Abstract

This document presents the framework and key methods for Large-scale Deterministic Networks (LDN). It achieves scalability for the number of supportable deterministic traffic flows via Scalable Deterministic Forwarding (SDF) that does not require per-flow state in transit nodes and precise time synchronization among nodes. It achieves Scalable Resource Reservation (SRR) by allowing for it to be decoupled from the forwarding plane nodes, and aggregating resource reservation status in time slots.
1. Introduction

Deploying deterministic service over large-scale network will face some technical challenges, such as

- massive number of deterministic flows vs. per-flow operation and management;
- long link propagation may bring in significant jitter;
- time synchronization is hard to be achieved among numerous devices, etc.
Motivated by these challenges, this document presents a Large-scale Deterministic Network (LDN) system, which consists of Scalable Deterministic Forwarding (SDF) at forwarding plane and Scalable Resource Reservation (SRR) at control plane. The technologies of SDF and SRR can be used independently.

As [draft-ietf-detnet-problem-statement] indicates, deterministic forwarding can only apply on flows with well-defined traffic characteristics. The traffic characteristics of DetNet flow has been discussed in [draft-ietf-detnet-architecture], that could be achieved through shaping at Ingress node or up-front commitment by application. This document assumes that DetNet flows follow some specific traffic patterns accordingly.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.2. Terminology & Abbreviations

This document uses the terminology defined in [draft-ietf-detnet-architecture].

TSN: Time Sensitive Network
CQF: Cyclic Queuing and Forwarding
LDN: Large-scale Deterministic Network
SDF: Scalable Deterministic Forwarding
SRR: Scalable Resource Reservation
DSCP: Differentiated Services Code Point
EXP: Experimental
TC: Traffic Class
T: the length of a cycle
H: the number of hops
K: the size of aggregated resource reservation window
2. Overview

2.1. Summary

The Large-Scale Deterministic Network solution (LDN) consists of two parts: The Scalable Deterministic Forwarding Plane (SDF) as its forwarding plane and the Scalable Resource Reservation (SRR) as its control plane. In the SDF, nodes in the network have synchronized frequency, and each node forwards packets in a slotted fashion based on a cycle identifiers carried in packets. Ingress nodes or senders have a function called gate to shape/condition traffic flows. Except for this gate function, the SDF has no awareness of individual flows. The SRR maintains resource reservation states for deterministic flows, Ingress nodes maintain per-flow states and core nodes aggregate per-flow states in time slots.

2.2. Background

This section motivates the design choices taken by the proposed solution and gives the necessary background for deterministic delay based forwarding plane designs.

2.2.1. Deterministic End-to-End Latency

Bounded delay is delay that has a deterministic upper and lower bound.

The delay for packets that need to be forwarded with deterministic delay needs to be deterministic on every hop. If any hop in the network introduces non-deterministic delay, then the network itself can not deliver a deterministic delay service anymore.

2.2.2. Hop-by-Hop Delay

Consider a simple example (without picture), where N has 10 receiving interfaces and one outgoing interface I all of the same speed. There are 10 deterministic traffic flows, each consuming 5% of a link's bandwidth, one from each receiving interface to the outgoing interface.

Node N sends ‘only’ 50% deterministic traffic to interface I, so there is no ongoing congestion, but there is added delay. If the arrival time of packets for these 10 flows into N is uncontrolled, then the worst case is for them to all arrive at the same time. One packet has to wait in N until the other 9 packets are sent out on I, resulting in a worst case deterministic delay of 9 packets serialization time. On the next hop node N2 downstream from N, this problem can become worse. Assume N2 has 10 upstream nodes like N,
the worst case simultaneous burst of packets is now 100 packets, or a 99 packet serialization delay as the worst case upper bounded delay incurred on this hop.

To avoid the problem of high upper bound end-to-end delay, traffic needs to be conditioned/interleaved on every hop. This allows to create solutions where the per-hop-delay is bounded purely by the physics of the forwarding plane across the node, but not the accumulated characteristics of prior hop traffic profiles.

