YAHOO! FINANCE TechCrunch YAHOO! SPORTS HUFFPOSTI MAKERS YAHOO! NEWS engadget YAHOO! MAIL YAHOO! ENTERTAINMENT Aol. BUILD RYOT

Transaction Box

An Apache Traffic Server plugin

Alan M. Carroll, Verizon Media Edge

10 Sep 2019

verizon media

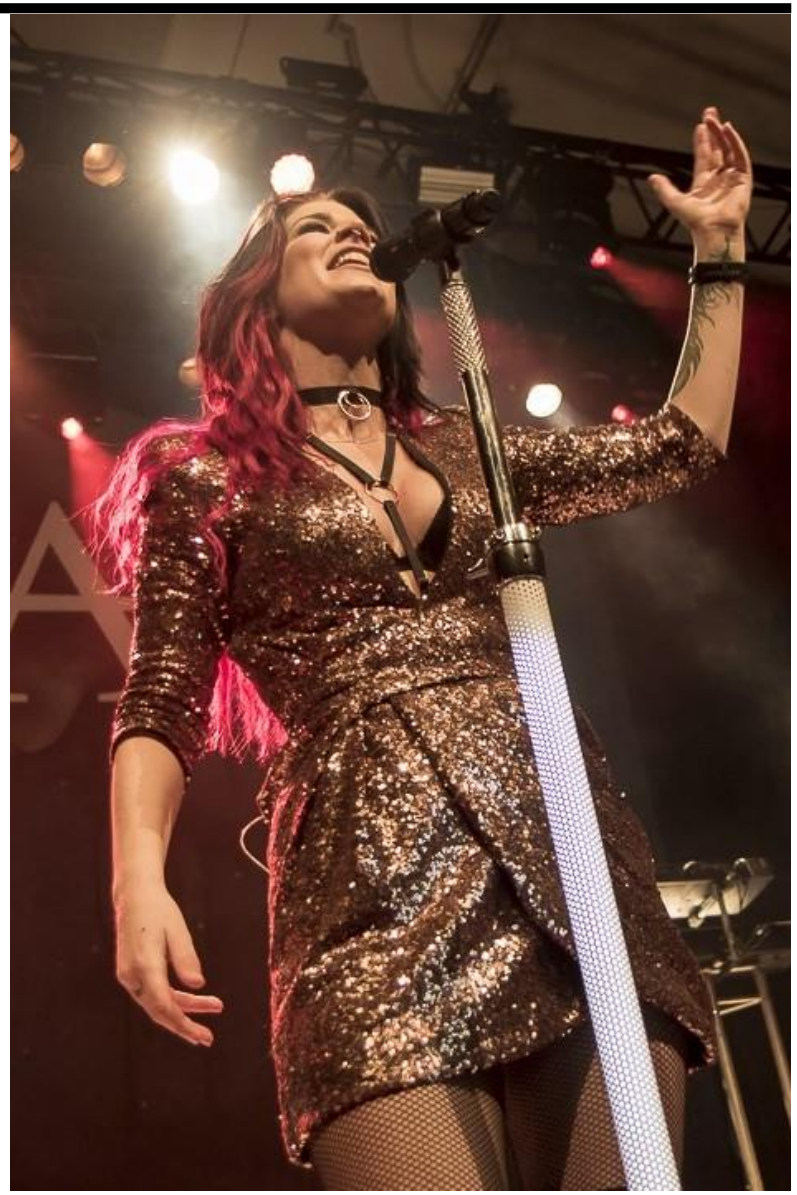
Introduction

Transaction Box (“TxnBox”) is a plugin for Apache Traffic Server.

It is intended as a general toolbox for manipulating transactions.

The key design goals are

- **YAML configuration.**
- **Rich set of proxy data access mechanisms.**
- **Extensive set of transaction manipulators.**
- **More specialized decision / comparison operators.**
- **Provide consistency between data for decisions and data for use.**
- **Easily extensible in syntax and implementation.**



verizon media

TxnBox Project

I expect TxnBox to undergo a number of iterations and changes from my original work. For this reason it has been done in a plugin rather than in the core.

This will enable

- Faster iteration.
- Easier testing and integration efforts.
- Possible to fork and experiment.
- Improvements in the C API.

The current version is sufficiently complete to achieve these, without being so far along change is excessively painful and expensive.

The implementation is currently starting production testing.

