Internet Engineering Task Force
Internet Draft
draft-ietf-sip-callerprefs-02.txt
July 13, 2000
Expires: January 2001

SIP WG
Schulzrinne/Rosenberg
Columbia U./dynamicsoft

SIP Caller Preferences and Callee Capabilities

STATUS OF THIS MEMO

Abstract

This document describes a set of extensions to SIP which allow a caller to express preferences about request handling in servers. These preferences include the ability to select which URIs a call gets proxied or redirected to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining three new request headers, Accept-Contact, Reject-Contact and Request-Disposition, which specify the callers preferences. The extension also defines new parameters for the Contact header. These extra parameters are present in the Contact header in REGISTER requests, and are used to associated attributes with particular addresses.

1 Introduction

When a SIP [1] server receives a request, there are a number of

decisions it can make regarding processing of the request. These include

- o whether to proxy or redirect the request;
- o which URIs to proxy or redirect to;
- o whether to fork or not;
- o whether to search recursively or not;
- o whether to search in parallel or sequentially;

The server can base these decisions on any local policy. This policy can be statically configured, or can be based on programmtic execution or database access.

However, the administrator of the server is the not the only entity with an interest in call processing. There are at least three parties which have an interest: (1) the administrator of the server, (2) the callee, and (3) the caller. The directives of the administrator are embedded in the policy of the server. The preferences of the callee can be expressed most easily through a script written in the call processing language (CPL) [2]. However, no mechanism exists to incorporate the preferences of the caller. This extension fills that gap by specifying mechanisms by which a caller can provide preferences on processing of a call. These preferences include the ability to select which URIs a call gets proxied or redirected to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining three new request headers, Accept-Contact, Reject-Contact and Request-Disposition, which specify the callers preferences. The extension also defines new parameters for the Contact header. These extra parameters are present in the Contact header in REGISTER requests, and are used to associated attributes with particular addresses.

2 Overview of Operation

This extension defines a set of additional parameters to the Contact header. These parameters specify attributes that define the characteristics of the UA at the address in the header. For example, there is a mobility parameter which indicates whether the UA is fixed or mobile. When a UA registers, it places these parameters in the Contact headers to characterize the URIs it is registering. This allows the proxy to have information about the contact addresses for a user.

The INVITE message, and its response, also contain Contact headers

used to route subsequent messaging. This extension allows these headers to contain extension parameters to provide additional information about the type of user agent being used. For example, by including the feature parameter with value "voicemail" in the 200 OK to an INVITE, the UAS can indicate to the UAC that it is a voicemail server. This information is useful for user interfaces, as well as automated call handling.

When a caller sends an INVITE, it can optionally include new headers which request certain handling at a proxy. These preferences fall into two categories. The first category, carried in the Request-Disposition header, describe desired server behavior. This includes whether the caller wishes the server to proxy or redirect, and whether sequential or parallel search is desired. These preferences can be applied at every proxy or redirect server on the call signaling path.

The second category of preferences are carried in both the Accept-Contact and Reject-Contact headers. These preferences contain rules that describe the set of desired URIs that the caller would like the server to proxy or redirect to. These rules are matched against the Contact headers sent in a registration (or through some other configuration means). If a rule in a Reject-Contact header matches a Contact header, that address is not proxied or redirected to. If a rule in a Accept-Contact header matches a Contact header, the q values in the rule are combined with the q values in the Contact header, resulting in a "merged" q value. This merged q value is then used by the proxy to determine the ordering of addresses to proxy or redirect to.

Note that this second category of preferences can only be applied at a proxy which accesses a registration database.

3 Design Alternatives

There are a number of alternatives for expressing caller preferences. Ideally, caller preferences, callee preferences, and administrator prefernces "meet" at each server which makes processing decisions. In practicality, a callee cannot install logic at each server in the network. It can only do so (using the CPL, for example), at those servers with which it has some kind of established trust relationship. These servers are those whose main goal is to provide services for the callee.

One might try to place caller logic at these "callee servers" in much the same way the callee places logic there - through the CPL or some other programmtic directives. However, this is also infeasible. A caller cannot apriori install logic in every server for every

individual he might call.