2.2.3. Cyclic Forwarding

The common approach to solve that problem is that of a cyclic hop-by-hop forwarding mechanism. Assume packets forwarded from N1 via N2 to N3 as shown in Figure 1. When N1 sends a packet P to interface I1 with a Cycle X, it must be guaranteed by the forwarding mechanism that N2 will forward P via I2 to N3 in a cycle Y.

The cycle of a packet can either be deduced by a receiving node from the exact time it was received as is done in SDN/TDMA systems, and/or it can be indicated in the packet. This document solution relies on such markings because they allow to reduce the need for synchronous hop-by-hop transmission timings of packets.

In a packet marking based slotted forwarding model, node N1 needs to send packets for cycle X before the latest possible time that will allow for N2 to further forward it in cycle Y to N3. Because of the marking, N1 could even transmit packets for cycle X before all packets for the previous cycle (X-1) have been sent, reducing the synchronization requirements between across nodes.

```
P sent in          P sent in         P sent in
  cycle(N1,I1,X)  cycle(N2,I2,Y)  cycle(N3,I3,Z)
  +--------+        +--------+         +--------+
  | Node N1|------->| Node N2|-------->| Node N3|---->
  +--------+I1      +--------+I2       +--------+I3
```

Figure 1: Cyclic Forwarding

2.2.4. Co-Existence with Non-Deterministic Traffic

Traffic with deterministic delay requirements can co-exist with traffic only requiring non-deterministic delay by using packet scheduling where the delay incurred by non-deterministic packets is deterministic for the deterministic traffic (and low). If LDN SDF is deployed together with such non-deterministic delay traffic than such a scheme must be supported by the forwarding plane. A simple approach for the delay incurred on the sending interface of a
deterministic node due to non-deterministic traffic is to serve
deterministic traffic via a strict, highest-priority queue and
include the worst case delay of a currently serialized non-
deterministic packet into the deterministic delay budget of the node.
Similar considerations apply to the internal processing delays in a
node.

2.3. System Components

The Figure 2 shows an overview of the components considered in this
document system and how they interact.

A network topology of nodes, Ingress, Core and Egress support a
method for cyclic forwarding to enable Scalable Deterministic
Forwarding (SDF). This forwarding requires no per-flow state on the
nodes.

Ingress edge nodes may support the (G)ate function to shape traffic
from sources into the desired traffic characteristics, unless the
source itself has such function. Per-flow state is required on the
ingress edge node.

A Scalable Resource Reservation (SRR) works as control plane. It
records reserved resources for deterministic flows. Per-flow state
is maintained on the ingress edge node, and aggregated state is
maintained on core node.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{system_overview.png}
\caption{System Overview}
\end{figure}
3. Scalable Deterministic Forwarding

DetNet aims at providing deterministic service over large scale network. In such large scale network, it is difficult to get precise time synchronization among numerous devices. To reduce requirements, the forwarding mechanism described in this document assumes only frequency synchronization but not time synchronization across nodes: nodes maintain the same clock frequency $1/T$, but do not require the same time as shown in Figure 3.

```
|<---T---->     |<---T---->|
|       |       |       |       |
Node A +-----------------+ Node A +-----------------+ T0 T0
       |       |       |       |
|<---T---->     |<---T---->|
|       |       |       |       |
Node B +-----------------+ Node B +-----------------+ T0 T0
```

(i) time synchronization   (ii) frequency synchronization

T: length of a cycle
T0: timestamp

Figure 3: Time Synchronization & Clock Synchronization

IEEE 802.1 CQF is an efficient forwarding mechanism in TSN that guarantees bounded end-to-end latency. CQF is designed for limited scale networks. Time synchronization is required, and the link propagation delay is required to be smaller than a cycle length $T$. Considering the large scale network deployment, the proposed Scalable Deterministic Forwarding (SDF) permits frequency synchronization and link propagation delay may exceed $T$. Besides these two points, CQF and the asynchronous forwarding of SDF are very similar.