Git repo: https://github.com/SolidWallOfCode/txn_box

Documentation: http://docs.solidwallofcode.com/txn_box

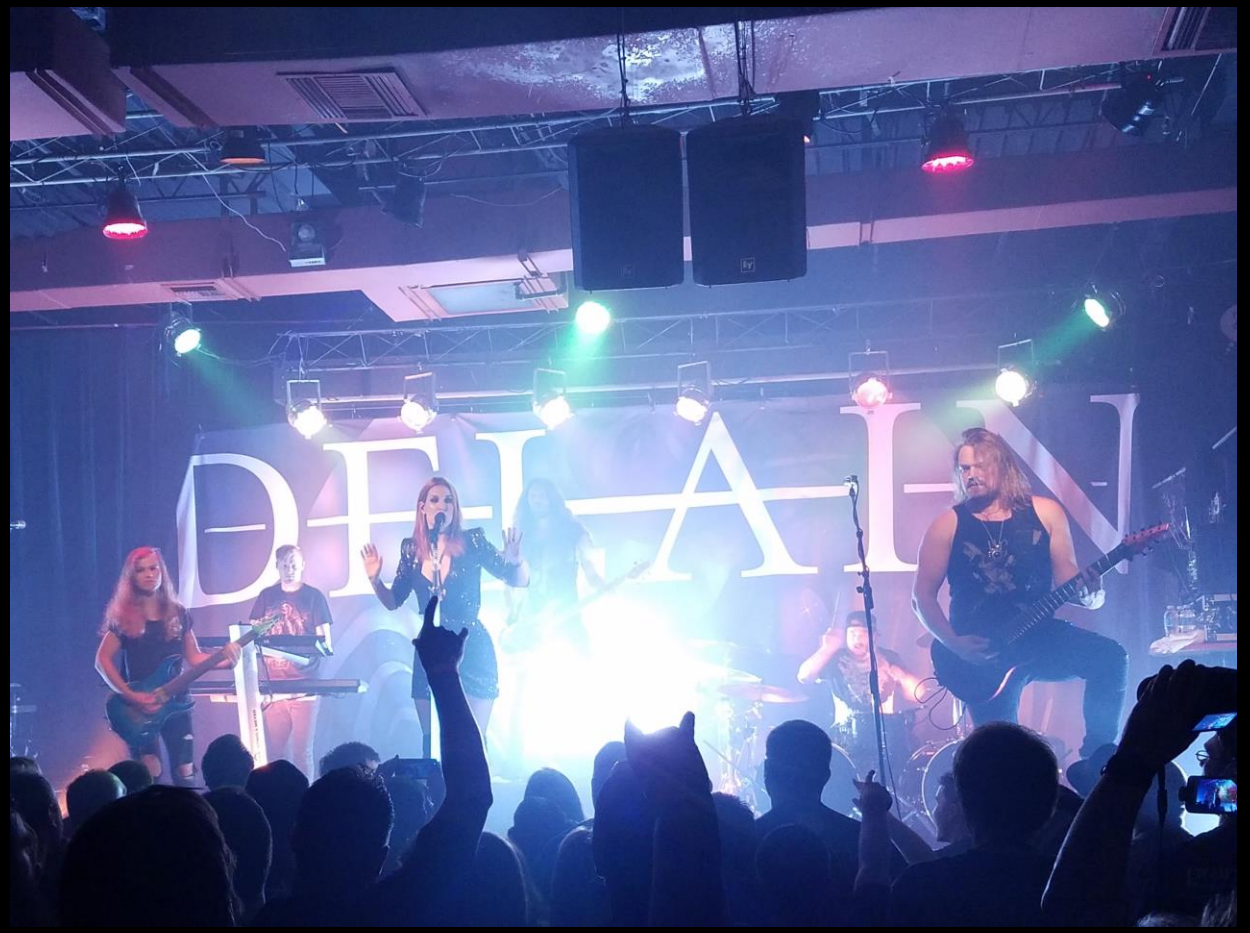
This slide deck: http://docs.solidwallofcode.com/asf/ApacheCon-NA-2019-TxnBox.pdf

verizon media

Motivations

From the dark currents of desire

to a light in the darkness



verizon media

Background

Traffic Server performs three fundamental functions -

Proxy

to intermediate between user agents and upstream hosts. Requests and responses can be changed, filtered, or duplicated.

Cache

for caching content for user agents to reduce burdens on upstream hosts.

Route

to decide to which upstream host to send a request.

The “remap.config” file is concerned primarily with routing, with the other functions added via plugins.

TxnBox is intended to provide a set of features that provide direct integration of proxy functions.

Request Routing

This is part of my overall effort to shift the focus of Traffic Server from routing *URL*s to routing *request*s. This is an explicit goal of the Layer 7 Routing project.

In modern CDNs, any part of the request can be part of the routing / modification decisions and Traffic Server should support that.

verizon media

URL Manipulation

A perennial complaint is the inability to move text from the host to the URL or vice versa in a remap. E.g. change

"yahoo.mail.com" to "yahoo.com/mail"

or the opposite

"apache.org/trafficserver" to "trafficserver.apache.org"

The need here is twofold –

- **Find the text in the header.**
- **Move that text to a new location.**

Current "remap.config" capabilities are quite limited. TxnBox will support this for any part of the header.

Regular expression matching

Current “remap.config” limits the regular expression matching, to such an extent that many sites use the “regex_remap” plugin to compensate.

TxnBox will support applying regular expressions to any part of the HTTP header for either routing or header manipulation.

verizon media

Access Controls

- **Lack of fine grained access control for rules.**

  The current ACL support is very limited, and generally can't handle more than one specification (see issue #1971).

- **Lack of access control fail over.**

  For current "remap.config", if a rule matches no other rule can match. This makes failing over to an alternative page if the ACL check fails impossible.

- **Limited set of data that can be checked for ACL enablement.**

  For instance, not able to check values in a client certificate, or cookie tokens.

verizon media

Obscure Syntax

The syntax of “remap.config” and supporting plugins is another chronic pain point.

Obscure and unique syntax which is

- **awkward to use and document.**
- **difficult to automate.**
- **nearly impossible to extend.**
- **different for every tool.**
- **no validation**

By using YAML and covering more functionality, TxnBox can ameliorate these issues.

It is a major goal to provide very specific error messages on failure, not just “failed to load remap.config”.

Consolidation

Much of what is considered standard request manipulation is actual done via plugins (e.g. “conf_remap”), each of which tends to have its own configuration and syntax. Other configuration, such as “hosting.config” duplicates the URL selection of “remap.config” to set transaction properties.

Further, it is frequently necessary to use multiple plugins to perform standard transaction manipulation.

