Token Cell Routing: A New Sub-IP Layer Protocol

Stewart Bryant
Institute for Communication Systems, University of Surrey
Guildford, Surrey, UK
s.bryant@surrey.ac.uk

Alexander Clemm
Futurewei
Santa Clara, California/USA
alex@futurewei.com

***Abstract*—We present Token Cell Routing (TCR), a new sub-IP layer protocol that provides a powerful yet hardware-friendly method of constructing data plane packets to meet the needs of new applications. Token cells are a special kind of lightly structured token which can be programmed with different semantics to serve different purposes. A TCR packet consists of a series of token cells, each associated with a certain functionality. Using longest prefix matching, code points are invoked that implement logic specific to the token cell's type. Packets also define token cell processing workflows and allow token cells to cross-reference data. This results in the ability to program very powerful functionality. At the same time, processing is kept simple, allowing for straightforward implementations able to operate at line rate. TCR is currently still in the concept stage while proofs-of-concepts and venues for standardization are being explored.**

Keywords—Networking protocols, network programmability, sub-IP

I. 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. Challenges arise from many sides [1]: New networking use cases call for greater service level awareness and adherence to stringent "high-precision" service level guarantees. Network providers and application developers demand greater programmability that provides them with the ability to exert greater control over packet forwarding behavior. Network optimization requires greater visibility into flow and packet telemetry.

To address those challenges, we present Token Cell Routing (TCR), a new sub-IP layer protocol. TCR is based on token cells which provide a flexible method to describe required packet actions to the forwarder. Token cells are a special kind of tokens that are distinguished by the fact that they have a certain lightweight structure and can be of different types, each associated with its own semantics and able to carry any parameters and data necessary to correctly execute the required action.

Processing of a packet is not limited to a single token cell; instead, sets of token cells can be processed, with interdependencies and processing workflow described as part of the packet and special token cells. This allows token cells to be serially chained or, for optimization purposes, be concurrently processed. It also permits cross-referencing of information across token cells.

The ability to compose packets from token cells results in a powerful mechanism that allows for the dynamic construction of packets with new network processing and forwarding semantics to meet the needs of new applications. Code associated with particular types of token cells is invoked through a match engine that causes the execution of supporting microcode based on longest prefix matching applied to a portion of the token cell. The processing of each token cell can by itself be very simple. Combined with the fact that the processing of successive or parallel tokens needing to be processed a part of a packet can be mapped to pipelines with a limited number of stages, TCR is expected to be reasonably straightforward to implement at line rate performance. At the same time, TCR itself is highly extensible, as it is possible to support new token cell types by registering corresponding codepoints with the match engine.

In combination, this allows for a "lego-esque" composition of packet processing behavior while at the same time being hardware friendly, easy to optimize for performance at line rate, general in nature, and easy to extend. A key differentiator from earlier protocols is the ability to process a variable number of processing actions at each hop, as directed by the token cell structure. This makes it possible, for example, to define behavior to meet a defined end-to-end latency objective, along a certain sequence of path segments with a specific backup, while collecting specific telemetry along different path segments, all within the same packet. Furthermore, token cells do not need to be processed in the order in which they are placed in the packet, reducing the need to rewrite packets upon processing, and may be explicitly programmed for a flow. All of these are features not available through other protocol alternatives today. While this is the first paper that describes TCR, it is our goal to eventually submit TCR to standardization in the IETF and an initial draft has been submitted [2].

The remainder of this paper is structured as follows: Section II provides an overview of related work. Section III provides an overview of TCR and constitutes the core of the paper. It discusses packet and token cell structure, explains how token cell and packet processing workflows are defined, presents a general design of a token cell processing engine, provides an overview of selected token cell types, and discusses other considerations. Section IV presents some use cases that illustrate how TCR can be used to achieve interesting packet behaviors. Section V outlines next steps and open issues. Section VI concludes the paper.

