TeleScope CQ Query Engine Language Specification

Kirill Belyaev, Fort Collins, CO, USA, 80523

May 23, 2012

1 The tutorial based on BGP (Border Gateway Protocol) routing data analysis

Current TeleScope CQ implementation employs XML parsing and specific pattern-matching that provides standard logical operator constructs to combine the XML attributes into a pattern set that is applied to the XML message packed in the XML format on the fly as it is received from the network. The whole pattern-matching expression starts with the -e (expression) command line option and is enclosed within the double quotes (“” ””). The transaction could then be altered or reset via the CLI interface through connecting to port 50005 of the running TeleScope CQ instance. The expression from the remote CLI interface is not delimited by ””. The example expression could be modeled as:

-e ”(MULTI_EXIT_DISC <10 & SRC_AS = 6447 & type = MESSAGE) | (ORIGIN = EGP & value = 0)”

The expression could be considered either:

- simple – involving only one subexpression and one type of logical operators
- complex – involving more then one subexpression and two types of logical operators - OR and AND (like the expression above)

The complex expression could be constructed out of a number (two and more) of simple subexpressions connected via a logical | OR operator and a set of parentheses.

Sample tables for simple and complex expressions are shown next.

Due to complexity of constructing regular expressions using -e command-line option we should follow the following guidelines:

- simple expression should NOT include ( ) parentheses: otherwise it would not be evaluated properly
- simple expression has only ONE type of logical operator - it is either OR | or AND &
- simple expression could have one or more tuples - see simple expression table

Table 1: Simple Expression

Example	Description
“”expr1””	1: should NOT have parentheses
“”””	2: empty expr
“”type = UPDATE””	1 tuple
“”ORIGIN = IGP””	1 tuple
”type = UPDATE & SRC_AS = 6447””	2 tuples
”MULTI_EXIT_DISC = 100 & SRC_PORT = 4321”	2 tuples
”type = UPDATE | DST_AS = 3200”	2 tuples
”type = MESSAGE | type = STATUS”	2 tuples
”type = UPDATE | type = MESSAGE | type = KEEPALIVE”	3 tuples
”MULTI_EXIT_DISC = 100 | SRC_AS = 6447 | SRC_PORT = 4321”	3 tuples

Table 2: Complex expression

Example: LCO is |	Description
“”(subexpr1) LCO (subexpr2)””	2 subexprs
”(type = STATUS) | (type = UPDATE)”	2 subexprs
“”(subexpr1) LCO (subexpr2) LCO (subexpr3)””	3 subexprs
”(type = STATUS) | (type = UPDATE) | (type = KEEPALIVE)”	3 subexprs

- complex expression must have more then one set of ( ) parentheses - see complex expression table
- | OR operator is the only operator used to chain subexpressions in complex expression involving multiple ( ) parentheses: (e1) | (e2)
- | OR operator could NOT be used inside the ( ) parentheses
- & AND operator could NOT be used to chain subexpressions in complex expression involving multiple ( ) parentheses
- & AND operator must be used inside the ( ) parentheses in complex expressions
- | and & could not be used within a single expression enclosed in ( ) parentheses or within a single simple expression

Here we provide the sample use cases of valid and invalid expressions:

- -e ”type = STATUS | type = UPDATE” - valid simple expression
- -e ”type = STATUS | type = UPDATE | type = KEEPALIVE” - valid simple expression
- -e ”type = UPDATE & SRC_AS = 6447” - valid simple expression

- -e "type = UPDATE" - valid simple expression
- -e "(type = STATUS) | (type = UPDATE)" - valid complex expression
- -e "(type = STATUS) | (type = UPDATE) | (type = KEEPALIVE)" - valid complex expression
- -e "(type = UPDATE)" - invalid complex expression - only 1 set of ( ) parentheses
- -e "(type = STATUS | type = UPDATE)" - invalid complex expression - only 1 set of ( ) parentheses
- -e "(type = UPDATE & SRC_AS = 6447)" - invalid complex expression - only 1 set of ( ) parentheses
- -e "(type = UPDATE) & (SRC_AS = 6447)" - invalid complex expression - & should not be used to chain the ( ) parentheses
- -e "(type = STATUS) | (type = UPDATE & MULTI_EXIT_DISC = 100) | (MULTI_EXIT_DISC = 10) | (type = KEEPALIVE & DST_AS >3200)" - valid complex expression
- -e "(type = STATUS) | (type = UPDATE & MULTI_EXIT_DISC = 100 & SRC_AS = 6447 & SRC_PORT = 4321 & ORIGIN = EGP) | (MULTI_EXIT_DISC = 10 & SRC_AS = 6447 & SRC_PORT = 4321 & ORIGIN = EGP) | (type = KEEPALIVE & DST_AS >3200)" - valid complex expression

To summarize the following two points should be taken into consideration when constructing complex expression:

- | operator is used OUTSIDE of ( ) parentheses: "(exp1) | (exp2) | (exp3)"
- & operator is used INSIDE of ( ) parentheses: "(tuple1 & tuple2) | (tuple3 & tuple4 & tuple5) | (tuple6 & tuple7)"

The TeleScope CQ Language operators are presented in the Table 3 with sample use cases.

Table 3: TeleScope CQ Language operators

Operator	Description	Example use
=	equality operator	ORIGIN = EGP
!	not-equal operator	SRC_AS ! 6447
<	relational less than operator	MULTI_EXIT_DISC <10
>	relational greater than operator	MULTI_EXIT_DISC >10
&	logical AND	ORIGIN = EGP & value = 0
|	logical OR	ORIGIN = EGP | value = 1
()	parentheses	(ORIGIN = EGP & value = 0) | (type = MESSAGE)

Parentheses are used for complex expression handling separating subexpressions within a single complex expression as shown in the last column of the table. The () parentheses are chained through logical OR operator.

TeleScope CQ provides a set of operators designed specifically for processing network prefixes including CIDR ranges. The operators follow the designated network prefix element (in our example it is the PREFIX attribute within the BGP XML Message) with the subsequent network range value. These are ordinary English letters having special meaning when used within the expression.

Table 4: Prefix operators

Operator	Description	Example use	Semantics
e	exact prefix match operator	PREFIX e 211.64.0.0/8	matching the networks wit the exactly defined networ prefix range.
l	less specific prefix match operator	PREFIX l 211.64.0.0/8	matching the networks wit less specific network pref range.
m	more specific prefix match operator	PREFIX m 211.64.0.0/8	matching the networks wit more specific network pref range.

Equality and negations operators (= and !) could be used in expressions involving both string and integer values and change the semantics of operation depending on operand type.

2 Example transaction management via the CLI interface

bash-3.2$ telnet hostname 50005

Connected to hostname. Enter password:

shell_:h

available commands are :

help (h); exit (q); show transaction (st); change transaction (ct); reset transaction (rt); shutdown (sd)

shell_:st

type ! NULL

shell_:ct

enter new transaction:

type = STATUS

shell_:st

type = STATUS

shell_: