rtgwg
Internet-Draft
Intended status: Informational
Expires: October 14, 2021

S. Bryant
A. Clemm
Futurewei Technologies, Inc.

April 12, 2021

Token Cell Routing Data Plane Concepts
draft-bcx-rtgwg-tcr-00

Abstract

Token Cell Routing is a powerful yet hardware friendly method of constructing data plane packets to meet the needs of new applications. It is based on the use of token cells (special kinds of lightly structured tokens) to provide pointers to procedures pre-positioned in the forwarding layer together with the parameters needed to provide the required processing context. A packet can be composed from multiple token cells as needed to result in new new network processing and forwarding semantics.

Table of Contents

1. Introduction

Advances in data plane protocols are needed to address new network requirements that stretch existing protocols (including MPLS, IPv6 and Segment Routing) to their limits. Token Cell Routing (TCR) is a new network layer data plane technology that provides the ability to program the data plane to meet the needs of many new operational scenarios. TCR is based on token cells which provide a flexible

method describing the required packet action to the forwarder as well as carrying any parameters and other data necessary to correctly execute the required action.

Packet actions are not limited to forwarding actions, and it is possible perform multiple packet actions at any given node. For example, packet actions can include application of special QoS algorithms, collection of telemetry, even assessment of dynamic conditions before the performing of other actions. Token Cells can be differentiated by type depending on the type of packet action that they represent.

TCR is thus analogous to each packet carrying a stack of pointers to procedures together with the data needed by those procedures. The structure used allows token cells to be sequenced and parallelized in groups, and permits the use of pointers to information in other token cells. This results in a powerful method constructing advanced new packet types from token cells to meet network needs of new applications.

TCR therefore supports customizable semantics with which packets are to be processed by nodes encountered along a path, it accommodates flexible addressing semantics that do not necessarily depend on a single addressing format. TCR accommodates custom "guidance" beyond forwarding (such as the definition of QoS treatments that are to be applied, the ability to differentiate behavior depending on dynamic context encountered at a node, and the ability to collect and pre-process telemetry in support of manageability applications). TCR enables the direct processing via a "scaffold" that explicitly indicates serialization, parallelization, disposition rules. It that enables a "lego-esque" composition of packet processing behavior / features while at the same time being hardware-friendly, and easy to optimize for performance at line rate general in nature and easy to extend.

A new TCR features is added by introducing a new procedure in the forwarder and then including in the packet a pointer to that procedure. The procedure knows how to interpret the other information carried in the token cell. In many ways this is similar to the way new FECs are introduced into MPLS and new instructions introduced to segment routing.

TCR thus provides a general, highly extensible data plane that supports custom semantics which is hardware friendly. It thus takes the programmability of both MPLS and Segment Routing to a new level of capability.

A key differentiator from earlier protocols is the ability to process a variable number of processing actions at each hop, at directed by the token cell structure. Furthermore token cells do not need to be processed in the order in which they are placed in the packet, and may be explicitly programmed for a flow.

We would like to note that at the time of writing the current -00 version of the draft, this represents a sketch of an idea that neither we or anyone else has built. Thus, it is likely to have bugs and certainly has many aspects that can be improved on. We would be delighted to work with others who are interested in exploring this idea and developing it further. A starter set of discussion items is included in Section 12 towards the end of the document.

2. The TCR Concept

The foundation of TCR is the construction of the packet from a set of token cells. A token cell is an extended length, type, value construct. The type and part of the value is processed by a longest match engine, which operates much like an IP address lookup engine, but operates on arbitrary constructs rather than being confined to address lookup. As part of its value, the token cell may also carry parameters specific to the token cell type that are needed to process the token cell. Depending on the token cell type, different code points can be invoked that process the token cell. The token cell type determines the semantics, i.e. the function to be applied. It also defines any structure that may be contained in the token cell value.

Token Cell processing can have generalizable packet processing semantics: forwarding is a common semantic, but other semantics can be applied, in a similar manner to the way in which an MPLS label has generalized semantics. The processing of a token cell is: Input - Match - Effect, where "effect" is one of forwarding, token cell disposition, or something else (such as, conditional directives or QoS treatment). Packet processing based on the first n bits of the token cell at a known position, which is matchable on prefix, in which case less significant bits serve as input.

The TCR approach to packet design is differentiated from common protocol designs in that it allow for processing of variable number of token cells per hop, as directed by token cell structure. Which token cells to process, and whether token cells have interdependencies that require them to be processed in serial order or whether they allow for parallelization, is indicated by the token cells themselves, as the token cell structure includes length and serialization indicators. Processing can be serialized per "next token cell" indicator. Processing can be parallelized per "manifest

token cell" that refers to parallel token cells in order to allow for optimization. Note that there is no requirement to "must" parallelize. Instead, parallelization is an optimization that nodes with support for parallel packet processing stages may take advantage of to reduce packet latency due to packet forwarding time.

Token Cells do not necessarily need to be processed in stack order but can be located wherever is most efficient. Some token cells may not be processed at all but can be used to carry meta-data (read-only or writable) that can be referred to from other token cells.

The TCR approach allow for extensibility and programmability through:

- Level 1: Combining different token cells
- Level 2: Parameterizing token cells
- Level 3: Introducing new types of token cells

3. Relationship to prior work

In general data plane packets have been designed with a fixed structure plus some variable number of TLVs to provide additional instructions/advice to the forwarder. In general any address (for example IP address) or instruction (MPLS label of SR SID) is of a fixed length although may be structured into a prefix and suffix arrangement to support aggregation and is processed using a longest match lookup. Where TLVs are used there is their inclusion in the packet is normally free form and thus it is necessary search the packet for the set of TLVs that need to be processed. MPLS has a very primitive parameter system in which one label may be used to provide context for the label that follows. Examples of this are the use of the context label and the use of the ELI/EL pair.

In deigning TCR we noted the generality and simplicity of the MPLS label stack model and the effectiveness of the longest match technique used in IP lookups. Putting these two together we concluded that an LTV approach allowed the creation of a simple, powerful, extensible, hardware friendly packet design.

As we will see later in the document, concatenating the T(ype) and the V(alue) allows a single longest match look-up to resolve which table to look up the action in, and then to absorb as much of the V as is necessary to determine the specific action to take on the T. The stack and pointer structure means that it is not necessary to search for the applicable TLVs, they the forwarder is led to them through the token cell structure pushed onto the packet.

The general method of processing is thus input - match - effect via a match processor. The normal effect is to process one token cell at a time in series, but an effect might be to process multiple other token cells in parallel where the forwarder supports multiple concurrent operations (where parallization is not supported serial processing results in the same effect). Token Cells can have interdependencies and they can allow for cross-referencing (e.g. meta-data, scratch-pad)

4. TCR Packet Structure

A TCR packet Figure 1 is a series of token cells, each token cell carrying a component of the packet delivery system or the payload itself.

```
<Preamble>
<Token Cell>
<Token Cell>
<Token Cell>
  ...
<Token Cell>
```

Figure 1: Structure of a TCR Packet

Token Cells are a unit of packet processing that may include parameters and/or data. The semantics, structure and processing of the token cell is determined by the token cell type. A Token Cell can be thought of as a type of "stem cell" that can be morphed into any packet component, including components not yet designed, resulting in a highly programmable packet design.

