Welcome to Acme Packet TechTalk!

Lots to cover today! We will begin the presentation promptly at 11:01 am ET.

Configuring Header Manipulation Rules - Part 1

January 2013

Disclaimer

Acme Packet has made no commitments or promises orally or in writing with respect to delivery of any future software features or functions. All presentations, RFP responses and/or product roadmap documents, information or discussions, either prior to or following the date herein, are for informational purposes only, and Acme Packet has no obligation to provide any future releases or upgrades or any features, enhancements or functions, unless specifically agreed to in writing by both parties.

Welcome to TechTalk!

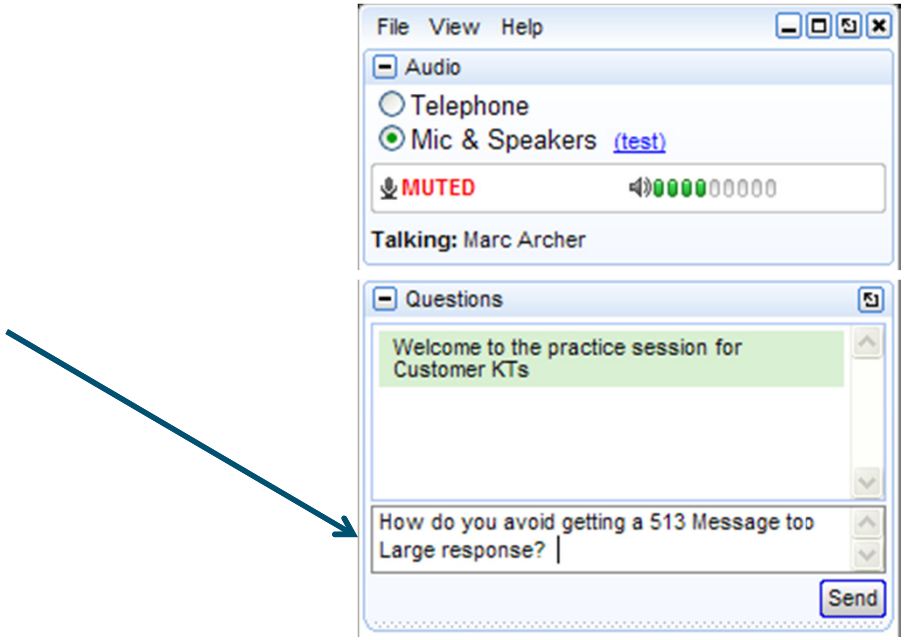
- Each webinar in our series will offer a blend of the following:
  - Feature education;
  - Configuration best practices;
  - Troubleshooting guidance; and
  - Log analysis
- Sessions will be very technical in nature, and targeted towards an audience of system operators and network engineers
- Our goal today is to pass along field experience with signaling protocols, interoperability requirements, and network topologies to you!

Logistics

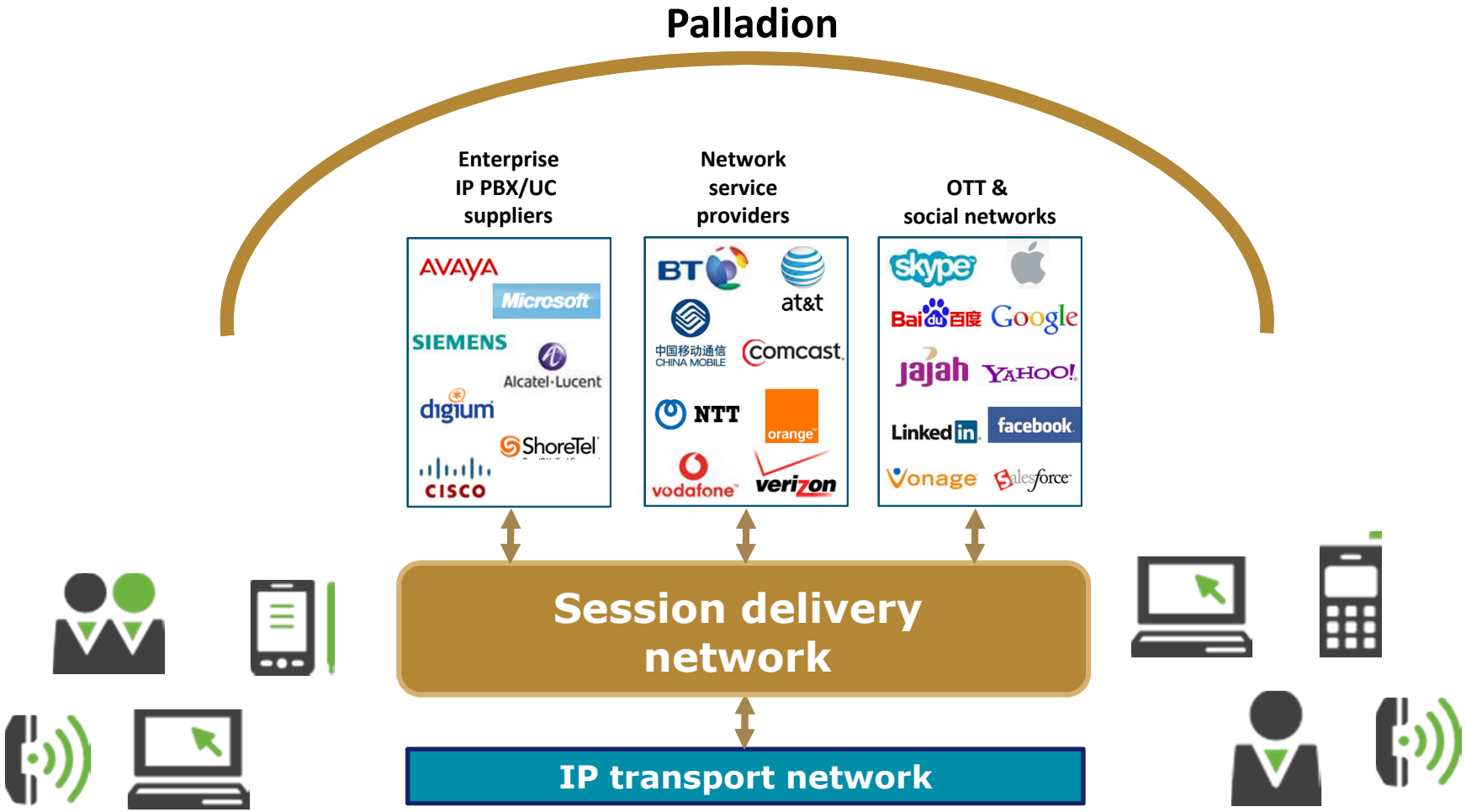
- To preserve the user experience, we will mute all participant phone lines
- Expect 50 minutes of presentation followed by 10 minutes of Q&A
- To submit a question, please use the GoToWebinar "Questions/Chat" window
  - You can access this by clicking on "Show Control Panel" on the right side of your screen
  - At the end of our presentation, we will answer as many questions as possible
- An evaluation form will be emailed to you after the webinar. We appreciate your feedback!