978-3-903176-36-2 ©2021 IFIP

II. Related work

Network layer protocols are normally highly optimized and only extensible through a type-length-value (TLV) approach where the TLV may be an extension header. TLV extensions are poorly regarded due to the difficulty of implementation optimization. Multi-protocol label switching (MPLS) [3] took a radically different approach in which the protocol is built around an opaque type that vectors the processor to an operation. New functionality is added by including a new function in the label switching router and invoking it by including a label in the packet that invokes the operation. Whilst TLVs (extension headers) are able to carry parameters in IPv6 [4], MPLS is unable to do this in the label stack. MPLS can carry parameters below the stack in a control word (CW) [5] and IETF Deterministic Networking [6][7] does this by using MPLS as a one hop tunnel to build a overlay network. Segment Routing (SR) [8] is a way of describing a sequence of forwarding instructions in a network layer header, and network programming is a method of providing a limited number of parameters in the suffix of an IPv6 address. None of the older network layer protocols have a sophisticated policy language.

Big Packet Protocol (BPP) [9] is new data plane protocol and programmable packet framework that allows to include special packet metadata that provides guidance to network devices for how to process the packet. Various applications have been defined, including latency-based forwarding able to guarantee packet delivery with high-precision latency objectives, and operational flow profiling to provide greater visibility and insights into flow performance. TCR is similar to BPP, and indeed partially motivated by it, in its ability to let operators dynamically introduce new custom packet processing semantics. However, the metadata used in BPP to define the guidance can be complex and include multiple conditional directives with varying numbers of operations and parameters. This makes efficient support by hardware challenging and does not allow for easy mapping into packet processing pipelines [10]. In addition, there are some inefficiencies in the realization of behavior that is differentiated by segment or node type, requiring the evaluation of additional conditions that can be avoided with token cells that can be arranged in a stack. These items have been addressed with TCR, which provides its semantics using a combination of multiple simple tokens instead of a single set of more complex protocol fields.

P4 [11] is a technology for the Programming of Protocol-independent Packet Processors. It is not a protocol, but it shares TCR's goal of making it easier to let users define networking behavior and facilitates the implementation of new protocols. It is thus potentially complementary to TCR and could possibly be used for its implementation, subject to further research.

III. Token Cell Routing

A. Overview

At the foundation of Token Cell Routing is the composition of a packet from a set of token cells, special kinds of tokens with a very lightweight structure and programmable semantics (Fig. 1). Each token cell represents a unit of processing, the precise semantics of which depend on the token cell type. Examples include forwarding token cells that contain addressing information used to guide a node's forwarding decision, token cells that contain guidance for QoS treatment of the packet related to service level guarantees, token cells that instruct the collection and aggregation of telemetry data, security token cells that contain authentication material, and even payload token cells that include the actual payload that is to be delivered.

Fig. 1. A TCR Packet

An important feature of TCR is that it enables the expression of the interdependencies between token cells. This allows a packet to specify which token cells should be processed serially and which token cells can be parallelized at a network node. The fact that processing flows are not restricted to simple push/pop semantics, processing whichever token is on top, is an important differentiator from other token-based protocols. Furthermore, token cell processing supports pipelining of output results from the processing of one stage as input to the next stage. These facilities enable the composition of network behavior by combining token cells as needed. At the same time, not all token cells need to be processed in any given node and stacking is possible, allowing the packet designer to combine behavior that applies end-to-end with segment- or even node-specific behaviors. In addition, special metadata and scratchpad token cells are supported that contain data which can be referenced from other token cells during processing. (Scratchpad token cells contain writeable metadata for use by applications such as flow telemetry collection [12] and aggregation).

Token cells themselves are processed by a processing engine that is based on a longest match engine. This engine operates much like an IP address lookup engine but is able to operate on arbitrary constructs rather than being confined to address lookup. 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 is: Input – Match – Effect, where "effect" is one of forwarding, token disposition, or something else (such as, conditional directives or QoS treatment).

In addition, the processing engine allows for the registration of code points for different token cell types, providing a hardware equivalent to corresponding software architectures that allow for the registration of callback functions. This facilitates the ability to easily extend TCR with new token cell types. By keeping token cell semantics simple, the processing of each token can be limited to a very small number of CPU cycles with a fixed upper bound. This, combined with the composition of packets into token cells, allows their mapping into packet processing pipelines that allow for the processing of packets at line rate.

