DetNet Working Group
Internet-Draft
Intended status: Informational
Expires: March 2, 2022

Y(J). Stein
RAD

August 29, 2021

Segment Routed Time Sensitive Networking
draft-stein-srtsn-01

Abstract

Routers perform two distinct user-plane functionalities, namely forwarding (where the packet should be sent) and scheduling (when the packet should be sent). One forwarding paradigm is segment routing, in which forwarding instructions are encoded in the packet in a stack data structure, rather than programmed into the routers. Time Sensitive Networking and Deterministic Networking provide several mechanisms for scheduling under the assumption that routers are time synchronized. The most effective mechanisms for delay minimization involve per-flow resource allocation.

SRTSN is a unified approach to forwarding and scheduling that uses a single stack data structure. Each stack entry consists of a forwarding portion (e.g., IP addresses or suffixes) and a scheduling portion (deadline by which the packet must exit the router). SRTSN thus fully implements network programming for time sensitive flows, by prescribing to each router both to-where and by-when each packet should be sent.

Status of This Memo

Copyright Notice

1. Introduction

Packet Switched Networks (PSNs) use statistical multiplexing to fully exploit link data rate. On the other hand, statistical multiplexing in general leads to end-to-end propagation latencies significantly higher than the minimum physically possible, due to packets needing to reside in queues waiting for their turn to be transmitted.

Recently Time Sensitive Networking (TSN) and Deterministic Networking (DetNet) technologies have been developed to reduce this queueing latency for time sensitive packets [RFC8557]. Novel TSN mechanisms are predicated on the time synchronization of all forwarding elements (Ethernet switches, MPLS Label Switched Routers, SDN whitebox switches, or IP routers, to be called here simply routers). Once routers agree on time to high accuracy, it is theoretically possible to arrange for time sensitive packets to experience "green waves", that is, never to wait in queues. For example, scheduling timeslots for particular flows eliminates packet interference, but eliminates the statistical multiplexing advantage of PSNs. In addition, the scheduling calculation and programming of the network to follow this calculation doesn't scale well to large networks.

Segment Routing (SR) technologies provide a scalable method of network programming, but until now has not been applied to scheduling. The SR instructions are contained within a packet in the form of a first-in first-out stack dictating the forwarding decisions of successive routers. Segment routing may be used to choose a path sufficiently short to be capable of providing sufficiently low end-to-end latency but does not influence the queueing of individual packets in each router along that path.

2. Forwarding and Scheduling

Routers (recall that by routers we mean any packet forwarding device) perform two distinct functions on incoming packets, namely forwarding and scheduling. By forwarding we mean obtaining the incoming packet, inspecting the packet’s headers, deciding on an output port, and for QoS routing a specific output queue belonging to this output port, based on the header information and a forwarding information base, optionally editing the packet (e.g., decrementing the TTL field or performing a stack operation on a MPLS label), and placing the packet into the selected output queue.

Scheduling consists of selecting which output queue and which packet from that output queue will be the next packet to be physically transmitted over the output port. In simple terms one can think of forwarding and scheduling as "which output port" and "which packet" decisions, respectively; that is, forwarding decides to which output port to send each packet, and scheduling decides which packet to send next.

Segment routing (as well as connection-oriented mechanisms) slightly simplify the meaning of forwarding to deciding "where" to send the incoming packet, while TSN slightly simplifies the meaning of scheduling to deciding "when" to send the outgoing packet.

Routers optionally perform a third user plane operation, namely per output port and/or per flow traffic conditioning. By conditioning we mean policing (discarding packets based on a token bucket algorithm), shaping (delaying packets), (W)RED, etc. Since we will only be interested in per-packet per router behavior we will neglect conditioning, which is either per router (not distinguishing between packets) or per flow (the same for all routers along the path).