The preamble contains a small number of packet elements that are always present in all packets, such as version identification and TTL. It has yet to be decided if the preamble should be carried conventionally or be carried within a token cell.

There is no separate payload portion of the packet. Instead, the payload is carried within a token cell, generally located at the tail of the packet. This allows for different payload delivery semantics at the destination, including simply stripping the payload off or applying a special type of codec. It also allows for the possibility of payload-less packets that can be used for signaling and control purposes.

A token cell MAY contain a complete TCR packet permitting hierarchical encapsulation.

5. Token Cell types and categories

Token Cells have a type. Types themselves can be categorized, depending on the purpose which token cells of that type serve.

The following token cell categories are provided as part of the TCR design:

- Forwarding: A forwarding token cell specifies the destination address and method of delivery of the packet. It may also include the source address as a parameter, but this could also be specified in a separate token cell. Different types within this category may differentiate between address types such as IPv6 or IPv4.

- SLO: A Service Level Objective (SLO) token cell specifies the target quality of delivery, such as latency, delivery time, required discard properties etc., each differentiated by corresponding IDs as separate types. This allows intermediate nodes on the path to apply special treatment to the packet, such as scheduling algorithms, resource reservation, or prioritization, in order meet SLOs as requirement.

- Metadata: These token cells carry metadata that can be referenced and accessed as other token cells are being processed. Metadata can thus be decoupled from token cells that access it, allowing for their independent disposal, not interfering with pushing and popping of other token cells.

- Scratchpad: Scratchpad token cell are in effect writeable metadata token cells, a category of token cell in which the network takes "notes" during the packet transit. Examples of this include recording the route, adding proofs-of-transit, telemetry data, or packet transit time of particular nodes that were traversed.

- Security: A security token cell signs parts of the packet with an agreed cryptographic signature. It includes a signature mask that specifies which token cells and/or portions thereof are covered by the signature. This allows for the possibility of not only the sender, but also nodes in transit being able to sign portions of the packet. One example use would be for telemetry data that is added to a scratchpad by a node being traversed while leaving other parts of the scratchpad open to modification by other nodes.

- Conditional: A conditional token cell is able to test one of more conditions and make invokation of the next token cell (NT) dependent on the conditions evaluating to true. This allows to define more complex behavior, such as the invocation of a

particular function depending on a dynamic condition encountered at a node.

- Directive: A directive token cell specifies some type of action that should be performed. An example would be a directive to collect telemetry or OAM data.

- Manifest: A manifest token cell provides a method of specifying which token cells may be processed in parallel. Parallel processing is optional, and the token cells can also be correctly processed serially. It is up to the entity that specifies the manifest to ensure that the parallelism is safe.

- Rendezvous: A rendezvous token cell is a token cell used to ensure that all parallel operations have completed and that it is hence safe to resume serial operation of the forwarder. A rendezvous token cell may specify the first serial operation to execute after the rendezvous, or it may simply hand off to a new token cell.

- Disposition: A disposition token cell describes what is to be done when the packet leaves the TCR domain. Such a token cell might, for example specify a pseudowire [RFC3985] action (strip the TCR header and send the payload to interface X), or a VPN action (lookup the payload IP address in VRF Y). However, the mechanism introduces the opportunity to attach a more sophisticated disposition action, for example "if the packet arrives before time T, forward using VRF V, otherwise drop the packet".

- Payload: A payload token cell simply carries the payload as its value.

- Other: This is a catch-all category to allow for token cell types that do not fit any of the other categories.

Some of the mentioned categories will be described in greater detail in Section 10. In addition to the mentioned categories, it is expected that other categories would be introduced as needed.

The grouping of types into categories may prove useful for various purposes. For example, it will allow for the articulation of packet grammars and "best packet practices", such as mandating that a packet contain at least one token cell of the forwarding category, or that the first token cell in a TCR packet must not be a token cell of categories metadata, scratchpad, or security. Types allow to further differentiate token cells within a given category.

It should be noted that some of the token categories should be considered experimental. Which token types and even which token

categories to support will in depend on the needs of actual deployments.

6. Token Cell Structure

The structure of a token cell is shown in Figure 2:

```
<Length>
<Next Token>
<Token Cell Type>  -+
    <Cat/Purpose>   |
    <ID>            +- Match Zone
<Token Cell Blob>   |
    <Prefix>       -+
    <Suffix>
```

Figure 2: Structure of a TCR Token Cell

Token Cells will vary in length depending on the token cell type and the contents of the token cell blob, although in practice a given token cell type MAY be a fixed size.

Next token cell is a relative pointer (offset) to the next token cell to be processed as part of the group of token cells to be sequentially processed as a part of the packet action at the node. If there is no next token cell to process, its value is null. Processing of a packet begins with processing the first token cell. Whether or not any additional token cells are processed depends on the guidance provided via the Next Token field.

The token cell type is used to identify the category or purpose of the token cell (Cat/Purpose) and the sub-type (ID) within that class. An example of a purpose might be "Forwarding", indicating that the token cell represents an instruction to forward the packet closer to the destination. An example of a sub-type within that category might be "IPv6", indicating tht the destination is identified by the following IPv6 address.

The set of IDs consists of a set of well known IDs and a set of user specified IDs. This provides both an extensible, and a programmable mechanism for enhancing the protocol over time and within deployments. Token cell type, category, and sub-type ID are not fixed-length fields but represent conventions.

The token cell blob carries the information needed to process the explicit packet that carries it, and/or is a place to record information about the packet for later use. Within the token cell blob there is a prefix and a suffix, which may themselves each be

structured. The structure of the token cell blob depends on the token cell type, some of which may themselves be subjected to standardization.

The token cell blob prefix, which is neither fixed nor a fixed length field is used to qualify the type in the lookup. For example, if the sub-type was IPv6 destination address the prefix would be an IPv6 address. However this is a more general device than just a destination address and allows for token cell type categorization. This may prove useful for various purposes (e.g. packet grammars and Best Packet Practices).

The match zone is the portion of the token cell that is subject to look up (see processing model section). The model is that the lookup will be a longest match of the whole match zone. The token cell design does not specify the length of the match zone or the length of either the ID or the prefix. It is a property of longest match lookup that it will either consume all the bits it needs, or reject the lookup. If a result is found the result can specify the structure of the token cell blob. Note that this is a model, and there is much scope for implementor optimization without sacrificing the generality of the design.

The number of bytes sent to the lookup engine is implementation specific. If the attempted match is longer than needed, the longest match will ignore the overspill. If more bytes are needed, it is a property of longest match that the lookup can be restarted from where it left off.

Within the blob, and in particular within the suffix, any structure can be carried such as subfields, parameters. The definition of what the suffix contains and how it is structured is part of the token cell type definition.

7. Token Cell Processing Model

The TCR processing model is shown in Figure 3.