Questions

- Use the *GoToWebinar* "Questions/Chat" window to ask the moderator a question… Time permitting, it will be addressed



Session Delivery Networks for end-to-end IP communications



Session Delivery Network solutions encompass eight product categories

Session border controller	Controls sessions at borders
Session manager	Manages subscriber access & interfaces to application servers (A-SBC + IMS CSCFs + BGCF)
Multiservice security gateway	Secures session delivery (data & voice) over untrusted networks
Diameter session controller	Enables LTE data & voice session roaming
Session-aware load balancer	Scales session control at borders (SBC, MSG & PEC)
Session routing proxy	routes sessions to/from access & interconnect borders
Application session controller	Empowers Web 2.0 applications to control sessions
Session recorder	Provides session recording utility for SDN

Introduction

Agenda – Part 1 (January 16th)

- HMR fundamentals
- Working with header elements
- Using built-in variables
- A mini primer on regular expressions
- HMR strategies and best practices
- Two sample HMRs of varying complexity
- Q&A

Agenda – Part 2 (February 13th)

- Recap of HMR Part 1
- HMR operations
- HMR for SDP
- Testing your HMRs using SIPp
- Testing your HMRs using test-sip-manipulation
- Three sample HMRs of varying complexity
- Q&A

HMR fundamentals

What is "HMR"?

- HMR originally stood for Header Manipulation Rules, but today it's far more than that
- HMR provides the ability to manipulate SIP, through configuration
- Manipulate: add, delete, replace, copy, move, reject, log
- Manipulate what?
  - SIP headers, parameters, URIs, parts of URIs
  - SIP MIME bodies: SDP, XML, ISUP, *anything*
  - SIP-I/SIP-T ISUP messages, parameters, and fields
- And more…
  - Copy from SIP into CDR fields
  - Route messages based on any SIP field
  - Reject requests based on any SIP field
  - Change response code numbers based on specific fields

Note: one item HMR cannot change is the SIP method itself, e.g. HMR cannot change an OPTION to an INVITE. Use Acme Packet SPL for that!

Why use HMR?

- The problem:
  - There are numerous proprietary implementations of SIP
  - There are numerous "bugs" in SIP implementations
  - The SBC can't natively support interworking every software version of every product of every vendor to every other one

- The solution:
  - Give the operator as much control as possible, to define their own behavior for their needs
  - Without constantly upgrading the SBC
  - Without constant Enhancement Requests
  - Without waiting

HMR in the SBC

- HMR can be done on either the inbound message (in-manipulationid) or the outbound message (out-manipulationid)
  - In both cases the SBC behaves as if it actually received that changed message, or sent the unchanged one
- Inbound:
  - HMR is essentially a pre-processing activity – it occurs after a very small list of tasks (like DDoS, parsing, etc.)
- Outbound:
  - HMR is essentially a post-processing activity and is done right before message is sent
  - After SIP-NAT, number normalization, response mapping, etc.
- HMR rules can be applied to session-agent, realm, or sip-interface for either direction

The sip-manipulation object

- Only one sip-manipulation gets applied to a message on inbound, and only one on outbound
  - sip-manipulations can be called from a session-agent, realm or sip-interface
  - A session-agent manipulation overrides all others, a realm overrides sip-interface
  - For a proxied SIP message there can be up to 12 "configuration slots" where you can invoke HMR but for a B2BUA, only FOUR HMRs can take effect (in/out on received message, and in/out on proxied message)

sip-manipulation contents

```
sip-manipulation
    name                    changeMyName
    description             change James to Jimmy
    header-rule
        name                    modifyRequestURI
        header-name             request-uri
        action                  manipulate
        comparison-type         case-sensitive
        match-value
        msg-type                request
        new-value
        methods                 INVITE
        element-rule
            name                    modName
            parameter-name
            type                    uri-user
            action                  replace
            match-val-type          any
            comparison-type         pattern-rule
            match-value             James
            new-value               Jimmy
```

This **HEADER RULE** will operate on the request URI. It will trigger the sub-rule modName each time the message being manipulated is an INVITE request.

Note that header-rule and element-rule names cannot be all upper case… That's reserved for internal variables.

This **ELEMENT RULE** will operate on the user portion of the request URI. It will replace regular expression matches of the string "James" with the string "Jimmy".

header-rules and element-rules

- A sip-manipulation can have any number of header-rules
  - Each header-rule name must be unique in a sip-manipulation and limited to a maximum of 24 characters
  - Each header-rule operates on one header, but multiple header-rules can operate on the same header

- A header-rule can have any number of element-rules
  - Each element-rule name must be unique within a header-rule and limited to a maximum of 24 characters
  - Each element-rule operates on one component of the header, but multiple element-rules can operate on the same component

How many header-rules and element-rules do I need?

- Each header-rule permits one "action" to be conducted against one single header
  - For each required header action you will need one header-rule
  - Recall that a header-rule can reference other header-rules within the same sip-manipulation rule set
  - A header-rule can also call another sip-manipulation outside of the current rule set
    - set the header-rule action to be *sip-manip*
    - *new-value* is used to hold the name of the sip-manipulation to be invoked
    - Don't forget to specify a header name to invoke the header-rule! We recommend the use of "To", "From" or "Cseq" as these are most likely exist in the sip header context

- Each element-rule permits one "action" to be conducted against one single header element
  - For each required header element action you will need one element-rule

Example 1: very simple case

- You want to remove a header, for example P-Asserted-Identity, from all messages going to endpoints in the access realm
- You can do this on inbound or outbound, but for this example, outbound makes more sense
- Example HMR:

```
sip-manipulation
    name                out_rem_PAI
    Description         "This HMR deletes the PAI header"
    header-rule
        name                    delPAI
        header-name             P-Asserted-Identity
        action                  delete
        comparison-type         case-sensitive
        match-value
        msg-type                request
        new-value
        methods
```

Example 2: a bit more complex

- Removing a P-Associated-URI from any response to a REGISTER request, but only if it's a tel-URI
  - Here we will need a comparison-type of pattern-rule to trigger the regular expression specified in the match-value

```
sip-manipulation

    name            delPAUwithTelURI
    description     "This HMR deletes PAU headers with a tel-URI"
    header-rule
        name                delPAU
        header-name         P-Associated-URI
        action              delete
        comparison-type     pattern-rule
        match-value         ^<tel:
        msg-type            reply
        new-value
        methods             REGISTER
```

Example 3: HMR, the real deal

```
sip-manipulation
     name                copyFrom
     description
     header-rule
          name                getFrom
          header-name         From
          action              store
          comparison-type     pattern-rule
          match-value
          msg-type            request
          new-value
          methods             INVITE
          element-rule
               name                getUser
               parameter-name
               type                uri-user
               action              store
               match-val-type      any
               comparison-type     pattern-rule
               match-value         ^\+?[0-9]+
               new-value
```

```
header-rule
     name                modPAI
     header-name         P-Asserted-Identity
     action              manipulate
     comparison-type     case-sensitive
     match-value
     msg-type            request
     new-value
     methods             INVITE
     element-rule
        name              modName
        parameter-name
        type              uri-user
        action            replace
        match-val-type    any
        comparison-type   boolean
        match-value       $getFrom.$getUser
        new-value         $getFrom.$getUser.$0
```

Can you figure out what this HMR does?
We'll see it again later in the presentation!

Working with header elements

Conditional execution of rules

- Every *top-level* rule (header-rule, mime-rule) will only execute its action if it passes the following constraints:
  - msg-type: any, out-of-dialog, or response
  - methods: if it's the right SIP method
- *Every* rule of *any* level has certain constraints it must pass to execute its action
  - match-value: if its match-value results in TRUE, based on its comparison-type and applied to the relevant SIP message/field
- Specific rules have specific constraints as well
  - header-rule will only act on the defined SIP header
  - element-rule will only act on the defined "type"
  - mime-rule will only act on the defined MIME body type

Sub-rules

- Every top-level rule has zero or more sub-rules
  - header-rule can have element-rules
  - mime-rule can have mime-header-rules

- Sub-rules will be executed, based on certain conditions
  - Only if the top-level rule's msg-type/methods matched, AND:
    - If the top-level rule's action is add, store, find-replace-all, or sip-manip OR
    - If the top-level rule's action is manipulate and its match-value was TRUE
  - Sub-rules will not execute if the top-level rule's action was delete or none

- Every sub-rule executes in provisioned order, even if one sub-rule matches FALSE (does not execute)

CHEAT SHEET

Rule actions

- store – the item matched by match-value is saved for later reference (empty match-value means match everything)
- manipulate/replace – the item is replaced with new-value
- add – the item is added, with the contents of new-value
- delete – the item is deleted
- find-replace-all – the match-value is a regex applied against the item and replaced with the new-value, then again from the end of the previously matched spot in item
- sip-manip – run the sip-manipulation identified in new-value
- reject – if it's a SIP request, it's rejected with the response-code in the new-value
- log – certain portions of the SIP message are logged in matched.log
- none – rule not executed, nor are any sub-rules executed

What's the "item"?

- For a header-rule, it's the whole header value
  - The 'header-name' field defines the specific Header
- For an element-rule, its defined by the 'type' field
  - Valid 'types' are: header-value, header-param-name, header-param, uri-display, uri-user, uri-user-only, uri-phone-number-only, uri-user-param, uri-host, uri-port, uri-param-name, uri-param, uri-header-name, uri-header, status-code, reason-phrase, mime
- For a mime-rule, it's a MIME body part
  - The 'content-type' field defines the specific body
  - *We'll explain this more in Part 2 of this TechTalk*

