Transaction Box

An Apache Traffic Server plugin

Alan M. Carroll, Verizon Media Edge

9 Oct 2019

Introduction

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

It is intended as a general toolbox for manipulating transactions.

This particular version of the work started after a previous effort to convert "remap.config" to YAML.

I discussed the basic design with Miles and he authorized the project.

Goals

- 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.

TxnBox Project

I expect TxnBox to undergo a number of iterations and changes from my original work. Because it is a plugin I can have

- 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/TxnBox-Summit-Fall-2019.pdf

ASF slack channel - #traffic-server-txnbox

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 built to provide a single plugin control for all of these functions.

Request Routing

This is part of my overall effort to shift the focus of Traffic Server from routing *URL*s to routing *requests*. 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.

TxnBox does this – any data related to the request can be used to control decisions, including routing decisions.

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.

In addition, specialized comparisons will be available to remove the need to use regular expressions in many circumstances (e.g. prefix, suffix, negative matching).

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.

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.

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 to 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.**

This is conceptually modeled on “feature extraction” for network packets, where it means essentially the same thing – extracting a subset of the packet for inspection and decision making.

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.

By convention, extractor names have dashes, and an extractor can have a *parameter* which is attached by a dot. E.g

creq-field.X-CDN

Which is the extractor “creq-field” with a parameter of “X-CDN”.

Extractors vs. literals

The format specifier and extended data are useful only in unusual 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 extractor. Nor does it contain quote marks.

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

```
“{{cssn-remote-addr}}”
```

Extractor Types

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

- String
- Integer
- Boolean
- IP Address
- Cons
- Tuple

Feature Modifiers

After extracting a feature can be modified. This is done with a list where the first element is the feature string 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 "hash" modifier changes the String into an Integer.

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

- Hashing
- String manipulation (sub-string, case control, etc.)
- URI encode / decode
- IP address processing (see IPSpace on a later slide)
- List processing

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. It is modeled on “remap.config” and designed to optimize comparing a feature to multiple fixed values.

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.

This helps with self service options – if property rule sets are wrapped with a selection for that property, then any actions are bounded by that selection and can't affect any other path.

Comparisons

Comparisons are used to compare a feature to specific values. Most comparisons are string based but there are comparisons for other types. Each comparison requires a specific type of feature.

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.

Planned comparisons

- String comparisons – match, suffix, prefix.
- Numeric comparisons – eq, ne, lt, le, gt, ge, in (range)
- Logic comparisons – any-of, all-of, none-of
- IP address comparisons – eq, ne, in (range / network)
- List comparisons – for-all, for-any, for-none

verizon media

Selection Example

```
with: creq-host
select:
- match: “special.yahoo.com”
  do:
  - preq-field.Y-TARGET: “special”
- suffix: “yahoo.com”
  do:
  - preq-field.Y-TARGET: “std”
- suffix: “aol.com”
  do:
  - preq-field.Y-TARGET: “other”
  - preq-field.V-TARGET: “aol”
```

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.

Regular expression matching is indicated by the parameter to string matching comparisons. For those that support it,

- **“rx” means use the value as a regular expression to match the feature.**
- **“nc” means no case (case insensitive) comparison.**

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 a feature string, mixing literals and multiple 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. This requirement 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 an 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

Examples

For the encouragement of others

Restrict access to stats page

This was the original issue sparked the "remap.config" work presented at the Spring 2019 summit. 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:
            creq-url: "{{http-status}}"
        # Otherwise without the rewrite it's a 404.
```

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.nc: "localhost"
          do:
          - with: creq-path
            select:
            - prefix: "config/"
              do:
              - ursp-status: 204
              - ursp-reason: "Config file not present"
              - when: prsp
                do:
                - remove-prsp-field.Content-Type:
                - remove-prsp-field.Content-Length:
                - remove-prsp-field.Connection:
```

verizon media

Default “Accept-Encoding” to identity

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

```
when: preq
do:
  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
```

Implementation

Putting it all together to

make the magic happen

verizon media

External Requirements

TxnBox depends on two external libraries.

- **YAML-CPP for parsing YAML. This is the same as used in the ATS core.**
- **libswoc++ which is the Solid Wall Of C++ library. It is essentially a derivative of utility classes in the ATS core. However, those are not normally accessible to plugins. In addition a number of new features and fixes were made to the library which were not allowed in to the ATS core. For these reasons the code was forked from ATS and developed in an independent library.**

Feature

The primary data value passed around is a *Feature*. This is a variant that can contain one of a fixed set of data types. These are

- intmax_t – integral value.
- bool – Boolean value.
- TextView – string.
- IPAddr – IP address
- Cons – pair of Features.
- MemView – fixed size array of Features (aka “Tuple”).