Similar to MPLS, TCR provides a light-weight method of pushing and popping token cells. Unlike MPLS the preamble needs to be retained with each push/pop action, but the preamble itself will be quite small and thus the operation is expected to consume much less forwarding resource than for example, pushing an IPv6 header.

Collectively, these facilities result in a powerful and highly-performant network programming framework that make TCR quite unique. The following subsections describe different aspects of TCR in further detail.

B. TCR Packet Structure

A TCR packet starts with a preamble which contains a small number of housekeeping items, such as version (to allow for subsequent iterations of the design) and hop-count (as a network safety feature). The remainder of the packet consists of a sequence of token cells. There is no separate payload portion of the packet. Instead, payload is carried as part of just another 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.

The first token cell after the preamble is the first token cell to be processed by the forwarder. This may be the only token cell to be processed at this layer of the network, or the token cell may instruct the forwarder to also process one or more additional token cells before sending the packet to the next hop. When a TCR packet exits the network layer, the set of token cells associated with that layer are popped. If required, the preamble is reconstructed and processing continues with the next token cell.

Token cells allow the expression of interdependencies between token cells. This allows a packet to specify which token cells should be processed as a serial sequence, and which tokens can be parallelized at a network node. Furthermore, token cell processing supports pipelining of output results from the processing of one stage as input to the next stage, similar to the way commands can be pipelined in Linux. For example, the processing of one token cell might result in determining the output interface as part of the forwarding decision, which might then be fed into a subsequent token tell specifying that certain telemetry data for that interface should be collected. Contrary to other protocols in which some such behavior is implied and part of built-in fixed semantics, TCR allows this pipelining behavior to be explicitly programmed and hence provides much greater flexibility. In addition, special metadata and scratchpad tokens are supported that contain data which can be referenced from other tokens during processing.

Which token cells to process, and whether to process token cells in serial order or whether to allow for parallelization, is indicated by the token cells themselves. Processing can be serialized using the "next token cell" indicator, processing can be parallelized using "manifest token cell" that refers to parallel token cells which allow for optimization.

C. Token Cell Structure

The structure of a token cell is shown in Fig. 2 .

The first field of a token cell indicates its length. 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.

The next field, Next Token cell (NT), is a relative pointer (offset) to the next token cell to be processed as part of the group of token cells that are to be sequentially processed as a part of the packet action at the node. If no next token cell is indicated, processing of the packet at that layer finishes. In cases where the token cell is one of several that are being processed concurrently in separate "threads", the processing of that packet processing thread concludes, as will be discussed further below.

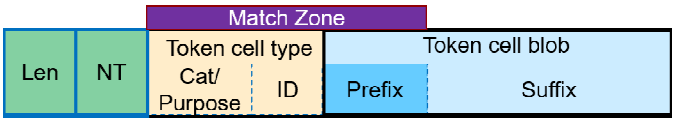

Fig. 2. The structure of a token cell

The token cell type is divided into two components in order to provide some structure to the token cell type identifier space. The first component serves to identify the category, or purpose, of the token cell. This may prove useful for various purposes (e.g. 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 must not be a token cell of categories metadata, scratchpad, or security). The second component identifies the sub-type (ID) within that class. For example, one sub-type within the forwarding category might indicate forwarding to an IPv6 address. Other categories are described in subsection III.E.

The set of IDs consists of a set of well-known IDs and a set of user-specified IDs. In conjunction with an extensible token cell processing engine as described further below, this provides both an extensible and a programmable mechanism for enhancing the protocol over time and within deployments.

Following the token cell type, 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 a blob, there is a prefix and a suffix which may themselves each be structured. The structure of the blob depends on the token cell type, some of which may themselves be well known structures. The 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 blob prefix would be an IPv6 address.

The match zone is the portion of the token cell that is subject to look up (see section III.F). The model is that the lookup will be a longest match across the whole match zone, including the token cell type and the blob prefix. 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 will reject the lookup. If a result is found, the result implies the token cell type and its length. The token cell type also specifies the structure of the token cell blob, i.e., of the remaining portion of the token cell.

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.