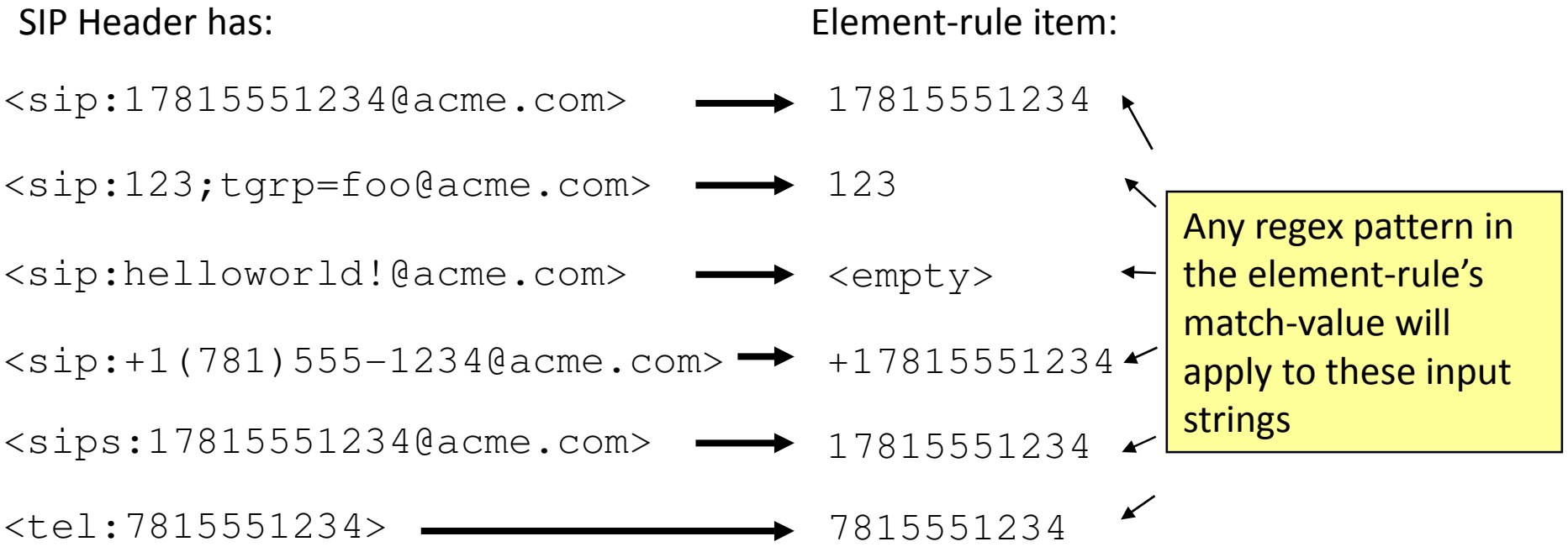
CHEAT SHEET

Example “items”

```
To: ”John Smith” <sip:+1(781)328-4400;rn=123@acmepacket.com;user=phone>;foo=bar
```

- The header-name is ‘To’
- The uri-display is ‘John Smith’
- The uri-user is ‘+1(781)328-4400;rn=123’
- The uri-user-only is ‘+1(781)328-4400’
- The uri-phone-number-only is ‘+17813284400’
- The ‘rn=123’ is a uri-user-param (‘rn’ is the param-type, ‘123’ is the value)
- The uri-host is ‘acmepacket.com’, the uri-port is ‘5060’
- The ‘user=phone’ is a uri-param (‘user’ is the param-type, and also a uri-param-name)
- The ‘foo=bar’ is a header-param (‘foo’ is the param-type, and also a header-param-name)

Uri-phone-number-only



- Using an element-rule, the *item* being worked on is constrained/filtered to be based on the 'type' configured
  - In this case of uri-phone-number-only, it's the digits, with/without the leading '+'
  - If it's not digits, it's as if the *item* didn't exist, so the action won't take place

The match-value field

- If the other constraints match (SIP message type, method name, parameter exists, etc.), the match-value is checked based on the configured 'comparison-type'
  - case-sensitive and case-insensitive compare the *string* in match-value against the *item*, using string comparison
  - refer-case-sensitive and refer-case-insensitive compare the *referenced* value in match-value against the *item*, using string compare
  - pattern-rule treats the match-value field as a regular expression, and performs regex comparison of it against the *item*
  - boolean treats the match-value as a boolean expression, and resolves it to TRUE or FALSE (i.e., not really related to the *item*)
- If the result of the match-value resolution is TRUE, then the action is performed
- If the match-value was a regex (and the comparison-type was pattern-rule), then it also stores the matches for later referencing

Dereferencing variables

- All references start with a '$' character
- References can refer to strings stored by previous Rules
  - The reference is defined by the *name* field of the Rule by which it was stored, e.g. $myHeaderRule.$0
  - If referencing sub-rules, the top-level Rule name must be noted first, e.g. $myHdrRule.$myElemRule.$0
- A reference ending with a $<number> resolves to the stored string or sub-string matched by the regex
  - $0 is the string matched by the whole regex pattern
  - $1 is the first parenthesis sub-group, $2 is the next, etc.
- A reference of just the $name resolves TRUE or FALSE for whether it matched or not (i.e., did it store anything)

Example references

- Assume the above is stored in an element-rule named "getPhone", of a header-rule named "getPAI"
  - $getPAI.$getPhone will resolve to TRUE
  - $getPAI.$getPhone.$0 will resolve to '17815551234'
  - $getPAI.$getPhone.$1 will resolve to '781'
  - $getPAI.$getPhone.$2 will resolve to '5551234'

CHEAT SHEET

Boolean operators

- New string-logic operators for use in match-value boolean logic
  - "==" case-sensitive string comparison
    - Resolves to TRUE if left and right strings are the same
    - Example: `match-value $getVal1.$0 == $getVal2.$0`
  - "~="case-insensitive string comparison
  - "!=" case-sensitive inequality (TRUE if left is not same as right)
- New integer-logic operators for match-value
  - "<=" less than or equal to, left-hand integer is less than right side
  - ">=" greater than or equal to operator
  - "<" less-than operator
  - ">" greater-than operator
  - Example: `match-value $getVal1.$0 >= $getVal2.$0`

The new-value field

- For most actions, the new-value field is used as an expression, which once resolved, results in a string to replace/add
  - The new-value is not a plain string – it's an expression
  - For sip-manip action, the new-value is used to hold the name of the sip-manipulation to be invoked
  - For reject action, the new-value field defines the response code, and optionally the reason-phrase, to use for the response