As aforementioned, forwarding decisions always select an output port, but when there are QoS criteria additionally select an output queue belonging to that port. The use of multiple queues per output port is to aid the scheduling, which then becomes a matter of selecting an output queue and always taking the packet at the end of the queue (the packet that has waited the longest). For example, the simplest nontrivial scheduling algorithm is "strict priority". In strict priority packets are assigned to queues according to their priority (as indicated by Priority Code Point or DiffServ Code Point field). The strict priority scheduler always first checks the queue with highest priority; if there is a packet waiting there it is selected for transmission, if not the next highest priority queue is examined and so on. Undesirably strict priority may never reach packets in low priority queues (Best Effort packets), so alternative algorithms, e.g., Weighted Fair Queueing, are used to select from priority queues more fairly.

TSN is required for networks transporting time sensitive traffic, that is, packets that are required to be delivered to their final destination by a given time. In the following we will call the time a packet is sent by the end user application (or the time it enters a specific network) the "birth time", the required delivery time to the end-user application (or the time it exists a specific network) the "final deadline" and the difference between these two times (i.e., the maximally allowed end-to-end propagation time though the network) the "delay budget".

Unlike strict priority or WFQ algorithms, TSN scheduling algorithms may directly utilize the current time of day. For example, in the TSN scheduling algorithm known as time-aware scheduling (gating), each output queue is controlled by a timed gate. At every time only certain output queues have their gates "open" and can have their packets scheduled, while packets are not scheduled from queues with "closed" gates. By appropriately timing the opening and closing of gates of all routers throughout the network, packets in time sensitive flows may be able to traverse their end-to-end path without ever needlessly waiting in output queues. In fact, time-aware gating may be able to provide a guaranteed upper bound for end-to-end delay.

However, time-aware scheduling suffers from two major disadvantages. First, opening the gates of only certain queues for a given time duration, results in this time duration being reserved even if there are very few or even no packets in the corresponding queues. This is precisely the undesirable characteristic of Time Division Multiplexing networks that led to their replacement by Packet Switched Networks. Minimizing time durations increases efficiency, but at the cost of obliging a time sensitive packet that just missed its gate to wait until the next gate opening, endangering its conforming to the delay budget.

In order to avoid such problems, one needs to know a priori the birth times of all time sensitive packets, the lengths of all links between routers, and the loading of all routers. Based on this input one can calculate optimal gating schedules for all routers in the network and distribute this information to all the routers. This calculation is computationally expensive and updating all the routers is communicationally expensive. Moreover, admitting a new time-sensitive flow requires recalculation of all the gating schedules and updating all the routers. This recalculation and communications load is practical only for small networks and a relatively small numbers of flows.

3. Stack-based Methods for Latency Control

One can envision mechanisms for reducing end-to-end propagation latency in a network with time-synchronized routers that do not suffer from the disadvantages of time sensitive scheduling. One such mechanism would be to insert the packet's birth time (time created by end-user application or time entering the network) into the packet's headers. Each router along the way could use this birth time by prioritizing packets with earlier birth times, a policy known as Longest in System (LIS). These times are directly comparable, due to our assuming the synchronization of all routers in the network. This mechanism may indeed lower the propagation delay, but at each router the decision is sub-optimal since a packet that has been in the network longer but that has a longer application delay budget will be sent before a later packet with a tighter delay budget.

An improved mechanism would insert into the packet headers the desired final deadline, i.e., the birth time plus the delay budget. Each router along the way could use this final destination time by prioritizing packets with earlier deadlines, a policy known as Earliest Deadline First (EDF). This mechanism may indeed lower the propagation delay, but at each router the decision is sub-optimal since a packet that has been in the network longer but is close to its destination will be transmitted before a later packet which still has a long way to travel.

A better solution to the problem involves precalculating individual "local" deadlines for each router, and each router prioritizing packets according to its own local deadline. As an example, a packet sent at time 10:11:12.000 with delay budget of 32 milliseconds (i.e., final deadline time of 10:11:12.032) and that needs to traverse three routers might have in its packet headers three local deadlines, 10:11:12:010, 10:11:12.020, and 10:11:12.030. The first router employs EDF using the first local deadline, the second router similarly using the second local deadline, and the ultimate router using the last local deadline.