This situation is primarily due to accidents of implementation and the difficulties of extending hand rolled syntax parsers. TxnBox is intended to provide a framework in which it is easier to extend TxnBox to provide these ancillary uses than to build from scratch.

Performance

It is easy to get “remap.config” non-performant, particularly with regular expression matching. This is compounded by the paucity of matching comparisons, so that common comparisons such as prefix and suffix require regular expressions.

Better syntax, more specialized comparisons for common cases, and static analysis can all contribute to improving performance without burdening operations staff.

One example is excessive use of regular expressions, which are expensive, in places where more specialized matching would suffice, such as “prefix” and “suffix” matching of literals.

TxnBox exerts itself to do computation at configuration load as much as possible.

YAML

Because of configuration issues, the Traffic Server community examined a number of alternatives and decided to commit to using YAML as the base configuration syntax for all Traffic Server configuration.

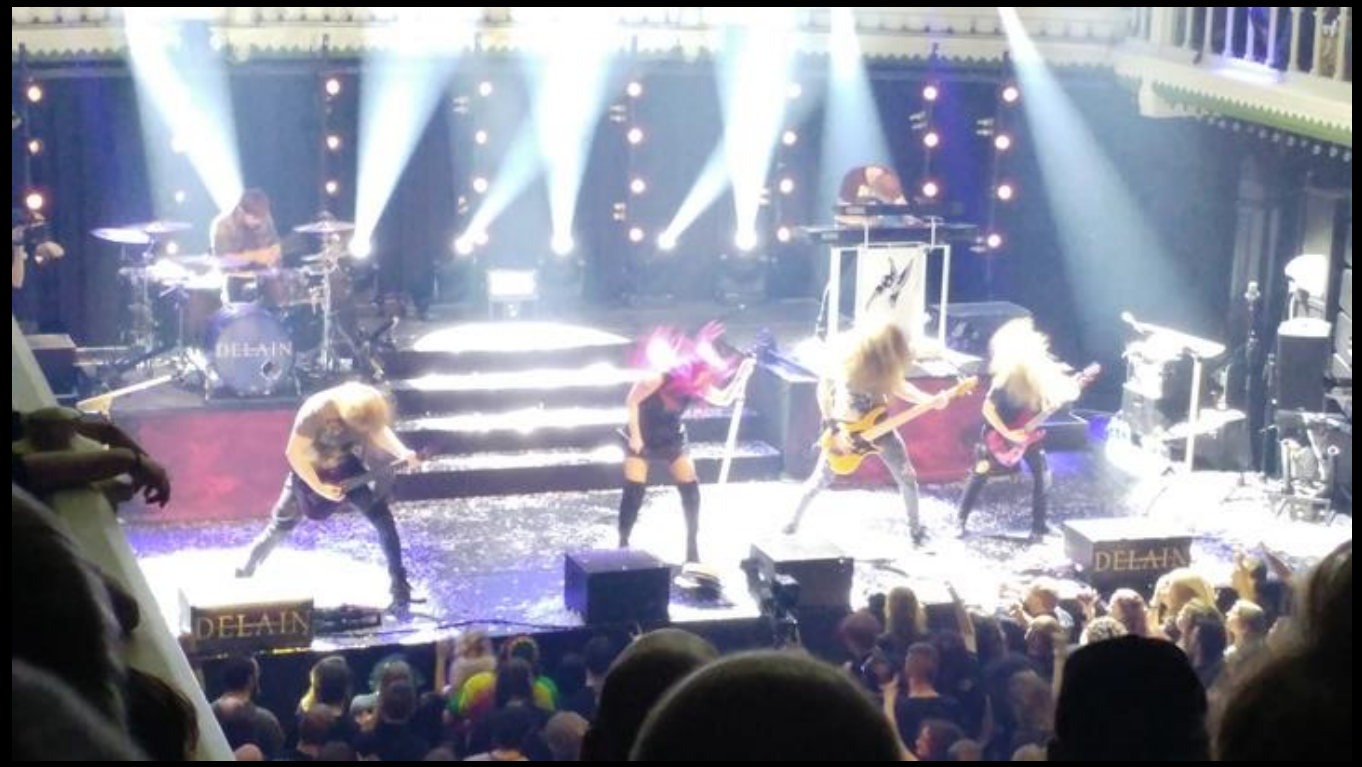
Some previous work was done to upgrade “remap.config” with the same syntax but better internals to fix various shortcomings.

At the Spring 2019 Summit this was rejected in favor of a complete YAML based overhaul, with the goal of taking full advantage of YAML to fix the deficiencies of the current configuration.

TxnBox is my proposal for implementing this goal.

Basics

A few powerful mechanisms combined to yield a flexible and precise result.



verizon media

Feature Extraction

To make a decision about a request, the entire transaction is not considered. A specific subset of the data is used. TxnBox terms this subset a *feature*. E.g the request URL, an HTTP field value, or a combination of these, or other values.

A feature is created by *extracting* it from the transaction.

This *feature extraction* is the fundamental operation in TxnBox.

A primary design goal of TxnBox is to generalize feature extraction.

- Any transaction data, not limited as a single, hardwired feature as in current remapping.
- Additional non-transaction data, such as random numbers or external data sources.
- *Use* of features anywhere values are used.

Notation

TxnBox uses a format string style for extraction. All features can be extracted as strings.

The general notation is deliberately styled on Python string formatting. Braces are used to delimit *extractors* with additional options for style and extended data. These three elements are separated by colons. E.g. for an IP address it might be

```
"{cssn-remote-addr:S:ap}"
```

"cssn-remote-addr" is the extractor which retrieves the client IP data. "S" is the format specifier for upper case, and "ap" is the extended data that indicates extracting the address and port.

Format specifiers are general and apply where possible to all extractors. They follow as closely as possible Python format specifiers.

Extended data is specific to the extractor, intended for idiosyncratic needs.

In general, TxnBox tries to accept single values where a list is expected as a list of length one.

Extractors vs. literals

The format specifier and extended data are useful only in rare circumstances. In addition by far the most common feature extraction is a single data element from the transaction. Therefore for convenience, unquoted strings are presumed to be extractors. E.g.

```
cssn-remote-addr
```

is an extractor that retrieves the remote IP address of the client session.

Quoted strings can contain extractors if those are delimited by braces, as in the same format string style noted previously.

The string, quoted or not, can be proceeded by the YAML type indicator "!literal" in which case it will be taken as a literal and no extractors will be found.

```
!literal "{cssn-remote-addr}"
```

This is a string with braces in it, it is not an IP address.

Braces can also be escaped by doubling them. The previous string is identical to

```
"{{cssn-remote-addr}}"
```

verizon media

Extractor Types

As a special case, if an feature format consists of a single extractor with no additional text, it can be treated as one of the following types

- **String**
- **Integer**
- **Boolean**
- **IP Address**

verizon media

Feature Modifiers

After extracting a feature it can be modified. This is done with a list where the first element is the extraction format and subsequent elements are objects called *modifiers*.

```
[ “{creq-url}” , { hash: 4096} ]
```

This extracts the client request URL and then hashes it in to one of 4096 buckets. The feature type is Integer because the modifier changes the String into an Integer.

It has planned to have a small set of modifiers for specific purposes, among them

- **hashing**
- **URI encode / decode**
- **IP address processing (see IPSpace on a later slide)**

verizon media

Selection

A *directive* is an action to perform.

A *comparison* is comparing a fixed value to a feature. A comparison matches the feature, or it does not.

***Selection* is using the result of comparisons on a feature to select one of multiple lists of directives to perform.**

This is the mechanism for conditional actions.

verizon media

No Selection Backtracking

A TxnBox configuration is a forest where selection creates the branches and the root of each tree is a hook. An integral part of the design is that selection is a one-way descent through the trees for each hook.

Once a selection has been made, only the directives selected are performed, all other branches are discarded.

This has its limitations, but I believe it is overall superior because of the lack of ambiguity. If a directive is performed, it is completely determined which selections occurred to cause that and what other directives were performed.

If no selection matches, the following (not nested) directives are performed.

Comparisons

Comparisons are used to compare a feature to specific values. Most comparisons are string based but there are comparisons for other types.

Comparisons are done in the same order as in the configuration and are first match.

In a selection, each comparison should also have a “do” keyword to specify the directives selected if the comparison matches. A missing or empty “do” value means no directive will be invoked and activity on that configuration tree ends.

Non-Text comparisons

- **Numeric comparisons – eq, ne, lt, le, gt, ge, in (range)**
- **Logic comparisons – or, and, not**
- **IP address comparisons – eq, ne, in (range / network)**

Selection Example

```
with: creq-host
select:
- match: “special.yahoo.com”
  do: # list of directives
- suffix: “yahoo.com”
  do: # …
- suffix: “aol.com”
  do: # …
```

verizon media

Regular Expressions

When a regular expression is used in a comparison that matches, its capture groups become available as features.

The active capture groups can be extracted via numeric indices, e.g. “{0}” for the entire match, “{1}” for the first group, etc.
The format string style lets these be mixed and matched such as “http://{1}/{2}”.

Regular expressions can be applied to a feature directly (not via selection).

This allows easy moving of text between the host and the path, or between any part of the transaction headers.

Values

Directives that set values (such as setting a field value) use extraction in exactly the same way as selection.

Any feature usable for selection can also be used as a value with the same syntax.

This is a major design goal – there is no distinction between data available for selection and data available for direct use

Because extraction is done with an feature string, mixing literals and multiple extracted features is trivial. In combination with regular expression support this enables mixing pieces of features (e.g., specific elements of the path in a URL).

Hooks

The “when” directive is used to specify on which hook other directives are invoked.

The top level of the global configuration of TxnBox requires the use of “when” to contain directives, it is the only one permitted. This attaches every directive to a specific hook.

“when” is a directive therefore can be used elsewhere. This is useful for situations where actions need to be performed on more than one hook for the same selection criteria.

TxnBox can be used via remap rules, in which case there is an implicit “when: remap” directive enclosing the configuration.

IP Addresses

IP addresses can be extracted from the transaction.

These can be compared using equality and range checking.

Predefined sets of IP address ranges can be created for use as an *IP Space*.

- **IP Spaces map from an IP address to property sets in an efficient manner.**
  An IP space is defined by a set of IP address ranges and set of property types, where each range has a property value for some subset of the property types.
- **An IP Space can be used to modify a IP address in to a property value which can then be used for comparison.**

IP Space Example

Data file, loaded as IP space “josh”:

```
172.16.1.0-172.16.1.127, prop1=alpha, prop2=beta
192.168.78.0/25, prop1=gamma, prop3=delta
```

Use:

```
define-ip-space: [ “josh”, “/home/y/conf/josh-ipspace.csv” ]
with: [ cssn-remote-addr, { ip-space.josh: prop1 } ]
select:
- match: “alpha”
  do: # …
- match: “beta”
  do: #...
```

verizon media

Static Analysis

The primary point of static analysis is to avoid requiring the user to remember how to optimize. As much as possible this should be automatic.

For instance, if a regular expression is applied, it can be detected if any of the match groups are used. If not, the match groups can be discarded immediately.

A selection with a large number of literal strings can be detected and optimized by using a hash map or trie to find the matching string instead of linear searching.

Examples

For the encouragement of others

verizon media

Restrict access to stats page

This was the original issue sparked the original “remap.config” work. The goal is to restrict access to the stats pages to specific source IP addresses while rendering it “invisible” to other requests.

The current ACL support doesn’t quite work, in particular “failing over” on access denied isn’t possible. With TxnBox it is simple to select the URL and then do another selection based on the client IP address.

```
- when: creq
  do:
  - with: creq-path
    select:
    - prefix: "/http-status.json"
      do:
      - with: cssn-remote-addr
        select:
        - in: [ "127.0.0.1/8" , "94.31.20.128/25" ]
          do:
            set-url: "{{http-status}}"
        # Otherwise without the rewrite it's a 404.
```

verizon media

Convert 403 responses for missing files.

Replace an upstream 403 response with a 204 response when making a request to a set of URLs with a common prefix (host and path).

The upstream responds with 403 for missing files, but this is not an error for files in a specific subdirectory.

This is a bit more complex than should be required due to issues with the underlying C API. That is being corrected but for now this is necessary.

```
- when: ursp
  do:
  - with: ursp-status
    select:
    - eq: 403
      do:
      - with: creq-host
        select:
        - match-nocase: "localhost"
          do:
          - with: creq-path
            select:
            - prefix: "config/"
              do:
              - set-ursp-status: 204
              - set-ursp-reason: "Config file not present"
              - when: prsp
                do:
                - remove-prsp-field: Content-Type
                - remove-prsp-field: Content-Length
                - remove-prsp-field: Connection
```

Default “Accept-Encoding” to identity

Force requests going upstream to have an “Accept-Encoding”. If not set, set it to “identity”.

```
when: preq
do:
  set-preq-field-default: [ Accept-Encoding, identity ]
```

That was simple!

verizon media

Redirection list

Given a large set of maintained redirections, each having a specific value for the “Host” field which indicates the redirection should take place, redirect incoming requests.

This demonstrates some of the advantages of YAML and use of the “this” extractor.

It is a situation in which static analysis would yield large benefits.

```
txn_box:
- when: pre-remap
  do:
  - with: creq-field.host
    select:
    - match: "redirect.yahoo.com"
      do:
      - redirect:
          location: http://yahoo.com
          status: 301
          reason: Resource has moved
          body: "<HTML><HEAD><TITLE>Resource moved</TITLE></HEAD><BODY><H1>Relocted</H1>The
resource \"{creq-url}\" is now at {this::location}.</BODY>"
```

```
defines:
  - &redirect-body
    status: 301
    reason: Resource has moved.
    body: "<HTML><HEAD><TITLE>Resource moved</TITLE></HEAD><BODY><H1>Relocted</H1>The resource
\"{creq-url}\" is now at {this::location}.</BODY>"

txn_box:
- when: pre-remap
  do:
  - with: creq-field.host
    select:
    - match: "redirect.yahoo.com"
      do:
      - redirect:
          location: http://yahoo.com
          <<: *redirect-body
```

PCRE performance

A recent issue involved a regular expression which led to a stack explosion crash.

```
^/alpha/bravo/[?]((?!param=(ichi|ni|san|shi)).)*$
```

The root problem is the expression wants to match URLs with query strings that do not have the param set to one of a particular set of values. This is painful for regular expression syntax.

The better solution is enabling "not" at a higher level, such that regular matches what is not wanted and it is the action that is not taken. E.g., "-v" for grep. But neither "remap.config" nor the "regex_remap" plugin can do that.

Two examples – note the "not" based example makes the result of the regular expression available, since it matched.

```
when: pre-remap
do:
- with: "{creq-url::pq}"
  select:
  - regex: "^/alpha/bravo/[?](?:param=(?:ichi:ni:san:shi))"
    do: # nothing
  - else:
    do:
      set-url: "staging.example.one"
```

```
when: pre-remap
do:
- with: "{creq-url-path}?{creq-url-query}"
  select:
  - not:
      regex: "^/alpha/bravo/[?](?:param=(ichi:ni:san:shi))"
    do:
      set-url "staging.example.one"
```

verizon media

Rate control beacons

For some source addresses there is very little beacon data so every beacon should go through. For others, there is plenty of data so the beacon rate should be reduced.\

This configuration selects based on the user agent IP address and rolls dice on whether to send the beacon request upstream.

This is still a bit rough, particularly with regard to how to handle beacons that don't get forwarded. For now this would be done by hand rolled rules, but in the longer term it should be done via an IP space.