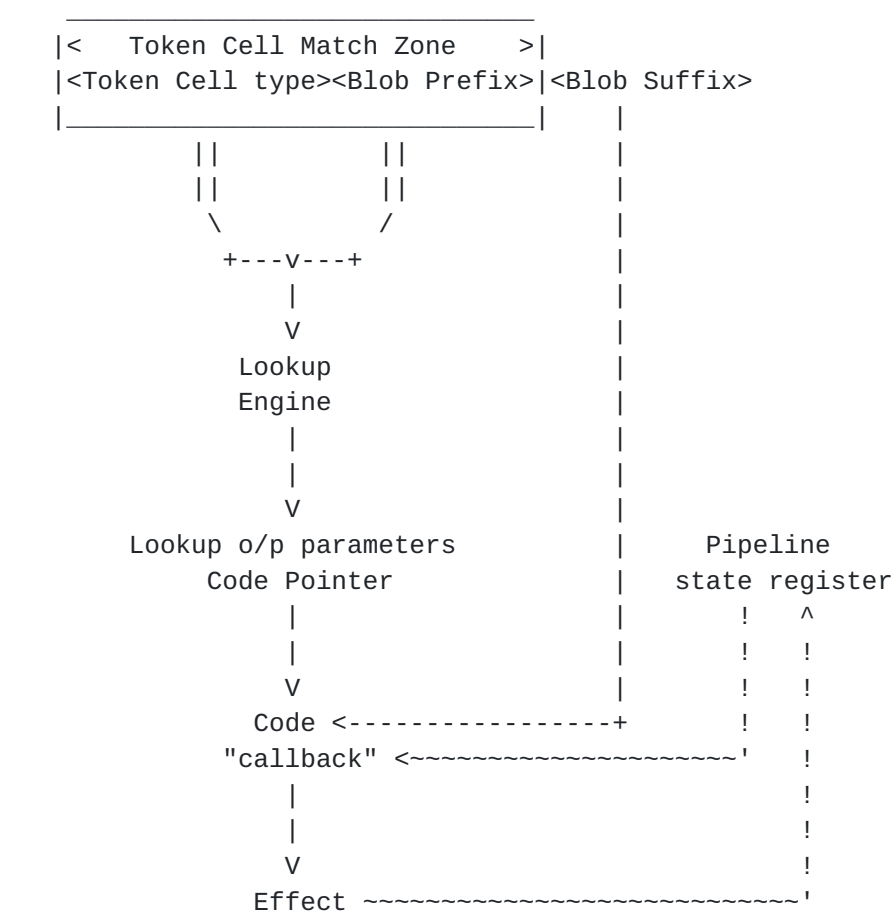

Figure 3: TCR Processing Model

This operates as follows. First the token cell match zone is fed into the lookup engine. The lookup engine performs a longest match and returns a parameter block which includes the address of the code to be executed on the token cell by the forwarder. That code knows how to interpret the token cell. Readers will recognize the genesis of this is a hybrid between an IP address lookup which performs a longest match lookup and returns a parameter block to the IP forwarding code, and an MPLS label lookup which performs a fixed size look-up logically returns a pointer to the executable code (MPLS forward packet, pseudowire, VPN etc) and a parameter block.

Where it cannot be clear that what the length of the blog prefix is from the lookup result, for example where the address is an IPv6 address, that length needs to be encoded in the prefix in some convenient way, such as as a prefix to the prefix.

From the above, it is clear that there is no constraint on the type and structure of the prefix and thus any address type or other construct may be submitted to the lookup engine. Token Cell types

and prefix sets may be added to the forwarder by adding appropriate data to the database queried by the lookup engine, and providing the corresponding callback code to the network processor, in a manner similar to that used in MPLS and in Network Programming. In this regard the approach is compatible with proven hardware forwarding models.

The callback code has access to any other element of the token cell that it needs, and indeed to other elements of the packet as required. Implementations MAY impose a reasonable limitation on the number of processing cycles within which callback code needs to complete.

As part of the effect of processing a token cell, a pipeline state register can be written which can serve as input for the processing of subsequent token cells of the same packet. This allows to compose a pipeline of token cells being processed in which the output of one function serve as input to the next function. In the special case of rendezvous token cells that have multiple predecessors, their respective outputs are merged as part of the specific rendezvous semantics.

8. Token Cell Processing Order

It is entirely possible to require several token cells to be processed at any given node. For example, one token cell may contain a forwarding directive, whereas another token cell might contain a directive related to QoS treatment of the packet for meeting a Service Level Objective (SLO), while a third token cell indicates that certain telemetry data from traversed nodes should be collected.

In the simple case that token cells can or should be serially processed, the processing of token cells will simply be chained, as directed by the token cells' respective NT fields. After processing of a token cell concludes, the reference in the NT field is resolved and processing continues with that token cell. If the NT contains no reference (or in the special case of a conditional token cell evaluating to false), the processing of the packet concludes. In that case, the processing of a packet will require n stages, n being the number of chained token cells.

A packet processing pipeline needs to support a depth that is equivalent to the maximum number of token cells that can be chained.

In some cases, optimization is possible by exploiting parallelization. In the earlier example, it may be possible to perform some tasks in parallel, such as the application of QoS treatment and collection of telemetry data, while other tasks may

still need to be serialized, such as determining which outgoing interface to use as a forwarding decision before performing the QoS actions against that interface. The fact that certain token cells may be processed concurrently can be indicated through a special manifest token cell that references the token cells that follow. Each of those token cells can in turn have their own successors, in effect resulting in separate packet processing "threads" (all processing different token cells of the same packet). While the processing of the manifest adds an additional cycle, depending on the complexity of the workflow this may be more than offset by the parallelization that ensues.

While full exploitation of the optimization potential may require advances in hardware pipeline design, it should be emphasized any such optimization is optional and not required. Also, to maintain packet ordering, the packet will generally still be required to pass through all n pipeline stages. That said, the option of parallelization does allow for hardware pipeline designs able to exploit concurrency of multiple threads and accommodating a larger number of token cells than would otherwise be supported by pipelines of a given depth, respectively reduce the pipeline depth that needs to be supported.

8.1. Serial Token Cell Processing

All packets have an element of serial processing in that the preamble and then the token cell that follows are always processed.

A serial group of token cells is constructed by using the next pointer to point to the token cell to processed on completion of the token cell. The end of a serial token cell group is logically indicated using a NULL next token cell pointer, although none of the foregoing should be taken as dictating the wire format which will be the subject of another text.

```
Token Cell T1 <Length>
              <Next Token> T2
              <Token Cell Type>
              <Token Cell Blob>
Token Cell T2 <Length>
              <Next Token> T4
              <Token Cell Type>
              <Token Cell Blob>
Token Cell T3 <Length>
              <Next Token> -
              <Token Cell Type>
              <Token Cell Blob>
Token Cell T4 <Length>
              <Next Token> -
              <Token Cell Type>
              <Token Cell Blob>
```

Figure 4: TCR Serial Token Cell Processing of Token Cells

Figure 4 shows four token cells. The serial processing instructed by this construct is that token cell 1 is to be processed, followed by token cell 2 and then followed by token cell 4. There are many reasons why this construct is interesting and these are discussed later in this document.

In order to simplify the graphical annotation, references to token cell cells are in the following simply indicated by a numeric identifier instead of being depicted by arrows.

8.2. Parallel Token Cell Processing

In some cases it is desirable to introduce parallel processing to the packet where there are token cells have no result interdependencies. We do this through the introduction of a manifest token cell that contains a set of pointers or offsets to other token cells.