D. *Composing Token Cells and Token Cell Processing Workflow*

It is entirely possible to require several token cells to be processed. For example, one token cell will contain a forwarding directive, whereas another token cell might contain a directive related to QoS treatment of the packet in order to meet a Service Level Objective (SLO), whereas a third one 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. 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 should be able to be chained.

The processing of each token cell will likewise require a few CPU cycles. While the Token Processing Engine (described in more detail in section III.F) may be highly efficient and able to conduct a longest prefix match on the match zone in a single cycle, processing at the codepoint that is invoked will require additional cycles in order to process blob suffix, parameters, and any associated logic. The processing needs to be bounded to be able to complete within a bounded number of CPU cycles. The precise number will be dependent on implementation, as the number of CPU cycles required to process a packet determines the amount of buffering and CPU "real estate" that is needed in order to be able to support processing at line rate. Eventually, it may be subjected to standardization or to the definition of capabilities.

In many 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 performed 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. 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.

The mapping of a workflow with multiple concurrent token cell processing threads is also depicted in Fig. 3. In the workflow, token cell 1 is succeeded by token cells 2, 3, and 4 which have no dependencies on each other and be processed concurrently. 2 has a successor in 5, whereas 4 has successors 6, 7, 8, and so on. The corresponding packet can arrange the token cells in multiple ways, two variants of which are depicted. Red (dashed) arrows indicate serial "next token cell" dependencies, whereas "M" token cells MA and MB refer to manifests with concurrent successors, indicated by grey (solid) arrows).

It should be noted that this is a somewhat extreme example, and that we do not expect many applications that require the processing of more than a small number of token cells. Likewise (not depicted), it is possible to remerge threads using a special rendezvous token cell that waits for each predecessor to be completed before resuming processing. While we expect this capability only to be rarely needed, the TCR framework does allow for it.

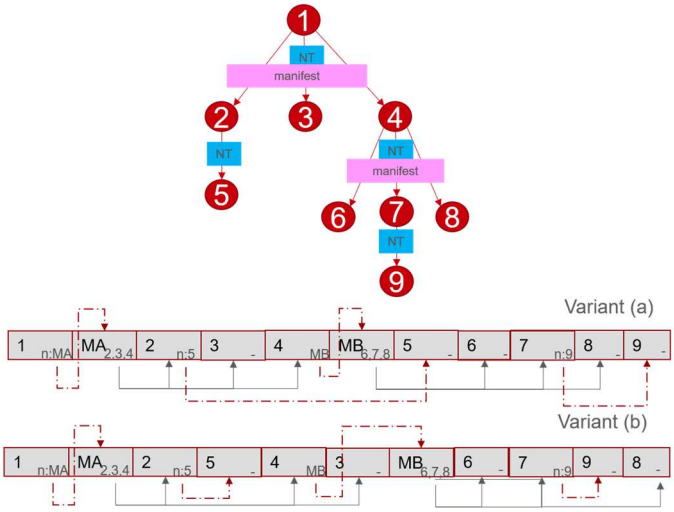

Fig. 3. TCR Parallel Processing Workflow and Use of Manifest

E. *Selected Token Cell Type Categories*

In this section we discuss some of the token cell type categories that we anticipate are needed in the 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. Different types within this category may differentiate between address types such as IPv6, IPv4, even E.164, and so on.

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.

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 tokens.

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

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.

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.

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 [13] 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".

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.

Conditional: A conditional token cell is able to test one of more conditions and invoke processing of successive actions accordingly. Depending on token cell type, these actions may be included in the token cell itself or may be executed via other tokens. This allows to define more complex behavior, such as the invocation of a particular function depending on a dynamic condition encountered at a node.

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

F. Token Cell Processing Engine

The TCR processing model is shown in Fig. 4. 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. In addition, it can read and write to a register that can be used to pass output from the processing of one token cell to the next, thus providing for the previously mentioned token cell pipelining option.

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.