The most efficient data structure for inserting local deadlines into the headers is a "stack", similar to that used in Segment Routing to carry forwarding instructions. The number of deadline values in the stack equals the number of routers the packet needs to traverse in the network, and each deadline value corresponds to a specific router. The Top-of-Stack (ToS) corresponds to the first router's deadline while the Bottom-of-Stack (BoS) refers to the last's. All local deadlines in the stack are later or equal to the current time (upon which all routers agree), and times closer to the ToS are always earlier or equal to times closer to the BoS.

The stack may be dynamic (as is the forwarding instruction stack in SR-MPLS) or static with an index (as is the forwarding instruction stack in SRv6).

For private networks it is possible for the stack to be inserted by the user equipment that is the source of the packet, in which case the top of stack local deadline corresponds to the first router to be encountered by the packet. However, in such a case the user equipment must also be time synchronized for its time values to be directly compatible. In an improved strategy the stack is inserted into the packet by the ingress router, and thus its deadlines are in concert with time in the network. In such case the first deadline will not explicitly appear in the stack and the initial ToS corresponds to the second router in the network to be traversed by the packet. In either case each router in turn pops from the stack the ToS local deadline and uses that local deadline in its scheduling (e.g., employing EDF).

Since the ingress router inserts the deadline stack into the packet headers, no other router needs to be aware of the requirements of the time sensitive flows. Hence admitting a new flow only requires updating the information base of the ingress router. In an efficient implementation the ingress router's information base has deadline offset vectors for each time sensitive flow. Upon receipt of a packet from user equipment, the ingress router first determines if the packet belongs to a time sensitive flow. If so, it adds the current time to the deadline offset vector belonging to the flow and inserts it as a stack into the packet headers.

An explicit example is depicted in Figure 1. Here packets of a specific time sensitive flow are required to be received by the remote user equipment within 200 microseconds of being transmitted by the source user equipment. The packets traverse a wireless link with delay 2 microseconds to reach the router R1 (the ingress router). They then travel to router R2 over an optical fiber experiencing a propagation delay of 18 microseconds, from there to router R3 experiencing an additional 38 microseconds of fiber delay, from there to router R4 (the egress router) experiencing 16 microseconds of fiber delay. Finally, they travel over a final wireless link taking again 2 microseconds.

```
 +----+  2  +----+  18   +----+  38   +----+  16   +----+  2  +----+
 | UE |-----| R1 |-------| R2 |-------| R3 |-------| R4 |-----| UE |
 +----+     +----+       +----+       +----+       +----+     +----+
```

Figure 1: Example with propagation latencies

We conclude that the total constant physical propagation time is 2+18+38+16+2=76 microseconds. Moreover, assume that we know that in each router there is an additional constant time of 1 microsecond to receive the packet at the line rate and 5 microseconds to process the packet, that is, 6 microseconds per router or 24 microseconds for all four routers. We have thus reached the conclusion that the minimal time to traverse the network is 76+24=100 microseconds

Since our delay budget is 200 microseconds, we have spare time of 200-100=100 microseconds for the packets to wait in output queues. If we have no further information, we can divide this spare 100 microseconds equally among the 4 routers, i.e., 25 microseconds per router. Thus, the packet arrives at the first router after 2 microseconds, is received and processed after 2+6=8 microseconds, and is assigned a local deadline to exit the first router of 8+25=33 microseconds. The worst case times of arrival and transmission at each point along the path are depicted in Figure 2. Note that in general it may be optimal to divide the spare time in unequal fashion.

```
 +----+  2  +----+  18   +----+  38   +----+  16   +----+  2  +----+
 | UE |-----| R1 |-------| R2 |-------| R3 |-------| R4 |-----| UE |
 +----+     +----+       +----+       +----+       +----+     +----+
       |   |      |     |      |     |      |     |      |   |
       |   |      |     |      |     |      |     |      |   |
       0   2     33    51     82   120    151   167    198 200
```