As another alternative, one could embed a script in the request, to be executed by proxy or redirect servers when making forwarding decisions. This would be an application-layer version of active networks. However, the generality of a script does not seem to be needed. It also makes combining caller and callee preferences a rather difficult problem and raises possible performance and security issues. Unlike the callee script, which needs to handle unknown callers, with a wide range of call properties, at unknown times in the future, a caller knows all but the set of communications capabilities of the callee. The caller can present the servers with its preferences on a call-by-call basis. Callers can thus place their preferences for this particular call in the request message. We propose a simple ordered list of preferences to make it possible to reconcile caller and callee preferences algorithmically.

In summary, there is a strong asymmetry in how preferences for callers and callees can be presented to the network. While a caller takes an active role by initiating the call, the callee takes a passive role in waiting for calls. This motivates the use of callee-supplied scripts and caller preferences included in the call request.

This asymmetry is also reflected in the appropriate relationship between caller and callee preferences. A server for a callee SHOULD respect the wishes of the caller to avoid certain locations, while the preferences among locations has to be the callee's choice, as it determines where, for example, the phone rings and whether the callee incurs mobile telephone charges for incoming calls.

The problem of feature negotation has also been approached in a more general way by [3]. However, that proposal is far more complicated than appears to be needed here, with syntax that does not fit into the current SIP syntax structure.

4 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [4] and indicate requirement levels for compliant SIP caller preferences implementations.

5 Header Field Definitions

Table 5 specifies an extension of Table 5 in RFC 2543 [1] for the three new headers defined here.

	where	enc	e-e	ACK	BYE	CAN	INV	OPT	REG
Accept-Contact	R	n	h	-	o	o	o	o	-
Reject-Contact	R	n	h	-	o	o	o	o	-
Request-Disposition	R	n	h	-	o	o	o	o	o

Table 1: Summary of header fields. "o": optional "-": not applicable, "R': request header, "r": response header, "g": general header, "*": needed if message body is not empty. A numeric value in the "type" column indicates the status code the header field is used with.

5.1 Contact, Accept-Contact and Reject-Contact Parameters

This specification adds the following extension parameters to the Contact header field and defines the same parameters for the Accept-Contact and Reject-Contact header fields. These parameters apply to a single URI. When used in a Contact header, they specify characteristics of that URI. When used in the Accept-Contact or Reject-Contact headers, they specify rules to apply for matching URIs.

```
cp-params   =  class-param | duplex-param |
               feature-param | language-param | media-param |
               mobility-param | other-param
class-param     =  "class" "=" <"> [<!>] 1#class-value <">
duplex-param    =  "duplex" "=" <"> [<!>] 1#duplex-value <">
feature-param   =  "feature" "=" <"> [<!>] 1#feature-value <">
language-param  =  "language" "=" <"> [<!>] 1#language-tag <">
media-param     =  "media" "=" <"> [<!>] 1#media-value <">
mobility-param  =  "mobility" "=" <"> [<!>] 1#mobility-value <">

other-param     =  other-name "=" <"> [<!>] 1#other-value <">
mobility-value  =  "fixed" | "mobile" | other-value
class-value     =  "personal" | "business" | other-value
duplex-value    =  "full" | "half" | "receive-only" |
                   "send-only" | other-value
media-value     =  ( "*/*" | (type "/" "*") |
                   (type "/" subtype) )
feature-value   =  "voice-mail" | "attendant" | other-value
other-name      =  UTF8-TOKEN
other-value     =  UTF8-TOKEN
UTF8-TOKEN      =  <any UTF-8 character encoding
                    except separator, CTL, and LWS>
```

The BNF and semantics of the language-tag are defined in Section 3.10 of RFC 2616 [5]. Note, however, that in their usage here they are case sensitive, and MUST appear as all lowercase. Also note that there MUST NOT be any linear white space between the tokens and quoted strings of the media-value. This is to align with HTTP 1.1 [5].

The exclamation mark in the parameter value MUST NOT be included if the cp-params are included in a Contact header. Most importantly, there MUST NOT be more than one class-value, duplex-value, or mobility-value when cp-params is included in a Contact header. These parameters refer to attributes which are mutually exclusive. As a result, a URI can only have one as a characteristic, whereas a rule in the Accept-Contact or Reject-Contact can specify more than one.

The parameters and their values have the following meanings:

class: The class parameter indicates whether the UA is found in a residential or business setting. (A caller may defer a personal call if only a business line is available, for example.)

duplex: The duplex parameter lists whether the UA can simultaneously send and receive media ("full"), alternate between sending and receiving ("half"), can only receive ("receive-only") or only send ("send-only"). Typically, a caller will prefer a full-duplex UA over a half-duplex UA and these over receive-only or send-only UAs.

features: The feature parameter enumerates additional features of the UA. It is assumed that these features are orthogonal, and could occur in any combination. "voice-mail" means that an automated system exists at this UA, which is capable of recording messages. "attendant" means that a human operator is available to take messages.

language: The language parameter lists the languages spoken by user or system behind the UA. This parameter may, for example, be used to have a caller automatically be directed to the appropriate attendant or customer service representative. Note that this parameter has a different functionality than the Accept-Language and Content-Language header fields, which describe the acceptable languages and languages used in the request and the media description, not the actual communications.

media: The media parameter lists the media types supported by the UA. In this context, supported means that the media type is acceptable as part of the media session established by SIP (and usually described by SDP [6]). It does not refer to the media types which can be supported within the bodies of SIP messages. Media types can be the standard Internet media types ("audio", "video", "text", "application"), optionally followed by a subtype (e.g., "text/html").

mobility: The mobility parameter indicates if the UA is fixed or mobile. In some locales, this may affect audio quality or charges.

In addition, the Contact header field may contain the description-param, methods-param and priority-param parameters.

The description parameter further describes, as text, the terminal. The description parameter MUST NOT be used in the matching operation described in Section 6.3.1.

The priority parameter indicates the minimum priority level this UA is to be used for. It can be used for automatically restricting the choice of terminals available to the caller. The priority parameter is not used in the matching operation described in Section 6.3.1. Its application is described in the procedure in Section 6.3.2.

The methods parameter indicates the methods this UA understands. It MUST NOT be used in any request excepting REGISTER. The methods parameter is not used in the matching operation described in Section 6.3.1. Its application is described in the procedure in Section 6.3.2.

```
priority-param     =  "priority" "=" <"> priority-value <">
description-param  =  "description" "=" quoted-string
methods-param      =  "methods" "=" <"> 1#methods-value <">
methods-value      =  ( "INVITE" | "OPTIONS" | "BYE" | "REGISTER"
                      | token)
```

Note that priority-value is defined in section 6.25 of [1].

There is some overlap between the indication of receiver capabilities in the session description message body and

the Accept-Contact and Reject-Contact header fields. However, current session description formats cannot express the preferences described here. Also, the capabilities described here are fundamental to call-routing and thus should not depend on the particulars of the various session description formats that might be used.

5.2 Accept-Contact

The syntax for the Accept-Contact header is defined below:

```
Accept-Contact  =  "Accept-Contact" ":" 1# rule
rule            =  ( name-addr | addr-spec | "*")
                   [ *( ";" (cp-params | q-param | scheme-param) ) ]
q-param         =  "q" "=" qvalue
scheme-param    =  "scheme" "=" <"> [<!>] 1#scheme <">
```

The header field specifies contact addresses which are acceptable to the caller. If a "*" is specified instead of a name-addr or addr-spec, it means the UAC doesn't care about the URI of the user eventually reached. Only the parameters of the Contact header are important. If the name-addr or addr-spec is present, and the userinfo field of the SIP URL is not present, it means the UAC doesn't care about the username of the user eventually reached. If the host portion of the SIP URL is a hostname, and has the value "x", it means the UAC doesn't care about the host portion of the URI eventually reached. If the name-addr or addr-spec is present, and contains URI parameters, if means the UAC wishes to be connected to an address that has been registered with these parameters.

We use "x" as the wildcard domain because of the URI formatting constraints. The domain must be present in a SIP URL, and cannot be the "*" character. The "x" character is allowed and looks kind of similar.

The scheme parameter describes the set of URI schemes which the caller is willing to accept redirects to or communicate with. The BNF for scheme is given in RFC 2396 [7], and can be any valid URI scheme.

In the following example, the caller would prefer not to talk to sales@acme.com later. She has a slight preference for fixed as opposed to mobile phones.

```
Accept-Contact: sip:sales@acme.com ;q=0,
  *;media="!video" ;q=0.1,
  *;mobility="fixed"  ;q=0.6,
  *;mobility="!fixed" ;q=0.4
```

In the next example, the caller would prefer to speak to someone from sales.org that supports video:

```
Accept-Contact: sip:sales.org;media="video"
```

5.3 Reject-Contact

The Reject-Contact header field specifies a list of URIs that the caller does not wish to communicate with. The BNF for the header is:

```
Reject-Contact  =  "Reject-Contact" ":"
                   1# ( ( name-addr | addr-spec | "*")
                   [ *( ";" cp-params | scheme-param ) ] )
```

If name-addr or addr-spec is not present (the "*" is present), it means the UAC does not care about the particular user or domain the request is routed to. The cp-params are used to filter out contact addresses based on their parameters alone. This process is described in Section 6.3.1. If either name-addr or addr-spec is present, and the URI does not contain a userinfo field, it means the UAC does not have a preference regarding the user name and/or password of the UA eventually reached. If domain of the URI is equal to "x", it means the UAC does not have a preference regarding the domain of the UA eventually reached.

