a footprint record may only contain the message type (e.g. Invite), the Call-Id, and the proxy address of the SIP server where the message is being sent from/received. Since the classifier operates at the OS layer, it enables real-time generation of footprint records. The footprint records are matched against the footprint patterns associated with the different states of the dialog model. A footprint pattern may be considered as an abstract representation of a footprint record, containing placeholders for concrete header values, which are instantiated during matching. For example, a footprint pattern "INVITE <Proxy-Address> <Call-id>" corresponding to a state where an invitation is in transit, may match with a footprint record "INVITE 9.123.44.78 ac34h56@p33.atlanta.com", with Call-id bound to ac34h56@p33.atlanta.com and Proxy-Address set to 9.123.44.78. As a SIP invitation proceeds through the system, the Call-id will remain the same, but the Proxy-address will change, and these bindings will allow us to trace the call through the network.

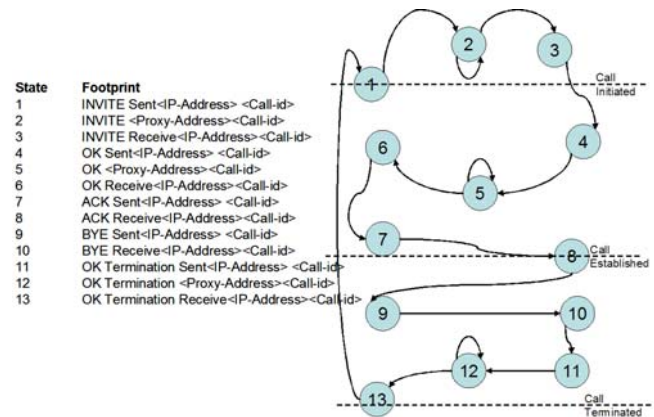


Figure 3. Footprint pattern and state model for call setup.

As an example, let us consider the modeling of a basic SIP-based call setup service between two SIP user agents, as shown in Figure 3. The procedure of a call setup is initiated once the INVITE message is sent from sending user agent. This corresponds to a state of the overall call progress and is represented by the footprint INVITE Sent <IP-Address><Call-id>. Similarly the state corresponding to the INVITE message being propagated through the proxies can be represented by INVITE <Proxy-Address><Call-id>. Finally, when the INVITE message reaches the destination user agent, it corresponds to the state represented by INVITE Receive <IP-Address><Call-id>. The states corresponding to the propagation of the OK, ACK (leading to the call getting established) and BYE, OK messages (for call termination) are similarly represented. This model can be used by the monitoring system to track in real-time the dialogs which are put on hold. Note that although the message diagram shows only one intermediate proxy, there can be a finite number of such proxies in general. This does not indicate an explosion in states as number of proxies increase, since the states and footprint patterns as explained above are abstract, and not bound to one particular proxy. Rather, they are dynamically bound to different

proxies as the dialog progresses. In a similar way, we can derive state machines for other SIP services (e.g. call on hold, forwarding etc.) and associate appropriate footprint patterns that would enable us to track those services. Such models may be composed hierarchically to encode different possibilities in a SIP dialog.

Figure 4 depicts the architecture of our SIP monitoring system. The two major components of the system are (i) SIP classifier, and (ii) SIP Monitoring Engine (SME). The monitoring algorithm in SME requires identifying the footprint records corresponding to a particular call for monitoring individual calls. For a non-forking and non-redirectioned call, the combination of Call-ID, and the tag values in the From and To headers remain unchanged. But

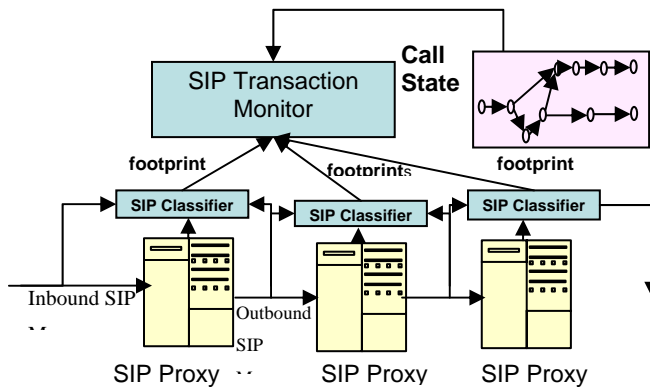


Figure 4. Monitoring architecture.

when a call forks or the call gets redirected, each forked/redirected leg of the call has a different tag value in the To header. A common use for destination URI re-writing is to instantiate a URI such as sip:bob@biloxi.com to sip:bob@bobs-host.biloxi.com, i.e. resolve a user within a domain to a specific host in the domain, (which for example could be running a softphone). Clearly, the ability to correlate the incoming INVITE (with destination sip:bob@biloxi.com) with the outgoing INVITE (to bob@bobs-host.biloxi.com) is useful for determining whether the messages and hence the corresponding footprints belong to the same call or not. Another instance of redirection is where an INVITE for sip:bob@biloxi.com from user (UA1) is responded with a 302 MOVED response by a redirect server<sup>1</sup> (say, P1), with a Contact: sip:bob@blues.com. This 302 message on the return path may trigger a new INVITE using sip:bob@blues.com at a proxy (P2) which is eventually routed to Bob's client (UA2). Thus, being able to correlate the INVITE with the corresponding 302 would allow a real-time monitoring system to infer from the footprints that the call originating from UA1 traversed through P1 and P2 with aforementioned modifications to the message, before reaching UA2, i.e. the call was forwarded to a new destination and the identity of the forwarding entity. In the above two instances, an invariant is the Call-ID header value, which can be used for correlating the inbound and outbound INVITE messages

<sup>1</sup> A Redirect server is a SIP server that resolves addresses and replies with 300 level responses, thus pushing routing to the edge of the network.

belonging to a particular call. However, there are many other instances where even the Call-ID field is not sufficient to correlate messages. In services such as call transfer or joining a conference call, a different subset of headers (and their values) needs to be used to correlate the messages. For example, when a call is transferred to a third party by using a REFER message[11], the REFER contains a header Refer-to (and a sub-header/option Replaces) which can be used to associate with the existing call that is being transferred. The programmability of the classifier enables such correlation of the inbound and outbound SIP messages. The SIP classifier, as shown in 4 can be logically thought of as monitoring both incoming and outgoing messages, with different sets of rules on each side. One way to identify how a SIP message is transformed while traversing a SIP proxy, is to place a pair of rules, one on the incoming and the other on the outgoing message stream of the proxy: by correlating matching messages, we are able to identify the specific transformation of SIP messages belonging to a particular dialog, e.g. the destination URI was re-written for a message that had a matching Call-ID. As another example, consider how we identify when a new participant joins an ongoing two-party call converting it to a 3-party conference call: classifier can be programmed to look for Join header (on a INVITE message from the new party) whose values matches any of the in-progress Dialog-IDs (here, the ability to maintain state at the classifier comes in handy).

The SME takes in the following inputs: (a) footprint records of ongoing SIP dialogs and sessions, generated by the classifiers and (b) a model of SIP dialogs or sessions, in terms of the states, and the corresponding expected footprint patterns. Given these, for each ongoing dialog or session instance, SME employs a model matching algorithm to determine in real-time the state the instance may be in. In addition, aggregate-level information e.g. number of instances at any given state of the model, may also be determined in real-time. Such statistics are helpful in efficient management of networks.

Now let us see how the monitoring system can be deployed in the above context of the help desk agent. The agent can sketch the expected call flow in the network and subsequently derive a call state model for a sample service scenario. The SIP classifiers can then be configured at the servers in the call path with appropriate rules to monitor a call in the real-time and to confirm if the expected changes are taking place, i.e. retrace the path of the call in real-time. If an anomaly is detected, additional rules can be instantiated to dig deeper. In essence, the operator can create a stimulus and the monitoring system can be used to detect the stimulus and its response at different parts of the system. Besides such functional testing of network, the monitoring system yields aggregate (e.g., number of dialogs in a particular state, average dialog termination time etc.) as well as individual information (e.g., which state is a dialog in at a particular instant of time) on the call paths of the ongoing SIP dialogs and sessions, which is useful in the deployment of effective network management tools such as load balancing. Problem determination systems may also make use of such information to narrow down possible root causes (e.g. call-initiation state machine in Figure 3 may not be completely traversed by calls if there is a problem with call setup in the SIP network).