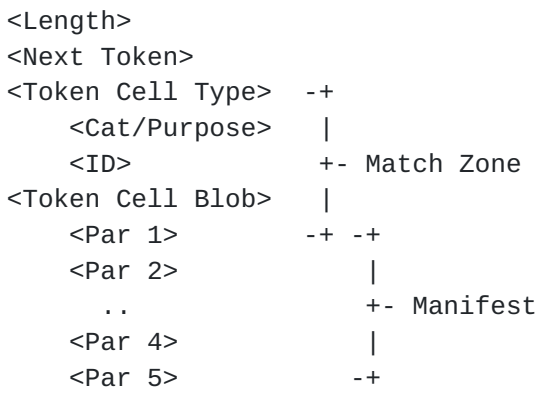

Figure 5: TCR Parallel Processing of Token Cells

The structure of the manifest token cell is shown in Figure 5. The initial part of the token cell is standard to all token cells. There follows a list of token cell pointers to the set of token cells to be processed in parallel. The end of the set of token cells to be processed in parallel is determined when the end of the token cell is reached as indicted by the token cell length. When all the child token cells have completes execution the token cell pointed to by the next token cell field is executed.

Note that the match zone overlaps the manifest. The token cells in the manifest will however be ignored by the longest match which will complete with the ID.

Also note that, as previously mentioned, parallelism is an efficiency issue, not a correctness issue. A forwarder that does not support the parallel dispatch of token cells or that supports less parallelism than specified in the manifest can choose to execute individual token cells (respectively groups of token cells) serially. It is up to the sender to construct the packet as needed and make any token cell interdependencies explit, without requiring the network to second-guess whether or not there are any such interdependencies. In other words, when the order in which token cells are processed might result in different behavior, it is the responsibility of the sender to specify any required serialization as needed.

In most cases where it is used, a manifest token cell will typically be the first token cell after the preamble, to exploit the possibility of concurrent processing threads for multiple token cells in the same packet from the onset. However, this is not an architectural requirement and manifest token cells could also occur later in the packet.

It is possible to remerge concurrent token cell threads using a special rendezvous token cell. A rendezvous token cell awaits for

each of its predecessors to have completed before resuming processing. In addition, output from predecessors (i.e. pipeline state register content) can be aggregated as needed.

8.3. Combined Serial and Parallel Token Cell Processing

Figure 6 illustrates the concept of parallel and serialized processing further. Please note that this is an admittedly complex scenario and most scenarios in practice will be simpler.

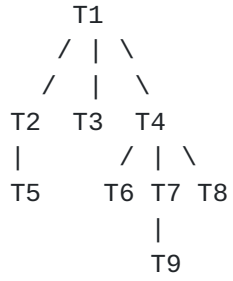

Figure 6: TCR Combined Serial and Parallel Processing of Token Cells

After processing T1, three tokens (T2, T3, T4) can be processed concurrently. T5 has a serialization dependency on T2. T6, T7, T8 can be procesed concurrently once T4 has been processed. Finally, T9 has a serialization dependency on T7.

A corresponding packet is shown in Figure 7. Manifest token cells are introduced to represent the fact that T2, T3, T4 respectively T6, T7, T8 can be processed in parallel. A Next Token field of "-" indicates there is no next token.

```
Token Cell T1 <Length>
              <Next Token> M1
              <Token Cell Type>
              <Token Cell Blob>
Token Cell M1 <Length>
              <Next Token> -
              <Token Cell Type = Manifest>
              <Token Cell Blob>
                <Par 1> T2
                <Par 2> T3
                <Par 3> T4
Token Cell T2 <Length>
              <Next Token> T5
              <Token Cell Type>
              <Token Cell Blob>
Token Cell T3 <Length>
              <Next Token> -
```

```
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell T4 <Length>
                <Next Token> M2
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell M2 <Length>
                <Next Token> -
                <Token Cell Type = Manifest>
                  <Par 1> T6
                  <Par 2> T7
                  <Par 3> T8
   Token Cell T5 <Length>
                <Next Token> -
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell T6 <Length>
                <Next Token> -
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell T7 <Length>
                <Next Token> T9
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell T8 <Length>
                <Next Token> -
                <Token Cell Type>
                <Token Cell Blob>
   Token Cell T9 <Length>
                <Next Token> -
                <Token Cell Type>
                <Token Cell Blob>
```

Figure 7: TCR Combined Serial and Parallel Processing of Token Cells

Token Cell M1 is a manifest that indicates three children (at the protocol level, the number of potential children is subject only to packet size constraints). From a protocol perspective all three children: token cell 2, token cell 3 and token cell 4 can execute immediately and concurrently. When token cell 2 completes token cell 5 runs, when token cell 5 completes that that processing branch is completed. Token Cell 3 runs and when it completes that token cell branch is completed. Token Cell M2 is also a manifest with three children: token cell 6, token cell 7 and token cell 8. When token cell 6 completes that processing branch is completed. When token cell 7 is completed token cell 9 runs. When token cell 9 completes that processing branch is completed. When token cell 8 is completed

that processing branch is completed. When token cells 5, 3, 6, 9 and 8 are completed the token cell group is completed.

It should be noted that the current design of the manifest token cell type includes all pointers to subsequent token cells as part of the token cell blob. Alternative designs are conceivable in which (for example) the next token field would be populated with the first of the concurrent successors, with only additional token cells (beyond the first) to be referenced from the token cell blob.

It should also be noted packet permutations can be accommodated, i.e. the order of the token cells in the token cell group can be any order convenient to the network application designer. For example, the same token cells from Figure 7 could have been arranged also in the following sequence: T1 - M1 - T2 - T5 - T4 - T3 - M2 - T6 - T7 - T9 - T8. The token cells do not need to be arranged in a stack in the way that MPLS or SRv6 arrange their labels or SIDs respectively; the order in which to process token cells is always resolved by the NT reference (respectively any manifest token cell references). However it is RECOMMENDED that backward references are avoided as a packet with no backward references will not form a processing loop.

9. Token Cell Pushing and Token Cell Popping

Token Cell groups can be stacked by pushing a group of token cells onto a packet to encode a hierarchical set of operations to be executed on the packet as it journeys to its destination. This is similar to IP tunneling or the pushing and popping of MPLS labels.

In a manner similar to the pipe model described in [RFC3270] it is a matter for the protocol designer and the operator whether the pushed token cells are able to understand and interact with existing the tokens cells. This will be discussed further in a future version of this document.

Similarly when a token cell group has accomplished its purpose on the packet journey it is popped so that the forwarded can gain access to the next element of processing.

Again considerations similar to [RFC3270] may apply.

For the purposes of this discussion a Token Cell group may consist of a single token cell.

Considerations regarding penultimate hop popping will be included in a future version of this text.

10. Selected Token Cell Type Categories

An overview of token cell type categories was given in Section 5. In this section, some of these categories will be explained in further detail.

10.1. Disposition Token Cells

A disposition token cell describes to the network what actions are to be performed as a packet egresses the TCR domain. A forwarding token cell may for this purpose include a reference to a disposition token cell as a parameter.

All existing IETF network protocols include a disposition semantic although sometimes this is implicit. For example when IPv6 has a next header of "TCP" the disposition instruction is to dispose of the IPv6 header and hand the packet to the TCP handler. Similarly pseudowires have a disposition instruction in the PW label instructing the forwarder how to dispose of the MPLS header and to reconstruct the packet in its original format.

However as the metadata carried in the packet become more sophisticated there is a requirement for the disposition to become more sophisticated.

Disposition token cells thus formalize the description of the egress behavior of the network on the packet and allow a richer egress semantic to be described.

It is conceivable to define TCR such that in the absence of a reference to a disposition token cell, TCR will revert to implicit disposition behavior when the destination address of a forwarding token cell is reached. That will involve popping the token cells up to the forwarding token cell and resuming processing with the subsequent token cell. In other words, a disposition token cell is in that case used to specify any special semantics that would go beyond vanilla implicit disposition.

10.2. Scratchpad and Metadata Token Cells

Metadata, provided by a sender of a packet or by an ingress node, can provide important guidance to nodes along a path to guide processing of the packet. Examples include SLOs that should be taken into account for QoS treatment, profiles to apply towards proessing a packet, and more. Other uses include the carrying of security material as well as the tagging of packets for classification purposes.

Scratchpads refer to writeable metadata, i.e., metadata that can not only be accessed but also modified or added by nodes "in flight", i.e. during transit. Example uses include collection of telemetry and iOAM data [I-D.ietf-ippm-ioam-data], auxiliary data used for measurements such as intermediate time stamps, or proof-of-transit data and data verifying certain properties of nodes being traversed (such as whether from a trusted vendor or located in a certain region).

Metadata and scratchpad token cells allow to carry such metadata as part of a packet independent of the token cells that access it. This decoupling allows to dispose of metadata and scratchpads independent of token cells that process it. For example, it makes it conceivable to collect different iOAM data along different segments of a path, as directed by different token cells, and to export the data only once an exporter or egress node is reached. This way, scratchpad token cells enable applications that rely on sharing of node-specific data along a path to do so without the complications of having to introduce piggyback extensions to the underlying protocol. In addition to adding and updating data items in scratchpad token cells along a path, also additional scratchpads can be added. It also avoids the need for the same metadata to be copied across token cells, for example in cases where the same SLOs are applicable across different segments, each governed by their own respective token cells.

Metadata and scratchpad items can be referenced by other token cells. The specific format of those references is to be determined by the rules governing the content of the respective token cell blobs for the token cell type; in general the reference format will involve an offset from the referring token cell.

Disposition semantics need to include defining what is to happen with metadata or scratchpad carried in corresponding token cells. Possible disposition actions include:

- Discard
- Export (possibly parameterized to specificy export mechanism as well as export target)
- Log (again, possibly parametrized with a logging target)

Metadata and scratchpad data can also be independently authenticated and secured. This allows, for example, to ensure that scratchpad data that is added or modified by intermediate nodes cannot be tampered with. It also allows for the implementation of operations

that authenticate both metadata and scratchpad data before processing it further.

Metadata, let alone scratchpad data, is a concept that is not directly supported by token-based protocols such as SR or MPLS today. While it would be possible to encode such as part of a token, its processing would require the token itself be processed (e.g. as part of a forwarding operation), tied to its being pushed or popped on a stack. This would make it harder to e.g. update scratchpad data (that should be protected from popping and may be buried underneath a token or label stack), to access data without effectively copying it across tokens, and generally to accommodate metadata and scratchpad processing where the lifecycle of data items does not directly correspond to that of segments. With TCR, there is no need to process metadata in "stacked order" nor need for complex token cell rewrite rules in order to preserve data.

10.3. Conditional and Directive Token Cells

Directive Token Cells allow the specification of some type of action or function that should be performed as a result of the token cell being processed. An example of a directive would be a request to collect telemetry or in-situ OAM data and record it as part of scratchpad. It is expected that there can potentially be a large number of possible directives, each distinguished by its own ID respectively token cell type. Different token cell types may impose their own respective structure on the token cell blob to represent different parameters.

Conditional Token Cells are similar to Directive Token Cells in that they allow the specification of an operation to be performed. However, it is different from other categories of tokens in that the outcome of the operation determines whether the NT field will be processed, i.e. whether processing will subsequently resume with the token cell referenced by the NT field or whether it should terminate. This allows for the definition of functionality that should be performed depending on certain conditions that are being encountered. For example, a conditional token cell might allow for a different paths to be selected depending on dynamic circumstances such as load conditions. Similarly, the collection of certain telemetry data might be made dependent on certain conditions being encountered.

10.4. Security Token Cells

Security token cells enable a security mechanism that allows to sign the invariant elements of the packet whilst avoiding signing the packet elements that are modified during the passage of the packet through the network, such as scratchpad token cells.

Likewise, they allow for differentiated signing of different parts of the packet by different signers, such as in cases where nodes add data items to a scratchpad that should be independently signed.

A security token cell allows to carry signature material pertaining to elements of the packet. It includes the following items:

- An identification of the signer
- A mask indicating the parts of the packet respectively token cell(s) being signed
- The signature material itself

The general structure of the security token cell is as follows

```
<Length>
<Next Token>
<Token Cell Type>  -+
    <Cat/Purpose>   |- Match Zone
    <ID>           -+
<Token Cell Blob>
    <Key ID>
    <HMAC>
    <N>
    <Token Cell Mask>
       <Token_Protected>
          <NP>
          <Full_Token>
          <Byte_Protected>
       <Bytes>
```

Figure 8: Structure of a TCR Security Token Cell

The token purpose and ID specify that this is a security token, the exact structure of the security token, and the type of signature that has been used. The structure used here is illustrative and used to explain the concept.

The token cell blob contains the security material and the mask. A possible structure is as follows:

As described in [RFC8754] the Key ID is used to identify the preshared key and the algorithm, though the algorithm may be indicated by the token ID (this is a matter for the token designer).

The Hashed Message Authentication Code (HMAC) is the hash of the complete TCR security token and the N TCR tokens, and within those tokens the token elements specified by the token mask.

For efficiency the mask operates at a number of levels:

- The next N token cells
- The marked tokens within the next N token cells
- Octets or words within a specific token cell

N specifies the number of tokens covered by this security token in addition to itself. If present, Token Cell Mask is a structure that specifies which tokens are actually protected. Token_Protected.NP is the number of tokens within the token mask. Token_Protected.Full_Token is a bit mask of Token_Protected.NP bits indicating which complete tokens are to be included in the signature. Token_Protected.Byte_Protected is a bit mask of Token_Protected.NP bits indicating which complete tokens are to be included in the signature.

11. Example applications of TCR

This section contains a number of examples illustrating how the TCR token system is used to create or "program" packets. The examples are illustrative and not exhaustive. Only the essential features of the packet are shown.

In order to simplify depiction of TCR packets, specifically the order in which tokens are processed and the cross-dependencies between tokens, we will introduce a syntax which identifies individual tokens by a numeric token identifier, and that allows to reference tokens using this identifier as opposed to packet offsets depicted using ASCII art pointers. This is merely done for convenience of textually representing TCR packets in this draft. It is independent of the actual TCR packet and token structure, in which tokens do not have separate identifiers and are simply referenced by offset. This applies to Next Token fields, as well as for disposition. A disposition token cell is referenced through a field in token cells whose type is of the forwarding category.

A packet is simply represented by a sequence of tokens. Each token is represented by a set of fields and their values. In addition, a "pseudo field" is introduced for the numeric token identifier. A reference to another token (from the <Next Token> field, from a manifest token, or from a rendezvous token) contains as value the respective token identifier.

11.1. Basic Tunneling of Payload

In this example we consider a the case where a packet is to be tunneled across a network as one might do in a sub-IP network.

For illustrative purposes we show in Figure 9 an IPv6 address being used as a destination address, but any other address family can be used with the corresponding forwarding token type.

```
              <Preamble>
Token Cell T1 <Length>
              <Next Token> -
              <Type = IPv6fwd>
              <Prefix = Dest Add>
              <Disp = T2>
Token Cell T2 <Length>
              <Next Token>
              <Type = Disp>
              <Disp Parameters>
Token Cell T3 <Length>
              <Next Token>
              <Type = Payload>
              <Payload>
```

Figure 9: Basic Tunnelling Over TCR

The packet starts with a preamble followed by token T1 which is IPv6 forwarding token, a token with the semantic that it is to deliver the packet as close to the target address as it can reach. There is no next token cell to be processed until arrival at the destination and so Next Token is NULL.

When the packet arrives at its destination the token that is pointed to by the disposition pointer is noted and all tokens up to the disposition token are popped (in this case only token 1 is popped).

The disposition token T2 is processed and the disposition parameters say how the payload is to be processed. This may be as simple as providing the protocol type of the payload, but it may also provide information other information such as the identity of a VRF table to use, or an interface to dispatch the packet to in the case of a pseudowire. It also provides a pointer to the payload. When the parser knows how to dispose of the packet, it pops Token 2 and processes token 3. Token 3, in the case of containing a tunneled IP packet, just requires the length to be noted and corrected for the token overhead, the type confirmed as payload and payload to be forwarded as an IP packet.

11.2. Latency-Based Forwarding

In this example we consider the previous case (basic tunneling of payload) Section 11.1 and extend it to provide support for on-time delivery according to an end-to-end latency objective. The support is provided using a Latency-Based Forwarding (LBF). LBF bases the decision on when to sent to packet on how urgently or at what exact time it needs to arrive. To do so, LBF involves an algorithm that determines at any given node whether the packet is "on track" to meet its latency objective, and matches the QoS treatment and scheduling of the packet against a latency "budget" that is determined from latency objective, the latency that was already incurred, and the expected remaining latency towards the destination. For further details, please refer to [DOI.10.1109_NOMS47738.2020.9110431].

To indicate that a packet should undergo LBF treatment, a corresponding token cell type of category "SLO" is introduced. When it is processed, the packet is subject to LBF. Parameters for LBF include the end-to-end latency objectives and a helper parameter to assess the latency being incurred. An additional input is the egress interface that is obtained from processing of the fowarding token cell that precedes it and that can be passed using TCR's pipeline state register.

A packet that enables LBF in conjunction with the preivious case is depicted in Figure 10. It adds one additional token cell over the previous use case, in essence adding the LBF functionality as a "module" that is combined with the earlier functionality. The ability to compose functionality by simply combining corresponding token cells into a packet is part of what makes TCR quite powerful.

Processing of the packet is similar to the processing of the basic tunneling case except that Token 1 the IPv6fwd token contains a pointer to a further token that is to be sequentially processed at every hop, including arrival at the destination. The Next Token pointer in Token 1 points to token 2 which specifies that the packet must arrive at a specified time, and contains information needed to record the time taken and hence the time at which it should optimally leave its current node.

When the packet arrives at its destination a final check is made to see if the arrival time requirement was met and it is processed according to the failure to arrive on time instructions in either the LBF_ontime token or the disposition token. All further processing is as per the above basic tunneling case.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> T2
                <Type = IPv6fwd>
                <Prefix = Dest Add>
                <Disp = T3>
   Token Cell T2 <Length>
                <Next Token> -
                <Type = LBF_ontime>
                <Blob = LBF parameters>
   Token Cell T3 <Length>
                <Next Token>
                <Type = Disp>
                <Disp Parameters>
   Token Cell T4 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 10: LBF Using TCR

11.3. Forwarding with Flexible Addressing

The is an emerging need to support multiple address technologies. There are two cases in point, firstly where an operator wants to use a short address to to address the infrastructure nodes in their network. This is particularly the case in segment routing where there is competition amongst the vendors to introduce address compression due to the extended size of SRv6 data plane headers. There is a secondary benefit to this in that if an address system other than IP is used within the provider network there are security benefits as has been found in operating MPLS. Finally it has been speculated that some application environments would prefer to use their native addresses rather than manage the mapping between those addresses and IP addresses.

This is achieved by creating a new token cell type, populating the FIB with the corresponding addresses and installing in the forwarder the necessary forwarding code. That forwarding code may be generic since the action will almost certainly be address family independent.

In the example shown in Figure 11, the address family is encoded in the address itself. In an alternative model, the token cell type would be specific to the address family.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> -
                <Type = FlexAddr>
                <Prefix = AddressFamily + Address>
                <Disp> T2
   Token Cell T2 <Length>
                <Next Token> -
                <Type = Disp>
                <Disp Parameters>
   Token Cell T3 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 11: Indirect LBF UsingUsing Flexible Addresses in TCR

11.4. Forwarding with iOAM analytics

In many cases, there is desire to collect in-situ OAM data [I-D.ietf-ippm-ioam-data] as a packet traverses a path. There are multiple applications for this, including but not limited to diagnosing performance, identification of bottlenecks, and generation of data to feed machine-learning algorithms for service optimization.

Using TCR, it is possible to indicate that iOAM data should be collected using a corresponding token cell. The token cell can contain in-situ parameters, such as which data items to collect. In-situ data itself can be added to a scratchpad, allowing for its export using a variety of means upon disposition of packet.

One scenario is depicted in Figure 12. In this particular example, we assume that the iOAM data can be collected per T3 in parallel with the forwarding decision being made per T2 in order to show also use of a manifest (T1). iOAM Data items are written to the scratchpad in T5. T4 indicates disposition, T6 contains the payload.

It should be noted that rather than simply collecting iOAM data, other operations could be applied to aggregate that data and result in more refined behavior. In the interest of brevity, the example does not feature security tokens used by intermediate nodes to sign the scratchpad data items that they add.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> -
                <Type = Manifest>
                  <Par 1> T2
                  <Par 2> T3
   Token Cell T2 <Length>
                <Next Token> -
                <Type = IPv6fwd>
                <Prefix = Dest Add>
                <Disp> T4
   Token Cell T3 <Length>
                <Next Token> -
                <Type = Directive-iOAM>
                  <Ioam-parameters - data items to collect>
                  <Scratchpad> T5
   Token Cell T4 <Length>
                <Next Token> -
                <Type = Disp>
                <Disp Parameters>
   Token Cell T5 <Length>
                <Next Token> -
                <Type = Scratchpad>
                <Blob-Scratchpad>
   Token Cell T6 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 12: iOAM in TCR

If iOAM data to be collected includes telemetry about the egress interface, the manifest cell can be omitted as the forwarding decision needs to be made prior to collecting the iOAM data. The resulting packet becomes even simpler, as depicted in Figure 13.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> T2
                <Type = IPv6fwd>
                <Prefix = Dest Add>
                <Disp> T3
   Token Cell T2 <Length>
                <Next Token> -
                <Type = Directive-iOAM>
                  <Ioam-parameters - data items to collect>
                  <Scratchpad> T4
   Token Cell T3 <Length>
                <Next Token> -
                <Type = Disp>
                <Disp Parameters>
   Token Cell T4 <Length>
                <Next Token> -
                <Type = Scratchpad>
                <Blob-Scratchpad>
   Token Cell T5 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 13: iOAM in TCR - serialized