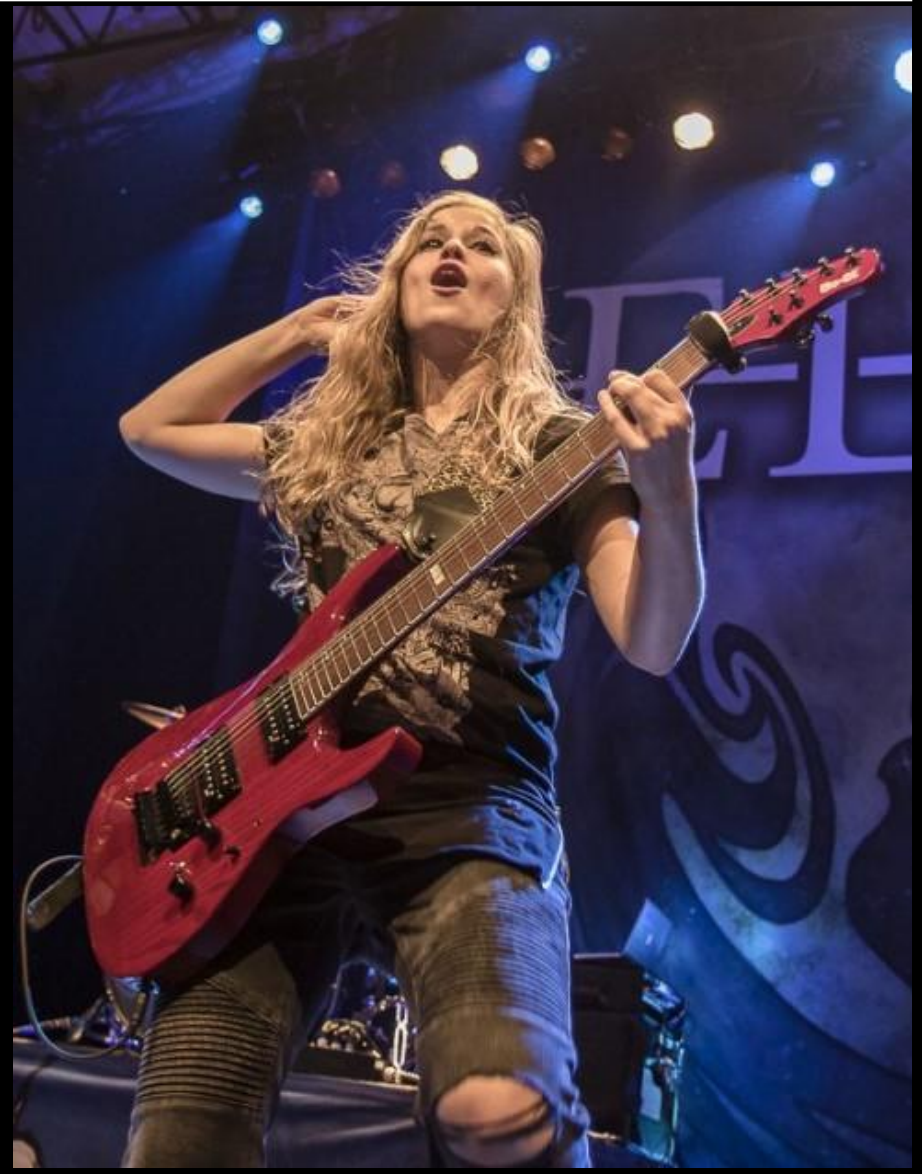
```
when: creq
do:
- with: cssn-remote-addr
  select:
  - in: "1.1.1.0/24"
    do:
    - with: "{rand::10}"
      select:
      - ne: 1
        respond: [ 204, "OK" ]
  - in: "2.2.2.0/24"
    do:
    - with: "{rand::100}"
      select:
      - ne: 1
        respond: [ 204, "OK" ]
```

verizon media

```
when: creq
do:
- with: creq-url-path
  select:
  - prefix: "/beacon"
    do:
      with:
      - cssn-remote-addr
      - ip-space: "beacon"
        property: density
      select:
      - match: medium
        do:
        - with: "{rand::10}"
          select:
          - ne: 1
            do:
              respond: [ 204, "OK" ]
      - match: low
        do:
        - with: "{rand::100}"
          select:
          - ne: 1
            do:
              respond: [ 204, "OK" ]
```

Open Issues

Can I hear a Pull Request?!!?



verizon media

Extractor Names

The names for extractors are quite arbitrary. The best naming convention is still debatable.

For instance it would be easy to tweak the syntax to move elements from the extension to the name, e.g. “{creq-field::name}” to “{creq-field-name}”. Or “{creq.field.name}”. Or even “{creq-field.name}”.

How verbose and specific should the names by? E.g. “creq-url-query” vs. “creq-query”. Is the latter sufficientl unambiguous?

Duration

Basic unit is seconds – durations are in this unit unless specifically noted otherwise.

Nomenclature

- **“ns”, “us”, “ms”, “s”, “Ks”, “Ms”, “Gs” – subsecond are lower case, multisecond is upper case**
- **“minute”, “hour”, “week”, “year” maybe.**

Setting duration unit

- **Extractor extension –** `cache-ttl::ms`
- **YAML type tag –** `!ms 100`

Unfortunately need two types – one for embedded extractors in strings and another for literals.

`cache-ttl::ms` **is the same as** `!ms cache-ttl`

verizon media

Always match

It is necessary to have an “always match” comparison for various reasons. What is a good name?

- **“else”**
- **“otherwise”**
- **“always”**
- **“anything”**
- **“whatever”**

Regular expression notation

Should regular expressions be marked by the name of the directive?

```
match-regex: “http://(?:.*)[.]com”
```

Or by a YAML type indicator?

```
match: !rxp “http://(?:.*)[.]com”
```

Both are roughly the same amount of typing.

The downside of the directive style is every directive that does matching must provide multiple versions.

The downside of the type indicator is obscure syntax, although it is in fact standard YAML.

Case sensitive vs. caseless comparisons have the same issue.

Should the type be similar to PCRE flags, e.g. a combination of letters indicating the comparison style?

- **Regular expression**
- **Case insensitive**

verizon media

Nested Extraction

Having considered nested extractions – where the extractor name is itself extracted – I have decided to not pursue implementation for several reasons.