Figure 2: Example with worst case times

Assuming that the packet left router 1 the full 33 microseconds after its transmission, it will arrive at router 2 after an additional 18 microseconds, that is, after 51 microseconds. After the mandatory 6 microseconds of reception and processing and the 25 microseconds allocated for queueing, we reach the local deadline to exit router 2 by 82 microseconds. Similarly, the local deadline to exit router 3 is 151 microseconds, and the deadline to exit router 4 is 198 microseconds. After the final 2 microseconds consumed by the wireless link the packet will arrive at its destination after 200 microseconds as required

Based on these worst case times the ingress router can now build the deadline offset vector (33, 82, 151, 198) referenced to the time the packet left the source user equipment, or referenced to the time the packet arrives at the ingress router of (31, 80, 149, 196).

Now assume that a packet was transmitted at time T and hence arrives at the ingress router at time T + 2 microseconds. The ingress router R1, observing the deadline offset vector referenced to this time, knows that the packet must be released no more than 31 microseconds later, i.e., by T + 33 microseconds. It furthermore inserts a local deadline stack [T+82, T+151, T+198] into the packet headers.

The second router R2 receives the packet with the local deadline stack, pops the ToS revealing that it must ensure that the packet exits by T + 82 microseconds. It properly prioritizes and sends the packet with the new stack [T+151, T+198]. Router R3 pops deadline T+151, and sends the packet with local deadline stack containing a single entry [T+198]. The final router pops this final local deadline and ensures that the packet is transmitted before that time The local deadline stacks are depicted in Figure 3.

```
    +----+  2  +----+  18   +----+  38   +----+  16   +----+  2  +----+
    | UE |-----| R1 |-------| R2 |-------| R3 |-------| R4 |-----| UE |
    +----+     +----+   |   +----+   |   +----+   |   +----+     +----+
          |   |      |  |  |      |  |  |      |  |  |      |   |
          |   |      |  |  |      |  |  |      |  |  |      |   |
          0   2     33  | 51     82  | 120    151 | 167    198 200
                        |            |            |
                        V            V            V
                      +---+        +---+        +---+
                      | 82|        |151|        |198|
                      |---|        |---|        +---+
                      |151|        |198|
                      |---|        +---+
                      |198|
                      +---+
```

Figure 3: Example with local deadline stacks

The precise mechanism just described is by no means the only way to compute local deadlines. Furthermore, combining time-aware scheduling solely at the ingress router, with EDF at all the other routers, can provide "green waves" with provable upper bounds to delay. However, optimizing such a scheme at scale may still be challenging. A randomized algorithm for setting up such a case is described in [AndrewsZhang].

4. The Time Sensitive Router

While a stack is the ideal data structure to hold the local deadlines in the packet, different data structures are used to hold the time sensitive packets (or their descriptors) in the routers. The standard data structure used in routers is the queue which, being a first in first out memory, is suitable for a policy of first-to-arrive first-to-exit, and not for EDF or other stack-based time sensitive mechanisms. More suitable data structures are sorted lists, search trees, and priority heaps. While such data structures are novel in this context, efficient hardware implementations exist.

If all the time sensitive flows are of the same priority, then a single such data structure may be used for all time sensitive flows. If there are time sensitive flows of differing priorities, then a separate such data structure is required for each level of priority corresponding to a time sensitive flow, while the conventional queue data structure may be used for priority levels corresponding to flows that are not time sensitive.

For example, assume two different priorities of time sensitive flows and a lower priority for Best Effort traffic that is not time sensitive. If applying strict priority the scheduler would first check if the data structure for the highest priority contains any packets. If yes, it transmits the packet with the earliest local deadline. If not, it checks the data structure for the second priority. If it contains any packets it transmits the packet with the earliest deadline. If not, it checks the Best Effort queue. If this queue is nonempty it transmits the next packet in the queue, i.e., the packet that has waited in this queue the longest.