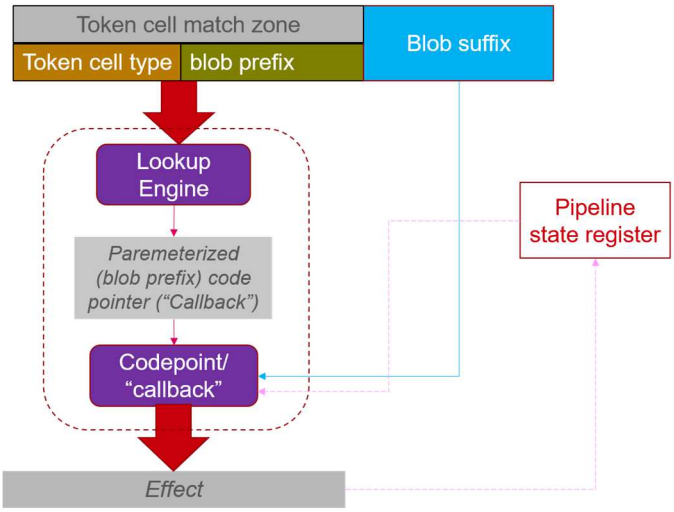

Fig. 4. TCR Token Cell Processing Engine

Where it is not clear from the lookup results what the length of the blob prefix should be, for example when the address is an E164 address, that length needs to be encoded in the prefix in some convenient way, such as a prefix to the prefix. From this, it should be 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.

IV. Example Use Cases

This section contains use cases, showing how composing packets from combinations of token cells leads to interesting behavior. In particular, we show how small modifications or additions of token cells can be used to compose fairly sophisticated behavior in a straightforward way.

A. Latency Based Forwarding

In Latency Based Forwarding [14], the requirement is to forward a packet such that it arrives according to some latency criteria such as arriving at a specific time. Doing so involves special actions at traversed nodes that involve determining a time budget (with both a lower and upper bound) within which to forward the packet, then performing special QoS operations that match queue and packet scheduling against budget. For LBF to be applied, a packet needs to carry the SLO as well as an indicator of latency incurred. This can be accomplished using a corresponding token cell type (for sake of simplicity, carrying all parameters in the token cell itself, not in a separate metadata token cell). A resulting TCR LBF packet is shown in Fig. 5. In this and the following figures, the blue square at the bottom of a token cell points to the disposition token cell. The red square

points to the next token cell. A grey square indicates there is no next token cell.

Token cell T1 is of forwarding type and responsible for specifying the delivery target address in the TCR domain. It specifies T2 as the next token cell to be processed every time T1 is processed. T2 specifies that the packet is to be delivered at a particular time (on time) using LBF, otherwise it is to be dropped. Token cells T1 and T2 are processed at every hop. When the packet arrives at its destination, T1 and T2 are discarded and T3, the disposition token cell, is processed which specifies any further instructions to be executed on the packet at delivery. The payload is carried in T4. If the implementations support it, T1 may be penultimate hop popped (PHP), in which case the first token cell at the destination node will be T2. T2 cannot be PHPed if the LBF policy applies to the last hop.

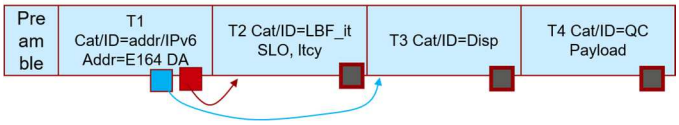

Fig. 5. TCR LBF Packet

B. *Fast ReRoute with Latency Based Forwarding*

Fast re-route (FRR) [15] is a technology that allows productive forwarding of packets to continue following a failure, but before the network has re-converged. For simplicity, FRR is often executed without regard for the pre and post convergence quality of service level requirements. What is much harder to achieve is FRR in which service level objectives are being adhered to whether or not a fast reroute needs to take place. However, it is straightforward to accomplish using TCR. For this purpose, a token cell specifying the SLO (and, for example, LBF action) is combined with token cells specifying the FRR.

As an example, Fig. 6 shows a TCR packet in which both FRR and LBF are supported, very similar to the earlier packet from Fig. 5 but with an additional token cell, T0, that is applied by FRR to indicate the reroute target. Both reroute target (T0) and forwarding destination (T1) point to the SLO token cell as next token cell, which is processed with either route target.