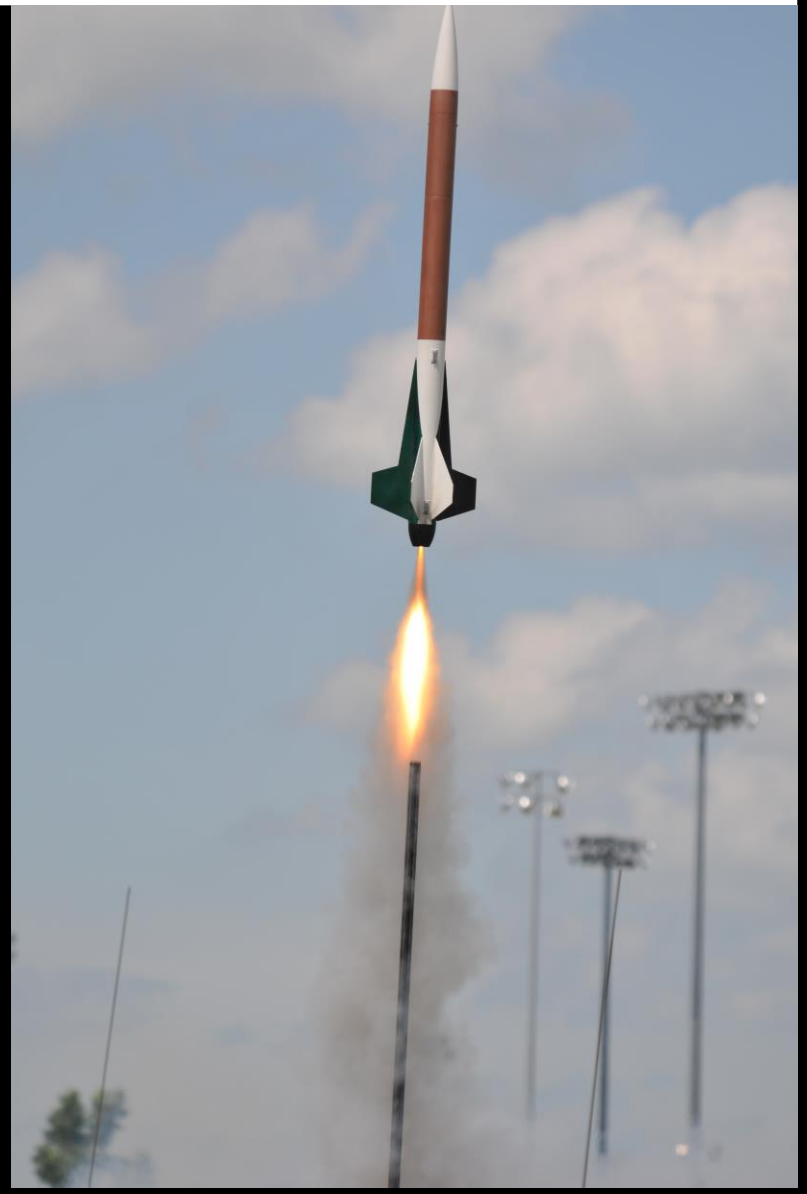
It makes static analysis very difficult to impossible.

The use and syntax is complex.

The uses cases are extremely obscure, I consider it unlikely an actual user would ever think of doing such things.

Future

I have such sights to show you...



verizon media

Documentation

There is some documentation but extending and improving it is a major work item for the rest of the year.

Documentation is a mix of Sphinx based user oriented information and Doxygen based developer reference.

A preliminary but out of date version is at http://docs.solidwallofcode.com/txn_box
- This is primarily due to the continual refactoring during development. Once that becomes more firm the documentation will be updated.

verizon media

Extractors

Other extractors that should be provided

- **Session data.**
- **TLS data.**
- **Random number generator.**
- **Cookies.**
- **Stats / counters.**
- **Transaction and session arguments.**
- **Explicit application of regular expressions, substitution.**

May want to look at making other data available via key/value pairs, such as being able to decorate a parent selection so that the values can be extracted by key.

verizon media

Variables

Variables will be available to support per transaction storage.

Most use cases for the “@ headers” can be replaced by use of TxnBox variables which will be more efficient and flexible.
These can also be used in some cases to replace the C API transaction variables.

Anything that be extracted can be stored in a variable, and retrieving the value is done via extraction.

A/B Testing

Feature modifiers were added primarily to support A/B style testing. The concept is a feature is extracted, hashed, and then the hash value used to determine if the request goes to bucket A or B.

```
with: [ "{creq-url}", { hash: 4096 } ]
select:
- eq: 0
  do:
    set-preq-host: "test.upstream.place"
```

This would extract the URL, hash it, and if 1 in 4096 send it to the test upstream.

Dynamic regular expressions

Currently regular expressions must be configuration load time literals. It is planned to make this more flexible by allowing the use of features in the regular expressions.

There are performance concerns here, because literal regular expressions allow computing various limits at configuration load time, and also pre-compiling the regular expressions.

L7R Project

TxnBox is designed to interact easily with the planned Layer 7 Routing work. One of the issues there was how to specify the upstream selection strategy. This would be trivial with TxnBox by having a directive to set it. Then the L7R code would need no strategy specification support at all.

Calling other plugins

The current design is a directive that specifies the dynamic library and function name, along with a set of string arguments. The function interface is a context object, along with a pointer to string view objects and a count.

By specifying the function name the function selection can be done by TxnBox rather than dispatch logic in the other plugin. Information needed by the other plugin can be gathered by TxnBox via extractors and passed in the function interface.

A goal of this design is enable additional extension flexibility. In effect arbitrary C functions can be called from with TxnBox, extending its capabilities. Data can be passed back via the context object.