- Special command characters are used in new-value field
  - '+' is used to append/concatenate, '^+' is for prepend
  - '-' is used to truncate/cut, and '^-' is used to truncate from the front

```
new-value <sip:12345@acmepacket.com>
new-value <sip:+$FROM_PHONE.$0+@acmepacket.com>
new-value $ORIGINAL^+\+   (which is the same as "\++$ORIGINAL")
```

CHEAT SHEET

Nested HMRs

```
sip-manipulation
     name                          parent_HMR
     description
     header-rule
          name                     nested_Nat_IP
          header-name              From
          action                   sip-manip
          comparison-type          case-sensitive
          match-value
          msg-type                 any
          new-value                Nat_IP
          methods
last-modified-by                   admin@10.23.0.38
last-modified-date                 2008-08-06 10:18:08
```

You must put a valid header name here to trigger a match

Tells the SBC to call another HMR

Specify the name of your "child" HMR – in this case, our old pal Nat_IP

Using built in variables

CHEAT SHEET

Referencing built in variables

- HMR offers many system-defined "built in" variables

LOCAL_IP	TO_USER	PAI_USER
LOCAL_PORT	TO_PHONE	**PAI_PHONE**
REMOTE_IP	TO_HOST	PAI_HOST
REMOTE_PORT	TO_PORT	PAI_PORT
REMOTE_VIA_HOST	FROM_USER	PPI_USER
TRUNK_GROUP	FROM_PHONE	PPI_PHONE
TRUNK_GROUP_CONTEXT	FROM_HOST	PPI_HOST
MANIP_STRING	FROM_PORT	PPI_PORT
MANIP_PATTERN	CONTACT_USER	PCPID_USER
CRLF	CONTACT_PHONE	PCPID_PHONE
ORIGINAL	CONTACT_HOST	PCPID_HOST
REPLY_IP	CONTACT_PORT	PCPID_PORT
REPLY_PORT	RURI_USER	CALL_ID
TARGET_IP	RURI_PHONE	TIMESTAMP_UTC
TARGET_PORT	RURI_HOST	T_GROUP
M_STRING	RURI_PORT	T_CONTEXT

```
match-value $PAI_PHONE.$0==17813284428
match-value !$PAI_PHONE
new-value <sip:+$RURI_PHONE.$0+@+$REMOTE_IP.$0>
```

Note the $0 notation is to dereference that variable

Remember example 3?

```
sip-manipulation                                header-rule
     name              copyFrom                       name              modPAI
     description                                      header-name       P-Asserted-Identity
     header-rule                                      action            manipulate
          name              getFrom                   comparison-type   case-sensitive
          header-name       From                      match-value
          action            store                     msg-type          request
          comparison-type   pattern-rule              new-value
          match-value                                 methods           INVITE
          msg-type          request                   element-rule
          new-value                                      name              modName
          methods           INVITE                       parameter-name
          element-rule                                   type              uri-user
               name              getUser                 action            replace
               parameter-name                            match-val-type    any
               type              uri-user                comparison-type   boolean
               action            store                   match-value       $getFrom.$getUser
               match-val-type    any                     new-value         $getFrom.$getUser.$0
               comparison-type   pattern-rule
               match-value       ^\+?[0-9]+
               new-value
```

Example 3 reduced

```
header-rule
      name                  modPAI
      header-name           P-Asserted-Identity
      action                manipulate
      comparison-type       case-sensitive
      match-value
      msg-type              request
      new-value
      methods               INVITE
      element-rule
          name               modName
          parameter-name
          type               uri-user
          action             replace
          match-val-type     any
          comparison-type    boolean
          match-value        $FROM_PHONE
          new-value          $FROM_PHONE.$0
```

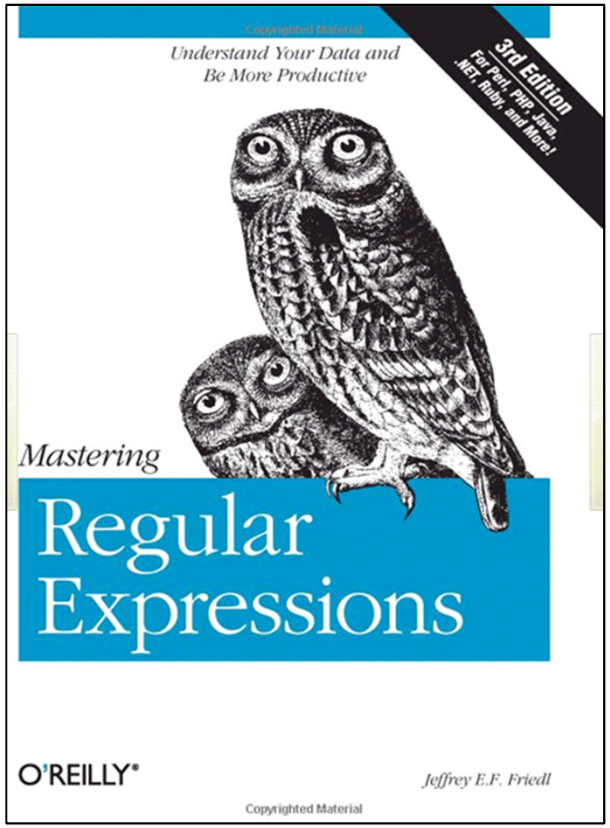
Now one header-rule can do what two rules did before!