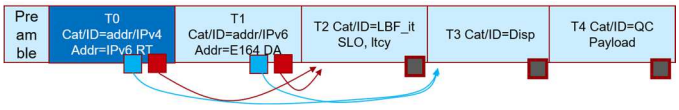

Fig. 6. TCR FRR with LBF Packet

We assume a packet that is using an IPv6 delivery address (T1). Following a failure of a link in the network, the packet is subjected to FRR, in this case, to illustrate address independence, using an IPv4 intermediate node to indicate the FRR target, resulting in T0 (highlighted in dark blue) being added to the packet. Note how the FRR token cell (T0) still uses T2 to specify the LBF behavior. When the packet completes its repair, T0 is popped and the packet is again forwarded based on T1, which continues to use the T2 LBF information, now reflecting any latency updates incurred as a result of the FRR diversion.

C. *Parallel Processing with OAM collection*

In this example (Fig. 7), we consider the case of a packet that is to be forwarded with the collection of OAM telemetry data at each hop. Forwarding lookups take time, and so in this example we parallelize the forwarding lookup and the OAM data collection.

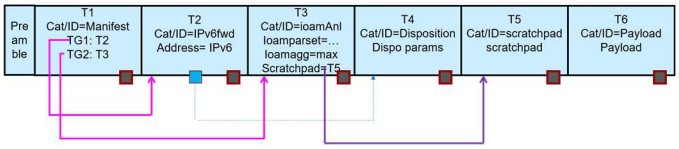

Fig. 7. TCR Parallel Processing: Forwarding and OAM

To cause the parallelization, the first token cell (T1) is a manifest token cell that points to the forwarding token cell (T2) and the OAM token cell (T3), enabling their parallel execution if this is supported by the hardware. The OAM information collected is recorded in the scratchpad token cell, T5. T5 is placed after the disposition token cell T4 to make the information available to the disposition token cell in the event that earlier token cells have been popped before arrival at the disposition node.

D. *Segment Routing with Differentiated LBF*

In this example (Fig. 8) we consider the case of a segment routed (SR) packet that is to be forwarded with LBF QoS, and then develop the design to show how multiple LBF considerations may be implemented.

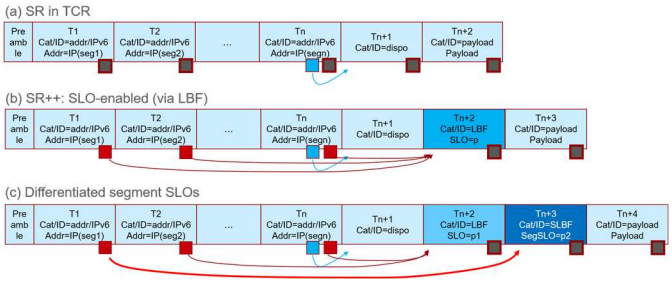

Fig. 8. TCR of Segment-Routed Packets with LBF

In the first example (a) we show an SR packet, which consists of a series of token cells each directing the packet along a segment and then being popped on arrival at the segment endpoint. Upon arrival at the final segment end point, disposition processing takes place. In the second SR example (b), we show the construction of an SR packet with a common LBF policy by including an LBF token cell and setting next token cell for all SR tokens to point to the same LBF token cell for latency policy information. Note that this only requires the addition of a single token cell. Finally, in the third example (c), we include a second LBF token cell and have the next token pointer of the first SR token point to it instead of the other one, thus showing how to introduce differentiated per-segment forwarding policy where different SLOs might apply for different segments. This is fairly sophisticated behavior that would be very difficult to achieve with traditional SR, yet all it takes is the addition of a single additional token cell.

V. Open Issues and Next Steps

TCR is still early work that is at the conceptual stage. The design has not yet been built, although the authors have considerable experience in the design of fast forwarding and

believe that it has significant potential. Up until this point, we have been able to address any new requirement that we have come across within this framework, whether that included the need to optimize token cell processing workflows, to introduce layered security mechanisms able to secure tokens, metadata, and scratchpad data independently, or the support of fast reroute (FRR). We believe that this attests to the flexibility and generality of its design. At the same time, we do expect the design to be easily implementable in hardware, leveraging existing longest prefix matching engines.