The scheme parameter describes the set of URI schemes which the caller is not willing to accept redirects to or communicate with. The BNF for scheme is given in RFC 2396 [7], and can be any valid URI scheme.

5.4 Contact Header

The cp-params parameter is allowed as an extension attribute to the Contact header, along with the priority-param, methods-param and description-param. This effectively means that the BNF for

extension-attribute, defined in Section 6.13 of RFC 2543 [1] can be redefined as:

```
extension-attribute  =  (cp-params |
                         priority-param | methods-param |
                         description-param |
                         (extension-name [ "=" extension-value]))
```

The example below describes a SIP terminal whose owner speaks English, Spanish and German. The terminal is capable of sending and receiving audio and video and can participate in a chat session. However, the owner only wants callers to use the terminal if the call is of priority "urgent" or higher. This Contact header would normally be included in a REGISTER message.

```
Contact: Carol <sip:carol@example.com> ;language="en,es,de"
  ;media="audio,video,application/chat"
  ;duplex="full"
  ;priority="urgent"
```

As another example, an INVITE message is sent with a Contact header that includes some of the parameters defined here:

```
INVITE sip:user@example.com SIP/2.0
Via: SIP/2.0/UDP host.example.com
From: sip:caller@university.edu
To: sip:user@example.com
Call-ID: 9sdnasdbasd@1.2.3.4
CSeq: 3 INVITE
Contact: Joe Caller <sip:caller@university.edu>;mobility="mobile"
```

In this case, Joe is indicating he is calling from a mobile host.

5.5 Request-Disposition

The Request-Disposition header field specifies caller preferences for how a server should process a request. Its value is a list of tokens, each of which specifies a particular feature.

When the caller specifies a feature, the server SHOULD treat it as a hint, not as a requirement and MAY ignore the feature request.

The header field has the following syntax:

```
Request-Disposition  =  "Request-Disposition" ":"
                        1# (proxy-feature | cancel-feature |
                        fork-feature | recurse-feature |
                        parallel-feature | queue-feature |
                        extension-feature)
proxy-feature        =  "proxy" | "redirect"
cancel-feature       =  "cancel" | "no-cancel"
fork-feature         =  "fork" | "no-fork"
recurse-feature      =  "recurse" | "no-recurse"
parallel-feature     =  "parallel" | "sequential"
queue-feature        =  "queue" | "no-queue"
extension-feature    =  token
```

proxy-feature: This feature indicates whether the caller would like each server to proxy or redirect.

cancel-feature: This feature indicates whether the caller would like each proxy server to send a CANCEL request downstream in response to a 200 OK from the downstream server, or whether this function should be left to the caller.

fork-feature: This feature indicates whether a proxy should fork a request, or proxy to only a single address. If the server is requested not to fork, the server SHOULD proxy the request to the "best" address (generally the one with the highest q value). The feature is ignored if "redirect" has been requested.

recurse-feature: This feature indicates whether a proxy server receiving a 300-class response should send requests to the addresses listed in the response (i.e., recurse), or forward the list of addresses upstream towards the caller. The feature is ignored if "redirect" has been requested.

parallel-feature: For a forking proxy server, this feature indicates whether the caller would like the proxy server to proxy the request to all known addresses at once, or go

through them sequentially, contacting the next address only after it has received a non-200 or non-600 final response for the previous one. The feature is ignored if "redirect" has been requested.

queue-feature: If the called party is temporarily unreachable, e.g., because it is in another call, the caller can indicate that it wants to have its call queued rather than rejected immediately. If the call is queued, the server returns "182 Queued". A pending call be terminated by a SIP CANCEL or BYE request.

Example:

```
Request-Disposition: proxy, recurse, parallel
```

The Request-Disposition header also allows an extension-feature, which can be used to specify additional dispositions. Section 8 specifies procedures for registration of new extension-features. Note that features requested in Request-Disposition MUST always be optional for the proxy to apply. In other words, if the client has a feature it insists on using, the Request-Disposition header MUST NOT be used for that purpose.