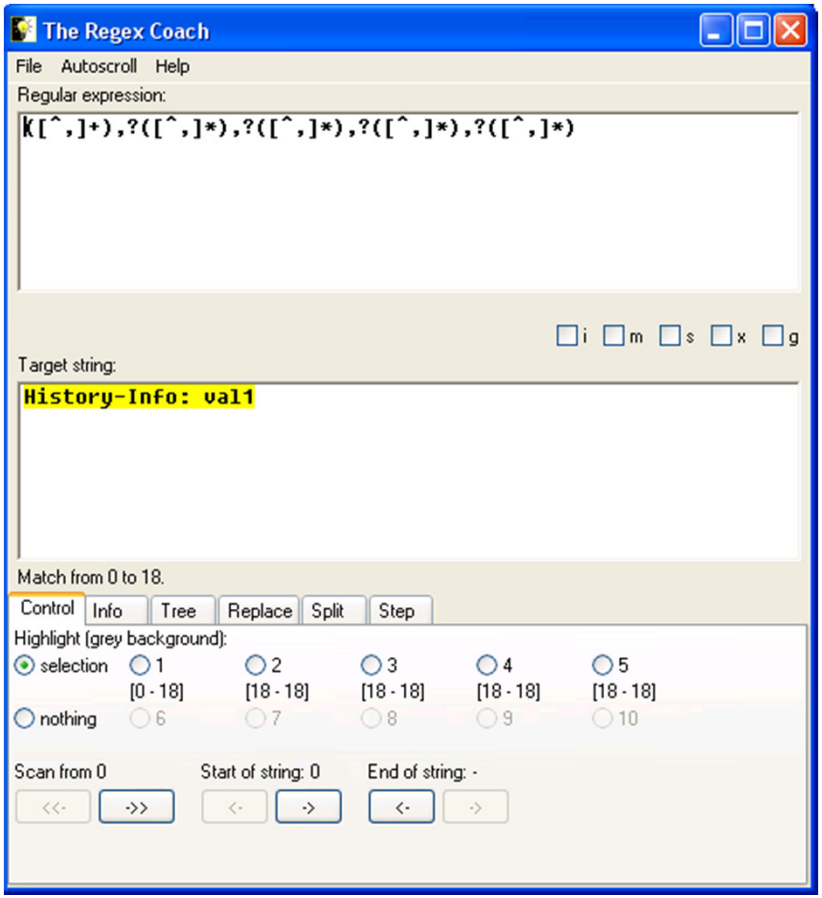
- Easier to provision
- Performs faster
- Easier to read

A mini primer on regular expressions

Regular Expressions

- Best book: *Mastering Regular Expressions* by Jeffrey Friedl
- Best tools: The Regex Coach and RegexBuddy (w/ tutorial)
- Best command: test-pattern-rule





Regex basics

- A regex engine starts at the first character of the regex pattern, and before the first character of the input string
- It walks across the string until the first command of the regex pattern is matched/true
- Then it moves to the next command in the pattern, and tries to match it to the current spot in the string – a spot can be between actual characters in the string
- This continues, until it can no longer match – if there is any (non-optional) pattern left, it failed (extra string doesn't matter)

CHEAT SHEET

Regex character commands

- Any character (letter, number, whatever) *except* "special characters" means match that literal character

- Special characters:

  . = match any character except carriage-return or linefeed

  ^ = match the *spot* of the beginning of a string

  $ = match the *spot* of the end of the string

  \ = escapes the next character, possibly making it a special/command, or making a special character no longer "special"

  + = match the previous command *one* or more times, as much as possible

  * = match the previous command *zero* or more times, as much as possible

  ? = match the previous command *zero* or *one* times, prefer *one*

  | = match the whole pattern on the left, or the one on the right of the bar

  [= starts a bracket "character class"

  (= starts a parenthesis group

  { = starts a brace min/max repetition count

Regex backslash commands

- When a "\" backslash is before a "special character", it makes it no longer special, but just literal (e.g., "\." matches a period)
- When a backslash is before a backslash, it makes it match a literal "\" backslash in the input string
- When a backslash is before a normal character, it means another command:

  \, = match any character, including carriage-return or linefeed

  \b = match the spot between a word character, and a non-word one

  \B = match the spot between two word characters, or between two non-word characters

  \d, \w, \s = match a digit, word character, whitespace respectively

  \D, \W, \S = match the opposite of the characters of \d, \w, \s

  \r, \n, \f, \t, \v = match carriage-return, newline, form-feed, h-tab, v-tab

  \R = match either \r or \n or both together \r\n

Regex character classes

- A bracket pair "[…]" encompass a character class
- This can take on of two forms: either a list of characters, possibly including ranges, to match *one* character in the string to, or the negation of it if "[^…]"

BRACKET PAIR:

`^555[12].*`

`5551234`

`5552234`

`5553234`

NEGATION:

`^555[^12].*`

`5551234`

`5552234`

`5553234`

Braced min/max repetitions

- A "{*n*,*m*}" is a command which means match the previous command a minimum of *n* times, to a maximum of *m* times, preferably more
  - *m* can be left blank, to mean the max is infinite
- A "?" is basically the common case of "{0,1}"
- A "*" would be the same as "{0,}" and a "+" would be "{1,}"

Breaking down a real-world example

```
element-rule
     name                    getDigits
     parameter-name
     type                    uri-user
     action                  store
     match-val-type          any
     comparison-type         pattern-rule
     match-value             ^\+1([0-9]{6}).*$
     new-value               $1
```

- ^ => start of string
- \+ => delimiter for the “plus” character
- [0-9] => any character in the specified range, meaning 0123456789
- {6}=> six characters from that range, so [0-9]{6} could be anything from 000000 to 999999
- .* => any sequence of characters
- $ => end of string
- Parentheses store the six digit string in the variable $1

CHEAT SHEET

…and accessing the values

```
element-rule
     name                    getDigits
     parameter-name
     type                    uri-user
     action                  store
     match-val-type          any
     comparison-type         pattern-rule
     match-value             ^\+1([0-9]{6}).*$
     new-value               $1
```