Clearly there is a lot of work that remains to be done, beginning with the development of a Proof-of-Concept. Secondly, while we have defined a number of token cell types, to be truly useful and provide users with a rich toolkit to define networking behavior, many more need to be introduced. A third aspect concerns conducting a detailed assessment and analysis of performance aspects, including gains that can be made due to parallelization and processing workflow automation. This will help guide exploration of a fourth aspect, the investigation of new hardware designs that will be capable of supporting such optimizations beyond providing the mere functionality in itself. Other aspects concern possibilities for optimization, such as the combination of manifest and rendezvous token cell types or the support of implementation profiles to account for different device capabilities.

VI. Conclusions

TCR provides a powerful new way to allow users to define sophisticated network service behavior. The definition of behavior can happen, to significant extent, by simply combining token cells of different functions, and by parametrizing those functions. Extensions of behavior are possible by introducing new token cell types, although that requires additional provisioning steps by a network operator.

At the same time, longest prefix match engines can be used as a basis for implementations able to perform at line rate. A rich set of capabilities are supported that are not found in other token or label protocols, such as support for metadata and scratchpads with a lifecycle independent of token cells that access it, support for processing involving workflows beyond what can be expressed by simple pushing to and popping from a label stack, and layered security signatures that allow for differentiated securing of contents as packets traverse nodes along a path.

While lots of work remains to be done, we do believe that TCR has significant potential. As mentioned in the introduction, it is our goal to subject TCR to standardization in the IETF as it matures. We hope to identify partners and collaborators who are interested in exploring TCR jointly further and encourage readers to reach out to us.

References

[1] ITU-T FG-NET2030: New services and capabilities for network 2030: description, technical gap and performance target analysis. FG-NET2030 document NET2030-O-027, 2019.

[2] Bryant, S., and A. Clemm, “Token Cell Routing Data Plane Concepts”, IETF draft-bcx-rtgwg-tcr-00, April 2021.

[3] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

[4] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[5] Bryant, S., Swallow, G., Martini, L., and D. McPherson, "Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN", RFC 4385, DOI 10.17487/RFC4385, February 2006, <https://www.rfc-editor.org/info/rfc4385>.

[6] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <https://www.rfc-editor.org/info/rfc8655>.

[7] Varga, B., Ed., Farkas, J., Berger, L., Malis, A., Bryant, S., and J. Korhonen, "Deterministic Networking (DetNet) Data Plane: MPLS", RFC 8964, DOI 10.17487/RFC8964, January 2021, <https://www.rfc-editor.org/info/rfc8964>.

[8] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/info/rfc8402>.

[9] Li, R., A. Clemm, U. Chunduri, L. Dong, K. Makhijani: “A New Framework and Protocol for Future Networking Applications.” ACM SIGCOMM Workshop on Networking for Emerging Applications and Technologies (NEAT), Budapest, Hungary, August 2018.

[10] Francois, J., A. Clemm, V. Maintenant, S. Tabor: “BPP over P4: Exploring Frontiers and Limits in Programmable Packet Processing.” IEEE Global Communications Conference, December 2020.

[11] Bosshart, P., D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and et al., “P4: Programming protocol-independent packet processors,” SIGCOMM Comput. Commun. Rev., vol. 44, no. 3, p. 87–95, July 2014.

[12] Brockners, F., S. Bhandari, C. Pignataro, H. Gredler, J. Leddy, S. Youell, T. Mizrahi, D. Mozes, P. Lapukhov, R. Chang, D. Bernier, J. Lemon: “Data Fields for In-situ OAM.” Internet Draft draft-ietf-ippm-ioam-data-12, IETF, February 2021.

[13] 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>.

[14] Clemm, A., T. Eckert: *High-Precision Latency Forwarding over Packet-Programmable Networks.* IEEE/IFIP Network Operations and Management Symposium (NOMS 2020), Budapest, Hungary / virtual, April 2020.

[15] Shand, M. and S. Bryant, "IP Fast Reroute Framework", RFC 5714, DOI 10.17487/RFC5714, January 2010, <https://www.rfc-editor.org/info/rfc5714>.