6 Protocol Semantics

6.1 UAS Behavior

User agent servers MAY include cp-params, priority-param, methods-param or description-param parameters as part of each Contact addresses they register. These parameters can be set through configuration, user input, or any means the implementor seeks to use. They SHOULD reflect actual characteristics of the URLs being registered.

Furthermore, the REGISTER request MAY contain a Require header with the option tag "pref" if the client wants to be sure that the registration server honors caller preferences.

When a UAS receives a request with the Accept-Contact, Reject-Contact and Request-Disposition, it MAY ignore these headers so long as it does not redirect the request. If the request is redirected, the UAS SHOULD follow the rules described in Section 6.3 for a proxy/redirect

server.

6.2 UAC Behavior

A UAC wishing to express preferences for a request includes the Accept-Contact, Reject-Contact, or Request-Disposition headers in the request, depending on its particular preferences. No additional behavior is required after the request is sent.

If the client wants to be sure that servers understand the headers described in this specification, it MAY include a Proxy-Require and Require option tag of "pref". However, this is NOT RECOMMENDED, as it leads to interoperability problems. In any case, client preferences can only be considered as preferences - there is no guarantee that the requested service or capability is executed. As such, inclusion of Proxy-Require and Require does not mean the preferences will be executed.

6.3 Proxy Behavior

The behavior described here assumes a server (proxy or redirect) has received a valid request with either the Accept-Contact or Reject-Contact headers, and that this proxy has a list of Contact headers obtained from looking up the Request-URI in the location service. The location service may have obtained this data through registrations, as described in Section 6.1, but other means may exist.

The processing depends heavily on a rule matching operation. This operation takes a rule (defined as a single element from the comma separate list of elements in the Accept-Contact or Reject-Contact headers), and matches it against the contact list obtained from the location service.

6.3.1 Rule Matching Procedures

The contact list is composed of a set of contact entries. Each contact entry consists of a URI along with a set of parameters. A rule, like a contact entry, consists of a URI (or the "*" character), and a set of parameters. If the rule does not contain a URI (just the "*" character), the rule matches the contact entry if and only if the parameters in the rule and the parameters in the contact entry match. If the rule contains a URI, both the URI and parameters must match for the rule to match the contact entry.

The URI in the rule and the URI in the contact entry match depending on the scheme. For non-SIP URIs, matching is based on the URI equivalency rules for that scheme. For SIP URLs, the userinfo, host, and URI parameters must match, where matching is defined as follows.

Note that these matching rules are not the same as the general URI matching rules in SIP [1].

If the rule contains a userinfo field, that userinfo field must match the userinfo field in the URI in the contact entry. Matching is based on case sensitive string comparison. If the rule contains a userinfo field, but the URI in the contact address does not, the userinfo in the rule does not match the userinfo in the contact entry. If the rule does not contain a userinfo field, the userinfo component matches.

If the rule contains a host not equal to "x", the host in the URI of the rule must match the host of the URI in the contact entry. Matching is based on case insensitive string comparison. If the rule has a host equal to "x", it matches any value of the host in the URI in the contact entry.

If the URI in the rule contains URI parameters (port is considered a URI parameter for purposes of this discussion), each parameter in the URI in the rule must match a parameter in the URI in the contact entry. Matching is based on case sensitive string comparison of both parameter names and values. Note, however, if the URI in the rule contains a parameter with a default value, this matches a contact entry with a URI that does not contain this parameter. If the URI in the rule contains a URI parameter that is not the default value, this does not match a contact entry whose URI does not contain this parameter. If there are no URI parameters in the rule, this is considered a match to any set of URI parameters in the contact entry.

To determine if the parameters in the rule match the parameters in the contact entry, the following process is followed.

The parameters match if and only if each parameter in the rule matches the contact entry. A single parameter in the rule matches the contact entry if that parameter is present in the contact entry, and their values match. If a parameter exists in the rule, and there is no parameter of the same name in the contact entry, whether or not this is a match is context dependent. If the rule is present in the Accept-Contact header, it is considered a match. If the rule is present in the Reject-Contact header, it is not considered a match.

Parameter names are matched by case-sensitive comparison. Parameter values are matched by set-comparisons. Parameter values in quoted strings are interpreted as sets, with elements separated by commas. Two elements in the set match if they are equal based on a case sensitive string comparison. There are two cases: if the quoted-string parameter value in a rule starts with an exclamation mark (!), the rule matches if the intersection of the set in the rule and in

the contact entry is empty. Otherwise, the rule matches if the intersection of the rule set with the contact set is non-empty. Note that this process does not apply to the priority-param, methods-param, description-param or scheme-param.