Separate prioritization and EDF is not necessarily the optimal strategy. An alternative (which we call Liberal EDF, or LEDF) would be for the scheduler to define a worst case (i.e., maximal) packet transmission time MAXTT (for example, the time taken for a 1500 byte packet to be transmitted at the output port's line rate). Instead of checking whether the data structure for the highest priority contains any packets at all, LEDF checks whether its earliest packet's local deadline is earlier than MAXTT from the current time. If it is, it is transmitted; if it is not the next priority is checked, knowing that even were a maximal size packet to be transmitted the scheduler will still be able to return to the higher priority packet before its local deadline.

5. Segment Routed Time Sensitive Networking

Since Segment Routing and the TSN mechanism just described both utilize stack data structures it is advantageous to combine their information into a single unified SRTSN stack. Each entry in this stack contain two subentries, the forwarding instruction (e.g., the address of the next router or the label specifying the next link) and a scheduling instruction (the local deadline).

Each SRTSN stack entry fully prescribes the forwarding and scheduling behavior of the corresponding router, both to-where and by-when the packet should be sent. The insertion of a stack into packets thus fully implements network programming for time sensitive flows.

For example, Figure 4 depicts the previous example but with the unified SRTSN stacks. Ingress router R1 inserts a SRTSN stack with three entries into the packet received. In this example the forwarding sub-entry contains the identifier or address of the next router, except for the Bottom of Stack entry that contains a special BoS code (e.g., identifier zero). The ToS entry thus contains the address of router R3 and the time by which the packet must exit router R2, namely T + 82 microseconds. Router R2 pops this ToS leaving a SRTSN stack with 2 entries. Router R3 pops the new ToS instructing it to forward the packet to router R4 by time T + 151 microseconds, leaving a stack with a single entry. Router R4 pops the ToS and sees that it has reached bottom of stack. It then forwards the packet according to the usual rules of the network (for example, according to the IP address in the IP header) by local deadline T + 198 microseconds.

```
    +----+  2  +----+  18   +----+  38   +----+  16   +----+  2  +----+
    | UE |-----| R1 |-------| R2 |-------| R3 |-------| R4 |-----| UE |
    +----+     +----+   |   +----+   |   +----+   |   +----+     +----+
          |   |      |  |  |      |  |  |      |  |  |      |   |
          |   |      |  |  |      |  |  |      |  |  |      |   |
          0   2     33  | 51     82  | 120    151 | 167    198 200
                        |            |            |
                        V            V            V
                    +-------+    +-------+    +-------+
                    |R2;  82|    |R3; 151|    |BoS;198|
                    |-------|    |-------|    +-------+
                    |R3; 151|    |BoS;198|
                    |-------|    +-------+
                    |BoS;198|
                    +-------+
```

Figure 4: Example with combined SRTSN stacks

6. Stack Entry Format

A number of different time formats are in common use in networking applications and can be used to encode the local deadlines. The longest commonly utilized format is 80-bit PTP-80 timestamp defined in IEEE 1588v2 Precision Time Protocol [IEEE1588]. There are two common 64-bit time representations: the NTP-64 timestamp defined in [RFC5905] (32 bits for whole seconds and 32 bits for fractional seconds); and the PTP-64 timestamp (32 bits for whole seconds and 32 bits for nanoseconds). Finally, there is the NTP-32 timestamp (16 bits of whole seconds and 16 bits of fractional seconds) that is often insufficient due to its low resolution (15 microseconds).

However, we needn't be constrained by these common formats, since our wraparound requirements are minimal. As long as we have no ambiguity in times during the flight of a packet, which is usually much less than a second, the timestamp is acceptable. Thus, we can readily use a nonstandard 32-bit timestamp format with say 12 bits of seconds (wraparound over 1 hour) and 20 bits for microseconds, or say 8 bits for whole seconds (wraparound over 4 minutes) and 24 bits of tenths of microseconds.

For the forwarding sub-entry we could adopt like SR-MPLS standard 32-bit MPLS labels (which contain a 20-bit label and BoS bit), and thus SRTSN stack entries could be 64-bits in size comprising a 32-bit MPLS label and the aforementioned nonstandard 32-bit timestamp. Alternatively, an SR-TSN stack entry could be 96 bits in length comprising a 32-bit MPLS label and either of the standardized 64-bit timestamps.