To show the modularity that TCR enables, the third scenario shows iOAM data to be collected while at the same time LBF is being applied to the same packet (Figure 14). Note that in this case, LBF and iOAM could have also been applied in parallel, in which case a manifest token cell would be added. T1 would then point to the manifest token cell, which in turn would point to T2 and T3 as successors.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> T2
                <Type = IPv6fwd>
                <Prefix = Dest Add>
                <Disp> T4
   Token Cell T2 <Length>
                <Next Token> T3
                <Type = LBF_ontime>
                <Blob = LBF parameters>
   Token Cell T3 <Length>
                <Next Token> -
                <Type = Directive-iOAM>
                  <Ioam-parameters - data items to collect>
                  <Scratchpad> T6
   Token Cell T4 <Length>
                <Next Token> -
                <Type = Disp>
                <Disp Parameters>
   Token Cell T5 <Length>
                <Next Token> -
                <Type = Scratchpad>
                <Blob-Scratchpad>
   Token Cell T6 <Length>
                <Next Token> -
                <Type = Payload>
                <Payload>
```

Figure 14: iOAM and LBF in TCR

11.5. FRR with Latency-Based Forwarding

This example depicted in Figure 15 extends the earlier LBF example to include a fast re-route diversion. We show only a single token cell (T0) added at the point of local repair (PLR), but of course the repair might be more complex and need multiple intermediate staging counts to successfully be repaired.

```
                <Preamble>
   Token Cell T0 <Length>
                <Next Token> T2
                <Type = IPv6fwd>
                <Prefix = Reroute Add>
   Token Cell T1 <Length>
                <Next Token> T2
                <Type = IPv6fwd>
                <Prefix = Dest Add>
                <Disp = T3>
   Token Cell T2 <Length>
                <Next Token> -
                <Type = LBF_ontime>
                <Blob = LBF parameters>
   Token Cell T3 <Length>
                <Next Token>
                <Type = Disp>
                <Disp Parameters>
   Token Cell T4 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 15: FRR with LBF Using TCR

The PLR was expecting to forward the packet normally and so is aware that the packet is latency sensitive and understands the semantics and hence importance of token 2. To maintain the expected path quality the PLR MUST use an FRR path while also ensuring the SLO is still being adhered to per the LBF token cell. The FRR path therefore needs to feature nodes able to support LBF token cells, and not incur a latency penalty that would physically prohibit being able to meet the SLO. This path can be selected by the SDN controller, or locally through the use of path attributes applied to the normal IP-FRR path selection process.

On detecting a local repairable failure of the next hop or the link to the next hop the PLR pushes one or more tokens as is necessary to deliver the packet to the destination. In each case the next token pointer points to Token 2 the LBF token. When the packet arrives at the intermediate node chosen by the PLR for the next stage of the repair, the token (in this case Token 0) at the top of the token stack is popped and forwarding proceeds as dictated by token 1. The above operation can clearly be carried out as many times as necessary on the packet to provide a repair, by pushing as many tokens as are needed.

Note that the scheme in this example uses an implicit disposition operation by intermediate nodes as described above in Section 10.1.

11.6. Segment Routing with Latency-Based Forwarding

The operation described in Section 11.5 in which repair tokens are pushed onto the packet is identical to the operation that happens when a packet is configured for segment routing (SR). Technically the packet depicted in Figure 15 IS a segment routed packet.

It therefore follows that implementing segement routing and segment routing enhanced by features such as enhanced QoS is trivial in TCR. As we shall see in the next sections, TCR is capable of enhancing SR significantly beyond that.

11.7. Enhanced Segment Routing with Latency-Based Forwarding

This example illustrates how TCR can be used to add an enhanced QoS capability such as latency based forwarding to segment routing. We saw previously Section 11.6 how all segments can be made to execute a common policy but it might be desirable to execute a different policy in different segments. For example, some segments might underly their own latency objectives (just for that segment), without affecting the overall end-to-end objective.

A packet that achieves this is depicted below (Figure 16. The packet in the example has three segments. Segments 1 and 3 apply LBF for the end-to-end latency objective, per T5. Segment 2 applies a "sub-SLO" just for that particular segment, per T4. Disposition of T1 and T2 is implicit (popping the token on reaching the segment destination), whereas disposition of T3 is explicit per T6.

```
                <Preamble>
   Token Cell T1 <Length>
                <Next Token> T5
                <Type = IPv6fwd>
                <Prefix = Seg 1 Add>
   Token Cell T2 <Length>
                <Next Token> T4
                <Type = IPv6fwd>
                <Prefix = Seg 2 Add>
   Token Cell T3 <Length>
                <Next Token> T5
                <Type = IPv6fwd>
                <Prefix = Seg 3 Add>
                <Disp = T6)
   Token Cell T4 <Length>
                <Next Token> -
                <Type = LBF_ontime (for segment)>
                <Blob = LBF parameters>
   Token Cell T5 <Length>
                <Next Token> -
                <Type = LBF_ontime (for end-to-end)>
                <Blob = LBF parameters>
   Token Cell T6 <Length>
                <Next Token>
                <Type = Disp>
                <Disp Parameters>
   Token Cell T7 <Length>
                <Next Token>
                <Type = Payload>
                <Payload>
```

Figure 16: Enhanced Segment Routing Using TCR

11.8. Enhanced Segment Routing with Differentiated iOAM

The final example shows a variation of the previous example. Instead of applying a specific latency objectives for particular segments, it is possible to also invoke other functionality, such as collecting certain iOAM data only for particular segments, or collecting additional iOAM data for one of the segments, or even for collecting different sets of iOAM data along different segments. The particular example depicted (Figure 17) shows a packet with three segments. One set of parameters is collected for segments 1 and 3 using scratchpad T6, another set of parameters is collected for segment 2 using scratchpad T7.

If instead, segment 2 should collect additional parameters beyond those collected for segments 1 and 2, this could be easily

accommodated by simply setting "Next Token" of T4 to T5 instead of null. (This does not include a possible optimization for parallelization, but attests to the flexibility of the approach.)

```
              <Preamble>
Token Cell T1 <Length>
              <Next Token> T4
              <Type = IPv6fwd>
              <Prefix = Seg 1 Add>
Token Cell T2 <Length>
              <Next Token> T5
              <Type = IPv6fwd>
              <Prefix = Seg 2 Add>
Token Cell T3 <Length>
              <Next Token> T4
              <Type = IPv6fwd>
              <Prefix = Seg 3 Add>
              <Disp = T8)
Token Cell T4 <Length>
              <Next Token> -
              <Type = Directive-iOAM>
                <Ioam-parameters - data items to collect>
                <Scratchpad> T6
Token Cell T5 <Length>
              <Next Token> -
              <Type = Directive-iOAM>
                <Ioam-parameters - data items to collect>
                <Scratchpad> T7
Token Cell T6 <Length>
              <Next Token> -
              <Type = Scratchpad>
              <Blob-Scratchpad>
Token Cell T7 <Length>
              <Next Token> -
              <Type = Scratchpad>
              <Blob-Scratchpad>
Token Cell T8 <Length>
              <Next Token>
              <Type = Disp>
              <Disp Parameters>
Token Cell T9 <Length>
              <Next Token>
              <Type = Payload>
              <Payload>
```

Figure 17: Enhanced Segment Routing Using TCR

12. Items for further discussion

This document does not constitute a finalized design and there are many design decisions that will require further discussion. The following is a partial list:

- Preamble. The preamble needs further study. The goal is that it contains only the minimum of information as it needs to be popped and pushed when a token cell is popped or pushed. We need to understand if there is a need for a number of token cell's indicator, and/or a last child indicator.