> Case sensitive comparisons are necessary because of internationalization. Case insensitive matching in UTF-8 depends on regional rules, and overly complicates the procedure.

If there is a scheme-param in the rule, and the quoted-string parameter value in the rule starts with an exclamation mark, the scheme of the URI in the contact entry must not match any of the schemes listed in the rule. If the quoted-string parameter value in the scheme-param doesn't start with an exclamation mark, the scheme of the URI in the contact entry must match one of the schemes listed in the rule. Matching of schemes is done by case insensitive string comparison [7].

The C code below describes the matching procedure between a rule and a contact entry. The function intersect() takes two arrays of strings, and returns true if there are any values common to both arrays, false otherwise. The function getparameterbyname() takes a rule and a string defining a parameter name. It returns a parameter from the rule with that name. However, if the parameter name is "scheme", the function returns a "scheme" parameter. This parameter structure has the name set to "scheme", the exclamation set to false, and the values array with a single value, containing the name of the scheme in the URI of the rule.

```
/* context values */
#define ACCEPT_CONTACT 0
#define REJECT_CONTACT 1

typedef int boolean;

typedef struct uri_parameters_s {
  char *name;
  char *value;
} uri_parameters_t;

typedef struct uri_s {
  char *scheme;
  char *userinfo;
```

```
    char *host;
    uri_parameters_t **params;
  } uri_t;

  typedef struct parameter_s {
    char *name;           /* parameter name */
    boolean exclamation;   /* whether ! was present in value */
    char **values;        /* list of elements in the value */
  } parameter_t;

  typedef struct rule_s {
    uri_t *URI;            /* URI */
    parameter_t **para;   /* list of parameters */
  } rule_t;


  /* little helper function to look up a parameter by its
     name within an entry */

  parameter_t *getparameterbyname(rule_t *r, char *name) {

    int i;
    parameter_t *p;

    if(strcmp(name,"scheme") == 0) {

      p = calloc(1, sizeof(parameter_t));
      p->name = "scheme";
      p->values = calloc(2, sizeof(char *));
      p->values[0] = malloc(sizeof(char) * (strlen(r->URI->scheme) + 1));
      strcpy(p->values[0], r->URI->scheme);

      return(p);
    }

    for(i=0; r->para[i] != NULL; i++) {

      if(strcmp(r->para[i]->name, name) == 0)
         return(r->para[i]);
    }

    return(NULL);
  }

  /* check if two sets of strings share at least one
     common value */

  boolean intersect(char *a[], char *b[]) {
```

```
    int i,j;

    for(i = 0; a[i] != NULL; i++) {
      for(j = 0; b[j] != NULL; j++) {

        if(strcmp(a[i], b[j]) == 0)
          return(TRUE);
      }
    }

    return(FALSE);
  }

  /* returns the default value of a URI parameter */
  char *defaultvalue(char *name) {

    if(strcmp(name, "transport") == 0)
      return("udp");

    return("some-value-which-matches-no parameter");
  }

  boolean matchuriparameters(uri_parameters_t **r, uri_parameters_t **e) {
    int i,j;
    boolean match;

    /* for each rule */
    for(i=0; r[i] != NULL; i++) {

      match = FALSE;
      for(j=0; e[j] != NULL; j++) {

        /* found the matching URI parameter in the entry */
        if(strcmp(r[i]->name, e[j]->name) == 0) {

          /* if they're not equal, return FALSE. Otherwise, set match
             to TRUE, indicating that we found a matching parameter*/
          if(strcmp(r[i]->value, e[j]->value) != 0)
            return(FALSE);
          else
            match = TRUE;
        }
      }

      /* parameter in rule not in entry */
      if(match == FALSE) {

        /* check if rule contains default value */
```

```
         if(strcmp(defaultvalue(r[i]->name), r[i]->value) != 0)
           return(FALSE);

       }
     }

     return(TRUE);
   }



   boolean MATCH(rule_t *r, rule_t *e, int context) {
     boolean match;
     int i;
     parameter_t *p, *q;

     match = TRUE;

     /* We represent a rule with a * as the match for URIs, as
        a URI with a scheme of * */

     if (strcmp(r->URI->scheme, "*") != 0) {

       /* the schemes must match */
       if (strcasecmp(r->URI->scheme, e->URI->scheme) == 0) {

        /* for sip, perform our SIP rules */
        if (strcasecmp(r->URI->scheme, "sip") == 0) {

          /* check for match of user and host */
          match=(((strcasecmp(r->URI->host, "x") == 0) ||
                  (strcasecmp(r->URI->host, e->URI->host) == 0)) &&
                 ((r->URI->userinfo == NULL) ||
                  ((e->URI->userinfo != NULL) &&
                   (strcmp(r->URI->userinfo, e->URI->userinfo) == 0))));

          if(match == FALSE) return(FALSE);

          /* match URI parameters */
          match = matchuriparameters(r->URI->params, e->URI->params);
          if(match == FALSE) return(FALSE);

        } else {
          /* match = scheme-appropriate comparison; */
          if(match == FALSE) return(FALSE);
        }
       } else {
         /* schemes don't match */
```

```
        return FALSE;
      }
     }

     /* compare parameters */
     for(i = 0; r->para[i] != NULL; i++) {

       p = r->para[i];

       /* is this parameter defined in the contact entry */
       if ((q =getparameterbyname(e, p->name)) != NULL) {

         /* is this an empty set match */
         if (p->exclamation == TRUE) {

           if (intersect(p->values, q->values) == TRUE) {
             return FALSE;
           }
         } else {

           /* not an empty set case */
           if (intersect(p->values, q->values) == FALSE) {
             return FALSE;
           }
         }
       } else {
         /* this parameter is not present in the entry.
            whether its a match or not is dependent on
            context */

         if(context == REJECT_CONTACT)
           return(FALSE);
       }
     }
     return TRUE;
   }
```