For IPv4 networks one could employ a 32-bit IPv4 address in place of the MPLS label. Thus, using the nonstandard 32-bit timestamp the entire stack entry could be 64 bits. For dynamic stack implementations a BoS bit would have to be included.

SRv6 uses 128-bit IPv6 addresses (in addition to a 64-bit header and possibly options), and so 160-bit or 192-bit unified entries are directly derivable. However, when the routers involved are in the same network, address suffixes suffice to uniquely determine the next router.

7. Stack Size

We can now address the question of the total overhead added by the SRTSN stack. Were each stack entry to comprise a 128-bit IPv6 address and a 64-bit timestamp then each stack entry would consume 24 bytes! In such a case a 10-hop stack would be larger than an average IPv4 packet.

But we needn't be so wasteful! Our deadline wraparound requirements are minimal as a timestamp is unambiguous when the wraparound duration exceeds twice the maximum time path time. In a single network the forwarding sub-entry may consist of a router address suffix, or even an index uniquely identifying each router. In fact, it is easy to see that each entry need only be ceil(log2(Nrouters)) + ceil(log2(2 max-path-time / time-resolution)) + 1 bits in duration.

For small networks this translates to about 16 bits per entry and for medium sized ones 32 bits per entry. So, an entire 4-hop stack may still occupy about as much as a single IPv6 address!

8. Control Plane

In the above discussion we assumed that the ingress router knows the deadline offset vector for each time sensitive flow. This vector may be calculated by a centralized management system and sent to the ingress router, or may be calculated by the ingress router itself.

In the former case there is central SRTSN orchestrator, which may be based on a Network Management System, or on an SDN controller, or on a Path Computation Element server. The SRTSN orchestrator needs to be know the propagation delays for all the links in the network, which may be determined using time domain reflectometry, or via one-way delay measurement OAM, or retrieved from a network planning system. The orchestrator may additionally know basic parameters of the routers, including minimal residence time, data rate of the ports, etc. When a time sensitive path needs to be set up, the SRTSN orchestrator is given the source and destination and the delay budget. It first determines feasibility by finding the end-to-end delay of the shortest path (shortest being defined in terms of latency, not hop count). It then selects a path (usually, but not necessarily, the shortest one) and calculates the deadline offset vector. The forwarding instructions and offset vector (as well as any other required flow-based information, such as data rate or drop precedence) are then sent to the ingress router. As in segment routing, no other router in the network needs to be informed.

In the latter case the ingress router is given the destination and the delay budget. It sends a setup message to the destination as in RSVP-TE, however, in this case arrival and departure timestamps are recorded for every router along the way. The egress router returns the router addresses and timestamps. This process may be repeated several times and minimum gating applied to approximate the link propagation times. Assuming that the path’s delay does not exceed the delay budget, the path and deadline offset vector may then be determined.

The method of [AndrewsZhang] uses randomization in order to avoid the need for centralized coordination of flows entering the network at different ingress routers. However, this advantage comes at the expense of much higher achievable delay budgets.

9. Security Considerations

SRTSN concentrates the entire network programming semantics into a single stack, and thus tampering with this stack would have devastating consequences. Since each stack entry must be readable by its corresponding router, protecting the stack would necessitate key distribution between the ingress router and every router along the path.

A simpler mechanism would be for the ingress router to sign the stack with a public key known to all routers in the network, and to append this signature to the stack. If the signature is not present or is incorrect the packet should be discarded.

10. IANA Considerations

This document requires no IANA actions.

11. Informative References

[AndrewsZhang]
Andrews, M. and L. Zhang, "Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule", Journal of Algorithms 52 57-81, 2003.

[IEEE1588]
IEEE, "Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE 1588-2008, DOI 10.1109/IEEESTD.2008.4579760, 2008.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010.

[RFC8557] Finn, N. and P. Thubert, "Deterministic Networking Problem Statement", RFC 8557, DOI 10.17487/RFC8557, May 2019.

Author's Address

Yaakov (J) Stein
RAD

Email: yaakovjstein@gmail.com