Conditional extraction string syntax

Conditional / default strings – currently considering using the syntax "{| ... | ... |}" to indicate conditional extraction.

For example, "{| A | B | C |}" would mean

- **If "A" is not the empty string, use A.**
- **Else if "B" is not the empty string, use B.**
- **Else if "C" is not the empty string, use C.**
- **Empty string**

This makes defaults easy, because a literal is never empty – "{|{creq.field::YRIP}|N/A|}".

verizon media

Design Notes

Comparing to Varnish

Varnish is much more procedural – Traffic Server operates more autonomously and therefore requires only tweaks, not instruction. The general flavor of Traffic Server plugins is *setting values* far more than *performing actions* compared to Varnish.

A significant difference is the use of “if” vs. selection, although these are logically equivalent.

Selection is cleaner, faster, and more compact for a single feature being compared to multiple values. It avoids repeating the feature string and the inevitable typographic errors.

On the other hand, if different features are sequentially compared, “if” is better. The “no backtracking” rule makes the behavior of consecutive checks different, and this case it is selection that requires excess verbosity in the configuration.

The “when” directive functions are similar to reserved “vcl_” function names, although it is easier to have multiple uses of a particular HTTP state.

Extractors are similar to VCL objects.

TxnBox Extensions

TxnBox is structured to ease the work of local extensions.

All of the primary entities (directives, comparisons, extractors) are defined by code during initialization using generic mechanisms. This means additional ones can be added in additional source files without modifying the existing code. A fork with additional entities is thereby made low maintenance – patching consists of adding entire files and rebuilding.

YAML

A fundamental requirement was for the configuration to be in YAML.

For this reason, as much as possible, specialized syntax was avoided even where it would be more compact or more understandable.

While human usability is important, it also considered important to support external tools working with the configuration. The more that is in YAML the easier this can be done and consistency itself improves human comprehension.

In addition, even for humans, structure in YAML can be described by a schema which in turn can be used by editors to make editing the files easier and less error prone.

Static Analysis

A benefit that emerged during the work was the ability to do static analysis on the configuration, even during configuration loading. This enables run time optimizations rather than cautiously conservative coding. The implementation doesn't have to assume the worst case for handling data, it can instead actually look and see if that case occurs and optimize accordingly.

This is used for extraction. If the format is literal then it can be stored statically in the configuration. Extracted strings can be stored in a temporary area if not used again, and copied to context storage otherwise.

Static Analysis

A longer term goal is to enable offline static analysis. A recurrent performance issue currently is mixing literal strings and regular expressions when matching URLs.

An offline analysis could match the literal strings against the regular expressions so that the literal strings could either be moved before the regular expression without changing the results, or eliminating literal strings that are never matched because a prior regular expression matches the same string.

Naming and attached arguments

The naming convention was changed recently so that arguments can be directly attached to the directive or extractor name.

The primary impetus was perceived ease of use. I think it makes the use more natural, closer to standard assignment. It also makes it easier to support non-quoted singleton extractors (which would be noticeably worse if the extension field was used).

The drawback is the argument cannot be extracted. Whether this is a real problem is unclear. It's a rather obscure feature and can be easily worked around in most cases.

If that becomes neccessary I will add some sort of "apply" mechanism for this, which would be more general.

The Problem of Lists

In working with use cases, the need for lists becomes clear. For example, how should mutli-valued fields be modeled?

This has two issues

- **the internal representation**
- **the syntax for manipulating lists**

List Internal Representation

Currently there are three mechanisms which are undergoing testing and experimentation.

- **Tuple support – an array of Features.**
- **Cons cells – the standard Lisp cons cell.**

A key point which makes this more challenging is avoiding standard memory allocation, which is a key design element. If at all possible, the data should be stored entirely in the transaction memory arena. This is somewhat tricky for incremental parsers.

This is where a cons cell shines – it interacts naturally with a memory arena and incremental parsing. In general, lists are almost always going to be iterated, and there's no current support for indexing. Internally, the parsing logic could use a Cons style during parsing and construct a Tuple from the Cons after (which would be fast and inexpensive).

List Manipulation

In general the planned mechanism for list manipulation will be feature modifiers. Among these would be

- prepend, append – adding elements or lists to an existing list.
- prefix, suffix – retain only the initial or final elements of a list.
- drop – remove elements from the start or end of a list (dual of prefix/suffix).
- flatten – if extractors can return lists then lists of lists become possible.
- replace – standard substitution – if a list element matches, it is replaced.
- split, join – break a string in to a list, and convert a list to a string.

Arithmetic Operations

Are arithmetic operations useful? Initially some useful ones will be provide by feature modifiers, such as

- **`clip: [ min, max ]` – force the value to be in the range *min* to *max* inclusive. If *min* or *max* is `NULL` then do not clip on that side. E.g. `clip: [ NULL, max ]` means change the value to be at most *max*.**

For generic operations, a potential syntax would be basically a syntax tree in YAML. E.g. "alpha + bravo – charlie" would be

```
minus:
- plus:
  - var.alpha
  - var.beta
- var.charlie
```

Tuple Selection

Handling “or” situations is easy – the values can be selected on in order while using YAML anchor / reference to duplicate the directives.

Multiple conditions on the same feature is also easy.

What is challenging is something that is straight forward in VCL, which is the logical “and” on two different features. Because of no backtracking, satisfying the first condition prevents “failing out” of the second condition.

In most use cases considered, this can be worked around.

To handle the more difficult cases there is support for *tuple selection*. An n-tuple of features is extracted and then n-tuples of comparisons are applied with the requirement all comparisons must match