A feature is presumed cheap to copy – variable sized data such as the content of a string is stored elsewhere.

Configuration Loading

MemArena is a class from libswoc++ that manages a memory arena.

When a configuration is loaded, an instance of the Configuration class is created and the relevant configuration data is loaded in to the instance.

Each Configuration has a MemArena that stores the data for the Configuration, including any strings. This minimizes the memory footprint because the parsed YAML tree can be discarded after configuration loading.

In general configuration loading is made more expensive and complex where this can save time, memory or complexity at runtime. As much as possible computations are front loaded into the configuration load.

Runtime Context

For each transaction an instance of the Context class is created. This contains a MemArena used to store transaction lifetime data. Almost all non-immediate feature data is stored here, in particular all feature strings are first stored in the MemArena and then a Feature pointing at the string is created.

Effort is made to avoid allocating MemArena space for transient strings. If a string is used only in a local context, it is put in a temporary area which is overwritten later as needed.

A pre-allocated value is computed during configuration load so that if it is known more than the default amount of allocated space is needed, the initial allocation during Context creation covers it.

The Context is passed to all directives, comparisons, and extractors.

Feature Strings

TxnBox provides support for feature strings.

- **Parsing a feature string, including modifiers, is done with a single call to Configuration.**
- **Extracting a feature using a feature string is a single call to Context.**
- **The resulting Feature instance is typed, and can be checked directly or via virtual method dispatch with metaprogramming support.**

This makes the use of feature extraction everywhere straight forward.

Extending

There are base classes for each of Comparisons, Extractors, and Directives. New instances of these can be added by writing a subclass that follows the implementation requirements and then inserting it in to a table of that type.

In particular new instances can be added in separate files, making a fork of TxnBox with locally extended instances easy to maintain.

Directives are marked with a bit mask of which hooks are valid for the directive.

Extractors are currently not marked for hooks, but are expected to return empty or nil values when used “out of scope”. This may be changed.

Comparisons are marked with the type of feature expected. Because extractors extract features of a specific type, type mismatches can be detected at configuration load time.

All of the internal implements use the same API that would be used by extensions, in order to validate the API works.

Extractors

An extractor is required to be able to return a string, and optionally a feature of a specific type. Other than that, there are no real restrictions on what data an extractor can access.

Extractors must return a Feature and therefore fundamentally one of the types Feature supports.

IP Space

The IpMap class in ATS is a very simplified derivative of a set of IP address handling classes from InfoSecter.

For libswoc++ I am porting the original version to modern C++. This will provide a very fast range based IP address container.

For properties, I will be using a token based compression to reduce the properties of an IP address range to a (relatively) small data object suitable for use as the payload in the IPSpace container. The token mappings can be attached to internal directive storage when the CSV file is loaded. The modifier is then easy to implement – it does the lookup and uses the token value computed at configuration load time to get the value token for the address.

C++ API

The existing C++ API was not suitable for this plugin. As a result I have been constructing a quite different one as needed. This is unfortunately one of the limiting factors on extending TxnBox, although if really needed direct access to the C plugin API is available.

This work is quite similar to Walt's proposal and I hope to work with that to possibly split this off as a distinct project or contribute it back to ATS.

I will discuss this more in my Tech Corner talk.

Open Issues

Can I hear you say 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 sufficiently unambiguous?

Should the argument separator be a different character, such as “@”?

```
    preq-field@X-CDN: “Yahoo”
```

Multiple Configuration Files

A persistent request from operations is splitting configuration in to multiple files so that configuration can be managed on a per application / property / business unit basis.

John Rushford tried implementing "#include" but that did not turn out well – I think it's simply never going to work well.

Consensus seems to be to do directory based loading – load every ".yaml" file in a directory. This is not yet supported but I think this is direction I intend to go.