- To access the digits stored by this element-rule, I need to dereference what is in the parentheses
  - $HR.$ER.$0 => this is the ENTIRE string that you're matching against, and this is how you would dereference the value to use it ELSEWHERE in your HMR
  - $0 => when used as the new-value for a given match-value, this is the ENTIRE string that you're matching against in that corresponding match-value
  - $1 => when used as the new-value for a given match-value, this is the value stored in the FIRST set of parentheses; to access it ELSEWHERE within the HMR you would do a $HR.$ER.$1
  - $2 => when used as the new-value for a given match-value, this is the value stored in the SECOND set of parentheses

HMR strategies and best practices

When authoring HMRs…

- Often times, there are different ways to tackle HMR requirements – and some are better than others
- A good rule of thumb is to think "performance first"
  - Reuse as many "built in" variables as possible
  - Avoid lengthy string matches unless absolutely necessary
  - Wherever possible, constrain your HMR appropriately by specifying a SIP method and message type
- Practice and document!
  - Over time, an HMR library proves to be invaluable

Tips on using regex patterns

- ALWAYS use a ^ beginning of line anchor if you can
- ALWAYS use a $ end of line anchor if you can
- Do not use parentheses unless you need to
- Try to avoid .* or .+ if you expect to have text following it
- Sometimes, the NOT operator e.g. [^abc] can give you the easiest access to the string you're looking to retrieve
- Avoid vague search patterns like (.*)foo(.*) or (foo.*bar)*
- Think of other possible combinations for your match
  - If you are looking for a particular string it could be at the beginning, the middle, or the end of a line. You have to plan for all cases!
- Test your pattern against things that do NOT match, or that you don't want to match

Tips on HMR for performance and security

- Use element-rules to store/modify/add/delete the specific SIP header pieces you want instead of just using a header-rule

- Reduce the number of header-rules as much as possible

- Try to make your rules generic and future-proof
  - Avoid creating sip-manips for each realm/agent
  - Don’t forget sips and tel URIs
  - Not all URI usernames are numbers, and some have user-params
  - REGISTERs can have more than one Contact header
  - Phone numbers can have visual-separators: +1(212)555-1212 => uri-phone-number-only

Use caution with regular expressions

- Be very careful with MIME manipulation, e.g. SDP
  - They're big strings, so regex MUST be efficient
  - SDP is a *protocol*, and has rules – don't break them
  - SDP offers are NOT always in INVITEs – for example, delayed offers come back in 18x/200ok, and answers can be in PRACK or ACK
- Be mindful of RegEx performance
  - Example: (.*)foo(.*) is really not efficient, though it may be the only solution
  - Often the worse-case performance is when your regex does NOT match

Two sample HMRs of varying complexity

Two HMRs of varying complexity

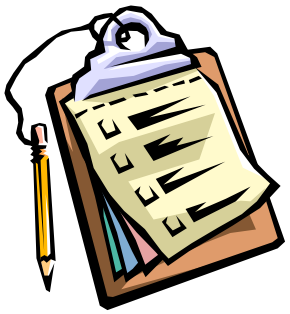
- Beginner: HMR to strip leading plus from Diversion
- Intermediate: HMR to add a cn RURI param containing charge number
- *Expert: two part HMR to turn Recvonly into Sendonly in transcoding scenarios*
  - *We'll save this one for the "HMR – Part 2" TechTalk*

Q&A

Summary – Part 1

Key takeaways

- Make sure you need to use HMR in the first place!
- Think “performance first”
  - Avoid string matches if at all possible
  - Use built in variables as much as possible
- Regular expressions are not as scary as you might think… You just need to practice!
- HMR is not as scary as you might think… You just need to practice!