For example, the rule:

sip:example.com;language="!en,de"

matches the contact entry:

sip:joe@example.com;language="es,nl"

but not any of:

sip:joe@example.com;language="en"
sip:bob@example.com;language="de,en"
sip:alice@example.com;language="en,es,fi"

As another example, the rule

*;duplex="full,half"

matches the contact entry

sip:user@host;duplex="full"

but not

sip:user@host;duplex="send-only"

The rule

*;scheme="http"

matches the contact entry

http://www.example.com

A server need not be aware of the particular semantics of any of the parameters. This allows for the definition of new parameters and values without explicitly programming them into the servers.

6.3.2 Contact List Processing

Given the matching rule above, the formal processing rules at the server proceed as follows. The server begins with a contact list for the callee, and a set of rules in the Accept-Contact and Reject-Contact headers.

The server first removes any contact entry from the contact list that matches a rule in the Reject-Contact header field.

A contact entry may contain a priority parameter. This means that a request should not be proxied or redirected to that contact entry unless the request is of equal or higher priority. The priority value of the request is derived from the Priority header field. If the request does not contain a Priority header field, the request priority is set to "non-urgent". Priorities are ordered from "non-urgent" (lowest), "normal", "urgent" to "emergency" (highest). Priority values not known to the server are mapped to "non-urgent". The server then removes any contact entry from the contact list whose priority value is higher than that of the request.

A contact entry may contain a methods parameter. This means that a request should not be proxied or redirected to that contact entry unless the method of the request is listed among those in the contact entry. The server removes any contact entry from the contact list whose method list does not include the method of the SIP request.

Each rule in the Accept-Contact header field is then processed. If the rule matches a contact entry (according to the matching rule in section 6.3.1, the q value of that entry is updated, in order to incorporate the caller's preferences. If the rule does not match a contact entry, nothing is done. This document does not prescribe a specific algorithm for updating the q value. Among many possibilities, a server MAY set the q value to the average of the original value specified by the callee, and the average q value of the caller's rules that match the contact entry. This gives equal weight to caller and callee preferences. If a rule or contact entry does not have a q value, it is taken to be one (this is in agreement with the HTTP defaults). The only requirement for the updating process is that if a contact entry has a q value of q1, and the q values among the matching rules are q2,q3,..qn, the merged q value, qm, must satisfy:

MIN(q1,q2,q3,..qn) <= qm <= MAX(q1,q2,q3,..,qn)

For those contact entries which did not match any rule in the Accept-Contact header, their final q value is set to zero.

> Note that this preference computation only determines the ordering of call attempts, so that the properties of the preference computation are of secondary importance. The q-value ordering provides only limited flexibility to indicate, for example, that a particular parameter is more important than another one or that combinations of two parameters should be weighed heavily.