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`

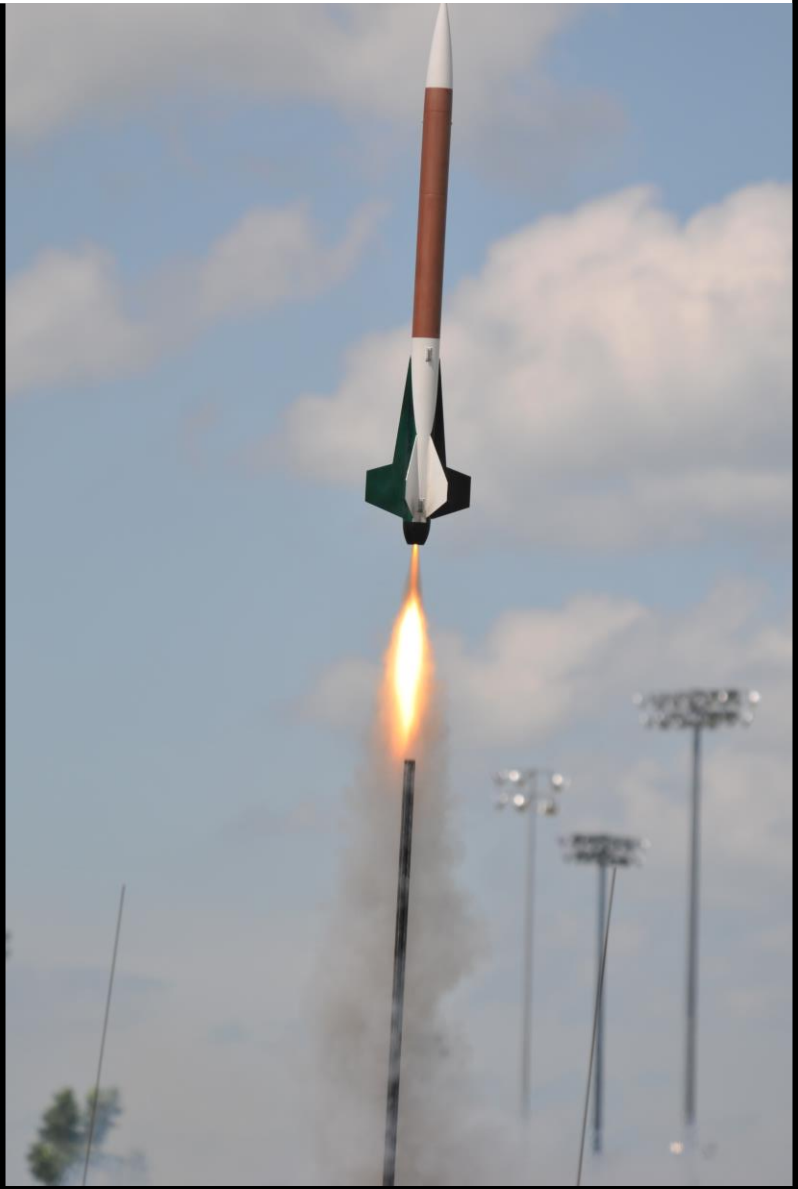
Always match

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

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

Future

I have such sights to show you…



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.

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.

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.

Session Support

I am currently working on session level support. Primarily this involves working with variables to see that variables set in a session directive are visible in transaction directives.

Otherwise, while a significant chunk of work, I don’t see any technological hurdles to implementation.

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.

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.

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|}”.

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.

Next Steps
Q4 2019, Q1 2020

- IPSpace
- Variables.
- Literal string matching optimizations.

Design Notes

I have a cunning plan…

verizon media

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 necessary I will add some sort of “apply” mechanism for this, which would be more general.

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. Many comparisons will take lists and do a logical or across the list.

Multiple conditions on the same feature is also easy.

What is challenging 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.

The current plan is to use a “tuple” comparison that requires a tuple feature and take a tuple of other comparisons, which are then applied. There is a match if all comparisons match. Ambiguity on a tuple element can be handled in the comparison for that element (e.g., use the “or” comparison to allow alternates).

Tuple Selection Example

```
with: [ cssn-remote-addr, creq-path ]
select:
- tuple:
  - in:
    - 2b40:6678:b7:b44/64
    - 2c88:2349:aa93:c9/64
    - 2b34:e001:b337:d021/64
    - ::1/128
  - prefix: “/http-status/”
  do:
    redirect: [ 302, “https://special-place.one” ]
# continue here if the pair doesn’t match.
```

verizon media

Another Tuple example

```
# Presumed to be called from remap.config, in which case the remap hook is assumed.
creq-field.CDNService: "cups"
ussn-conn-dscp: 0
when: ursp
do:
- with: [ ursp-field.Cache-Control, ursp-status ]
  select:
  - tuple:[ { match: “” }, { in: [ 199, 400 ] } ]
    do:
    - ursp-field.CacheControl: "max-age=3600, public"
```