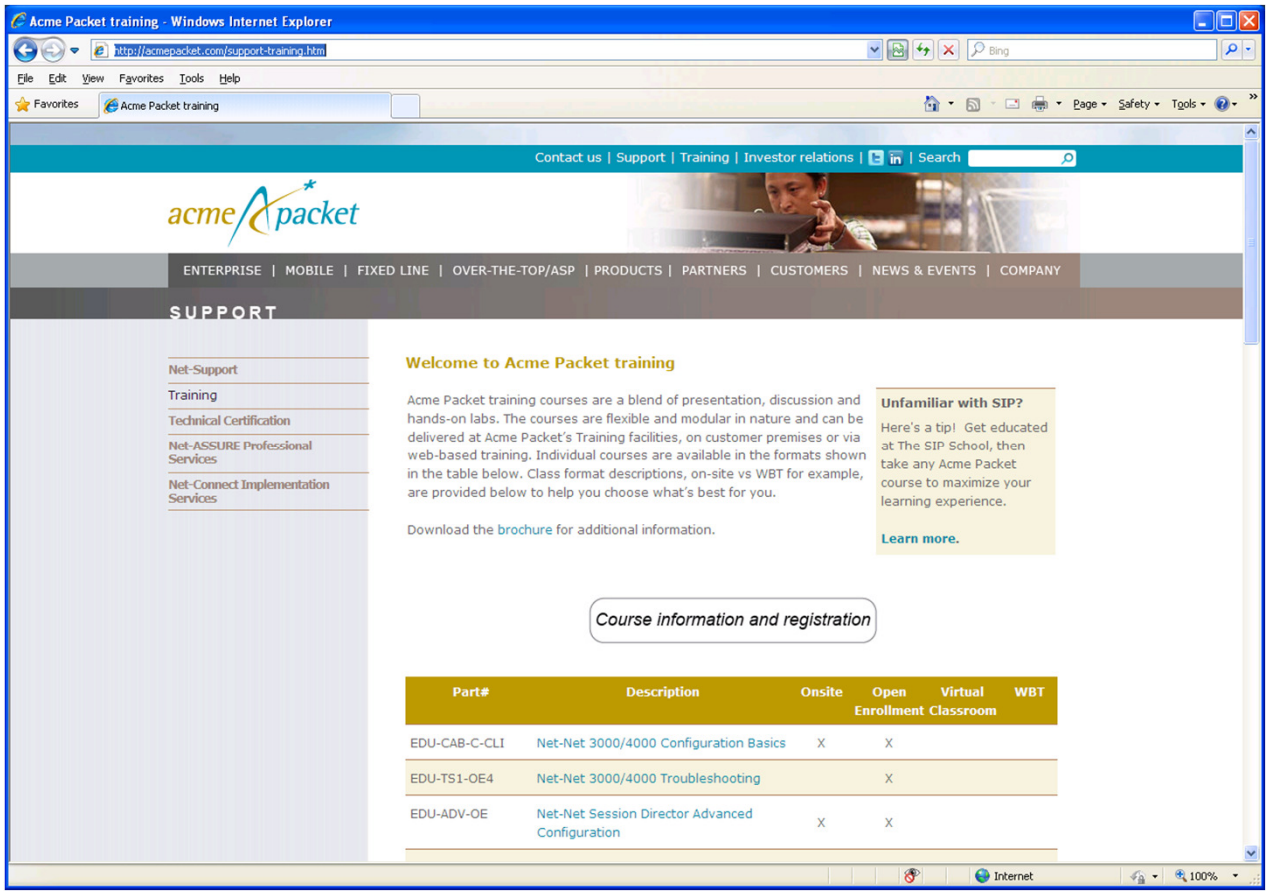


For more information

- Acme Packet documentation
  - Acme Packet HMR Developer's Guide
  - Net-Net Session Director Configuration Guide
  - Net-Net Session Director ACLI Reference Guide
- Acme Packet Best Current Practices (BCP) documents available on the Acme Packet Support Portal
- The Acme Packet Community
- Acme Packet training
- Direct Acme Packet support

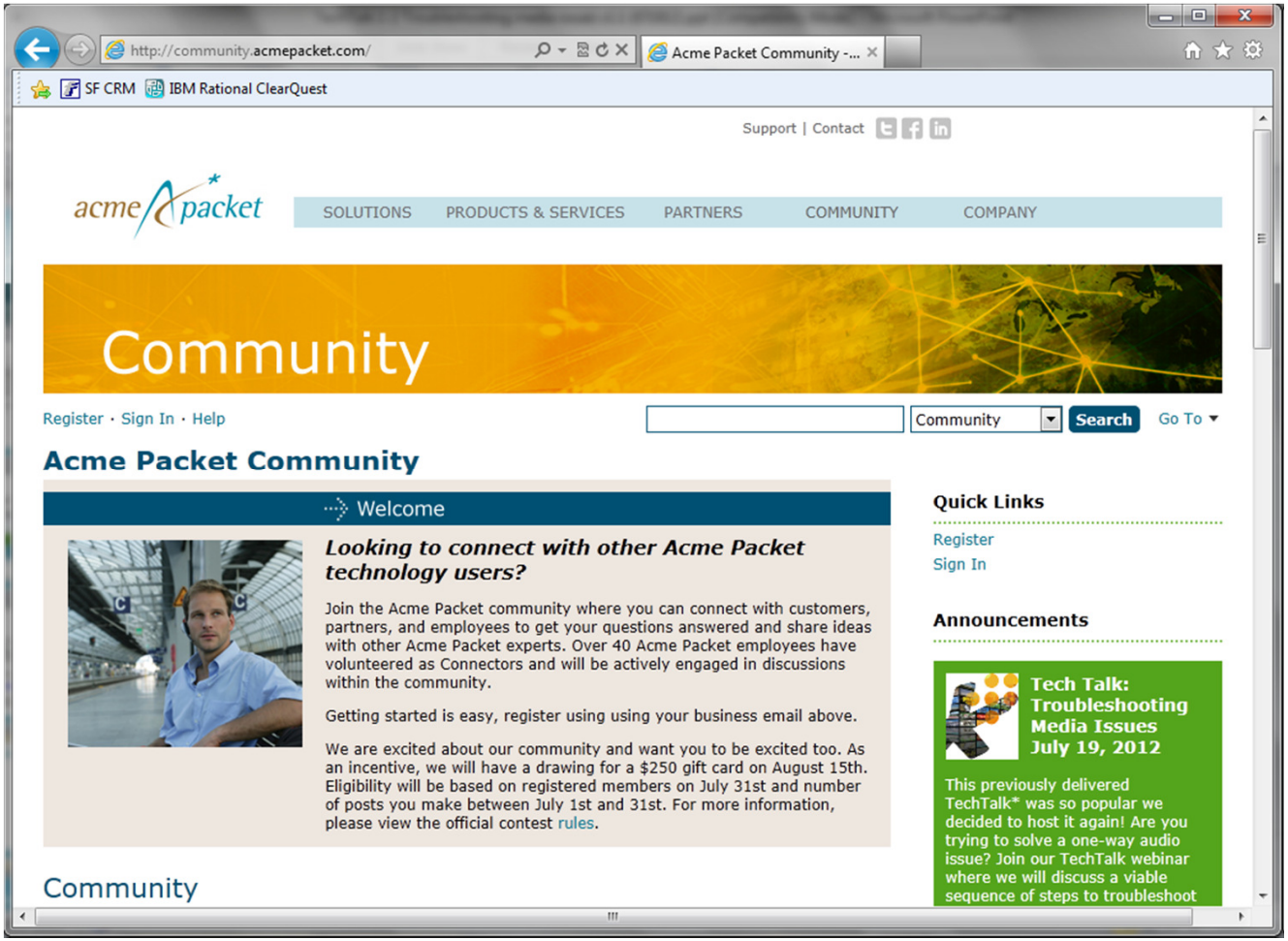
And remember…

- Formal Acme Packet training is available in the area of HMR and other topics



Join the Acme Packet Community

- Come get connected! Post questions – and answer some too!



Thank you for attending!

Our next TechTalk Webinar:

HMR Part 2 – Advanced Concepts

13 February 2013