## 5. RELATED WORK

Most of the available network monitoring tools such as Netcool [12] and SNMP [13] operate on single network entity (e.g. an Internet Application Server) and hence fail to give end-to-end performance of network sessions. Some tools such as Netflow [14] and Gigascope [15] provide end-to-end performance of network flows, but only for lower layer - such as IP and TCP - network flows. Indeed there are a host of tools [16], like SIPFlow from Sipient [17] available for monitoring application layer protocols such as SIP, but they are mostly designed for monitoring of individual SIP servers. SIPFlow allows SIP message filtering based on message types only, i.e. is fairly coarse-grained, while the SIP Classifier is both programmable to operate user-specified rules on any subset of the SIP message, as well as on the past history of the message stream. The current state of the art indicates that currently there is no tool available for real-time monitoring end-to-end SIP dialogs that can be used for functional network testing and subsequent root cause analysis, and operational SIP network management.

## 6. CONCLUSIONS

End-to-end SIP dialog and session monitoring in real-time is required for functional testing of SIP overlay network containing malfunctioning or ill-configured SIP entities or for efficient runtime SIP network operations such as load balancing and problem determination. To meet this requirement, we have designed a monitoring system consisting of a state-based monitoring engine and a SIP message classifier module for collecting appropriate monitoring information from the network in real-time. Further work needs to be done to test the monitoring system and evaluate its performance in a SIP-based network testbed.

## 7. REFERENCES

- [1] B. Campbell et al. *The Message Session Relay Protocol*. SIMPLE Working Group Internet-Draft, Jan 2004.
- [1] B. Campbell, editor. *Session Initiation Protocol (SIP) Extension for Instant Messaging*. RFC 3428.IETF, Dec 2002
- [3] Handley, M. and V. Jacobson, *SDP: Session Description Protocol*, RFC 2327, IETF Apr 1998.
- [4] A. Niemi et al., *Session Initiation Protocol (SIP) Extension for Presence Publication*. SIMPLE Working Group Internet-Draft, June 2003.
- [5] H. Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. RFC 1889.IETF, Jan 1996.
- [6] J. Rosenberg et al. *SIP: Session Initiation Protocol*. RFC 3261. IETF, June 2002.
- [7] G. Camarillo and Miguel-Angel Garcia-Martin. *The 3G IP Multimedia Subsystem (IMS) : Merging the Internet and Cellular worlds*. John Wiley and Sons, 2004.
- [8] S. Donovan, The SIP INFO Method. RFC 2976. IETF, Oct 2000.
- [9] J. Rosenberg, The Session Initiation Protocol (SIP) UPDATE Method, RFC 3311. IETF, Sept 2002
- [10] <http://developer.berlios.de/projects/ser/>
- [11] A. Johnston, ed. *Session Initiation Protocol Service Examples*. Internet Draft version 12, January 2007

[12] IBM Tivoli Netcool:

<http://www-306.ibm.com/software/tivoli/welcome/micromuse/>

[13] SNMP Version 3 (snmpv3),

<http://www.ietf.org/html.charters/OLD/snmpv3-charter.html>

[14] Cisco IOS Netflow,

[http://www.cisco.com/en/US/products/ps6601/products\\_ios\\_protocol\\_group\\_home.html](http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html)

[15] Gigascope, <http://www.research.att.com/viewProject.cfm?prjID=129>

[16] SIP Testing and Measurement,

<http://www.sipcenter.com/sip.nsf/html/Testing+Measurement>

[17] Sipient Systems, <http://www.sipient.com/>

[18] A.Acharya, X. Wang and C. Wright. A Programmable Message Classification Engine for Session Initiation Protocol (SIP). IBM Research Report RC 24022, Aug 2006.