- Token Cell identification. As an alternative to referencing other token cells by offset, it is conceivable to introduce token cell identifiers and refer to token cells by their ID.

- Manifest token cells. Rather to require processing of a full token cell, it is conceivable to express manifests as part of a token cell preamble, at least as long as the number of token cells to process in parallel are limited. This could save on stage in a token cell processing pipeline. However, it would result in a slightly longer or more complex preamble.

- Manifest token cells (cont'd.) It should be noted that the current design of the manifest token cell type includes all pointers to subsequent token cells as part of the token cell blob. Alternative designs are conceivable. For example, the design might be altered to allow for the next token to be populated and only require any additional token cells to be referenced from the token cell blob.

- Rendezvous token cells. As an optimization, it is conceivable to combine manifests and rendezvous points into a single token cell.

- Security token cells. Further investigation is needed to determine the most effective structure, specifically compact yet efficient encodings for the token cell mask that is used to identify the portions of the packet being signed.

- Disposition token cells. There are different ways that disposition token cells could be referred to for processing, including through a DT parameter (Disposition Token Cell) as part of forwarding token cells, through a separate field as part of the token cell structure, or indirectly by virtue of popping a forwarding token cell when the destination is reached and processing the token cell behind it.

- Disposition token cells for processing by intermediate nodes. The example in Section 11.5 uses an implicit disposition operation of forwarding token cells by intermediate nodes, in which a forwarding token cell is simply popped and processing simply resumes with the subsequent token cell. It is conceivable to apply explicit disposition operations instead for the sake of more consistent semantics. Explicit semantics of "pop and resume processing with subsequent token cell" could be incorporated into the semantics of a forwarding token cell itself, or potentially indicated by a corresponding flag in the prefix.

- Implementation profiles. Implementations may impose a reasonable limit on the number of token cells that can be serially processed at any one node. This will facilitate mapping to packet processing pipelines, which then support a predefined number of token cell processing stages per packet at line rate while maintaining constant packet processing delay at any given node. Determination of reasonable constraints is for further study. Analogous limitations may apply to the number of processing cycles that can be performed by callback functions, within which the processing of any given token cell needs to complete.

- Implementation profiles (contd.). For further study are any mechanisms for the discovery of constraints as mentioned in the previous list item, as well as any capabilities for negotiation of corresponding profiles and the definition of node behavior when such limitations were to be breached (in all likelihood resulting in aborting the processing of the packet, dropping it with corresponding error code).

- The design only requires forward token cell pointers and this prevents processing loops. We need to understand if this is too significant a restriction. If this restriction is removed some form of maximum tokens to be processed limit, similar in concept to TTL may be needed.

13. Security Considerations

This section concerns itself with the security of the TCR dataplane.

The security of the control and management plane is a matter for the designers of those aspects of the solution. However, it is not anticipated that the securing of those components will be any more onerous than securing the control and management plane of other IETF designed dataplanes.

Security of entry to the dataplane will depend on what entities have access to the dataplane. If TCR is used as a single domain sub-IP

layer in the way that MPLS is used, then it will have the same security properties as MPLS in that it is extremely difficult for an unauthorized party to inject traffic into such a network because with TCR such traffic is easily recognized at network ingress and dropped. If the traffic is a TCR packet that is to be carried across the TCR network it will be encapsulated and so, in the absence of a TCR network error, will not be able to escape the encapsulation and cause harm. Only if a TCR network (or node) were to peer with another TCR network would there be a security concern with that third party having the ability to control the actions taken on the packet. Such a case is for further study. It should be noted that similar situations have been satisfactorily addressed in MPLS.

We now need to consider the security of the contents of the packet. Clearly we could craft a token that signed the payload, and where the payload was a token, we have the option of including that signature in the token itself. However, securing the tokens themselves is more interesting because we need to authenticate selected components of the packet header, such as single tokens, groups of tokens or even components/portions of tokens. This is needed to allow the differentiation of metadata that must not be altered from scratchpad data that may be modified during packet transit. In addition, we need to allow intermediate nodes along a path to authenticate data that they modify, e.g. scratchpad data items that they create.

The approach used in TCR allows the authentication of tokens and processing guidance they contain for additional security. Furthermore there is flexibility for intermediate hops to provide their own authentication, to secure scratchpad-type data added to packets along a path. This also allows for "authenticity chains" in which nodes verify the authenticity of data items they operate on before modifying and signing the update.

It is clear that the above approach is different from earlier protocols where payloads are generally signed in their entirety and do not include support for differentiated signing, accommodating multiple signers of different packet aspects along a path.

Most protocols secure their payload in its entirety, exposing only the packet header for processing (unless that is tunneled as well). TCR is a protocols that include additional packet components that may require more differentiated securing. Specifically, it includes guidance for how to process packets, including tokens, and metadata. In addition, TCR includes some packet components that can be modified or added by intermediate nodes in transit, specifically scratchpad data. This includes telemetry and iOAM data as well as data to indicate and verify certain properties of nodes that were traversed. While some data must not be modified, other data items might be added

and/or be subject to modification. An example would be data that aggregates or analyzes telemetry data encountered in transit, for example an indicator of a minimum or maximum (queue length, utilization, latency) encountered along a path. TCR thus includes a mechanism that allows the operator to ensure the authenticity of packet data beyond the payload, but that allows them to do this in a way that exempts certain data items which are allowed to be modified along the way, with the option to allow the corresponding nodes to secure these modifications. This allows receiving applications to (for example) verify the authenticity of scratchpad data, and allow for the modification of data items where such modification is permitted without compromising the authenticity of the remaining portions of the packet.

The design of the security token is described in Section 10.4 . This can only be used to sign itself and tokens or token contents after the security token. By including in the security token a mask structure it is possible to select what is to be signed. The efficiency of this method is described in Section 10.4.

Matters related to inter-domain security will be considered in a future version of this text.

14. IANA Considerations

This document makes no IANA requests.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

15.2. Informative References

[DOI.10.1109_NOMS47738.2020.9110431] Clemm, A. and T. Eckert, "High-Precision Latency Forwarding over Packet-Programmable Networks", NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, DOI 10.1109/noms47738.2020.9110431, April 2020.

[I-D.ietf-ippm-ioam-data]
Brockners, F., Bhandari, S., and T. Mizrahi, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-11 (work in progress), November 2020.

[RFC3270] Le Faucheur, F., Wu, L., Davie, B., Davari, S., Vaananen, P., Krishnan, R., Cheval, P., and J. Heinanen, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services", RFC 3270, DOI 10.17487/RFC3270, May 2002, <https://www.rfc-editor.org/info/rfc3270>.

[RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture", RFC 3985, DOI 10.17487/RFC3985, March 2005, <https://www.rfc-editor.org/info/rfc3985>.

[RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <https://www.rfc-editor.org/info/rfc8754>.

Authors' Addresses

Stewart Bryant
Futurewei Technologies, Inc.

Email: sb@stewartbryant.com

Alexander Clemm
Futurewei Technologies, Inc.

Email: ludwig@clemm.org