If the server proxies, the contact list is then sorted according to the q value. Processing from this point depends on the configuration and policy of the server. If the server elects to do a sequential proxy, it SHOULD try the highest q value contact entry first, trying addresses with decreasing q values as each attempt fails. If the server elects to do a forking proxy, it SHOULD group contact entries with "close" q values together, and try the group with the highest q value first, then the group with the next lowest q values, and so on. The precise method of the grouping is left to the implementor. A reasonable choice is to round each q value to the nearest tenth, and group those with the same rounded value.

If a proxy server is recursing, it SHOULD apply the caller preferences to the Contact headers returned in the redirect responses. Any contact entries remaining after the application of caller preferences should be added to the list of untried addresses. This list is then resorted based on q values. The server uses this list for subsequent proxy operations.

If the server is redirecting, it SHOULD return all entries in the contact list, including those with a zero q value.

If the server is executing any other type of policy, as a general guideline, it SHOULD prefer contact entries with higher q values than those with lower q values.

6.3.3 Request-Disposition Processing

If the request contains a Request-Disposition header, the server SHOULD execute the behaviors described by the tokens, unless it has local policy configured to direct it otherwise.

7 Interactions with CPL

When the called party has a Call Processing Language (CPL) [8] script present, feature interactions are introduced. CPL addresses this by

allowing the CPL script to control whether caller preferences are applied to the location list or not. CPL also allows the called party to discard certain rules from the caller preferences before their application. For more information, see [8].

8 IANA Registration Procedures

8.1 cp-params

New cp-params parameters and values can be defined at any time and registered with IANA. When registering new parameters and values, the following information MUST be provided:

Contact: Name, organization, email address, and phone number of person registering the attributes.

Attributes: A list of the new attributes being registered. For each, the meaning of the attribute must be described, in sufficient detail so that a user agent would be able to ascertain whether the parameter applies to it, and if so, which value to use. The attributes MUST also be associated with a finite set of values, each of which is a valid UTF8-TOKEN. For each value, a description of the value must be provided. The registration MUST indicate whether the parameter values are mutually exclusive or not; that is, whether only one, or more than one, can appear in the Contact header.

8.2 Request-Disposition

New request disposition values can be defined at any time and registered with IANA. Request dispositions MUST always be optional for a proxy to grant. When registering new values, the following information MUST be provided:

Contact: Name, organization, email address, and phone number of person registering the value.

Behavior: The requested behavior when the attribute appears in the request disposition.

Interactions with other Values: Any interactions with other values specified or here registered with IANA.

9 Changes since -01

- Specified that parameters can be included in Contact in INVITE and its 200 OK response as well.
- Allow Request-Disposition in REGISTER.
- Added IANA registration for request-disposition values.
- Added methods to contact parameters.
- Alterned bnf so that cp-params can be applied to Contact, Accept-Contact, and Reject-Contact.
- Updated code, compiled and verified
- Removed ring-feature

10 Open Issues

- Is IANA registrations for request-disposition values a good idea?

11 Security Considerations

The presence of caller preferences in a request has a significant way in which the request is handled at a server. As a result, is is especially important that requests with caller preferences be authenticated. The same holds true for registrations with contact parameters.

Processing of caller preferences requires set operations and searches which can require some amount of computation. This enables a DOS attack whereby a user can send requests with substantial numbers of caller preferences, in the hopes of overloading the server. To counter this, servers SHOULD reject requests with too many rules. A reasonable number is around 20.

12 Acknowledgements

Parameters of the terminal negotiation mechanism in Section 5.1 were influenced by Scott Petrack's CMA design. Jonathan Lennox and John Hearty provided helpful comments.

13 Author's Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

14 Bibliography

[1] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[2] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," Request for Comments 2824, Internet Engineering Task Force, May 2000.

[3] G. Klyne, "A syntax for describing media feature sets," Request for Comments 2533, Internet Engineering Task Force, Mar. 1999.

[4] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments 2119, Internet Engineering Task Force, Mar. 1997.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol -- HTTP/1.1," Request for Comments 2616, Internet Engineering Task Force, June 1999.

[6] M. Handley and V. Jacobson, "SDP: session description protocol," Request for Comments 2327, Internet Engineering Task Force, Apr. 1998.

[7] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifiers (URI): generic syntax," Request for Comments 2396, Internet Engineering Task Force, Aug. 1998.

[8] J. Lennox and H. Schulzrinne, "CPL: a language for user control

of internet telephony services," Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

Table of Contents