Figure 4 compares CQF and SDF through an example. Suppose Node A is the upstream node of Node B. In CQF, packets sent from Node A at cycle $x$, will be received by Node B at the same cycle, then further be sent to downstream node by Node B at cycle $x+1$. Due to long link propagation delay and frequency synchronization, Node B will receive packets from Node A at different cycle denoted by $y$ in the SDF, and Node B swaps the cycles carried in packets with $y+1$, then sends out those packets at cycle $y+1$. This cycle mapping (e.g., $x \rightarrow y+1$) exists between any pair of neighbor nodes. With this mapping, the receiving node can easily figure out when the received packets should be sent out, the only requirement is to carry the cycle identifier of sending node in the packets.
3.1. Three Queues

In CQF each port needs to maintain 2 (or 3) queues: one is used to buffer newly received packets, another one is used to store the packets that are going to be sent out, one more queue may be needed to avoid output starvation [scheduled-queues]. In SDF, at least 3 queues are needed.

As Figure 5 illustrated, a node may receive packets sent at two different cycles from a single upstream node due to the absence of time synchronization. Following the cycle mapping (i.e., \( x \rightarrow y+1 \)), packets that carry cycle identifier \( x \) should be sent out by Node B at cycle \( y+1 \), and packets that carry cycle identifier \( x+1 \) should be sent out by Node B at cycle \( y+2 \). Therefore, two queues are needed to store the newly received packets, as well as one queue to store the sending packets. In order to absorb more link delay variation (such as on radio interface), more queues may be necessary.

![Figure 4: CQF & SDF](image)

(i) CQF
(ii) SDF

![Figure 5: Three Queues in SDF](image)
3.2. Cycle Mapping

When this packet is received by Node B, some methods are possible how the forwarding plane could operate. In one method, Node B has a mapping determined by the control plane. Packets from (the link from) Node A indicating cycle x are mapping into cycle y+1. This mapping is necessary, because all the packets from one cycle of the sending node need to get into one cycle of the receiving node. This is called "configured cycle mapping".

Instead of configuring an explicit cycle mapping such as cycle x -> cycle y+1, the receiving Node B could also have the intelligence in the forwarding plane to recognize the first packet from (the link from) Node A that has a new cycle x number, and map this cycle x to the next cycle after the current cycle y, aka: cycle y+1. We call this option "self synchronized cycle mapping".

3.2.1. Cycle Identifier Carrying

In self synchronized cycle mapping, cycle identifier needs to be carried in the SDF packets, so that an appropriate queue can be selected accordingly. That means 2 bits are needed in the three queues model of SDF, in order to identify different cycles between a pair of neighboring nodes. There are several ways to carry this 2 bits cycle identifier. This document does not yet aim to propose one, but gives an (incomplete) list of ideas:

- DSCP of IPv4 Header
- Traffic Class of IPv6 Header
- TC of MPLS Header (used to be EXP)
- EtherType of Ethernet Header
- IPv6 Extension Header
- TLV of SRv6
- TC of MPLS-SR Header (used to be EXP)
- Three labels/adjacency SIDs for MPLS-SR

4. Scalable Resource Reservation

SDF must work with some resource reservation mechanisms, that can fulfill the role of the Scalable Resource Reservation (SRR). This resource reservation guarantees the necessary network resources when
deterministic flows are scheduled including the slots through which the traffic travels hop-by-hop. Network nodes have to record how many network resources are reserved for a specific flow from when it starts to when it ends (e.g., \(<\text{flow\_identifier}, \text{reserved\_resource}, \text{start\_time}, \text{end\_time}>\)). Maintaining per-flow resource reservation state may be acceptable to edge nodes, but unacceptable to core nodes. [draft-ietf-detnet-architecture] pointed out that aggregation must be supported for scalability.

SRR aggregates per-flow resource reservation states for each time slot:

1. Dividing time into time slots. Then the per-flow resource reservation states can be expressed as \(<\text{flow\_identifier}, \text{reserved\_resource}, \text{start\_time\_slot}, \text{end\_time\_slot}>\) accordingly. Note that time slot here is irrelevant to the cycle in SDF.

2. Edge node still maintains per-flow resource reservation states. While core node calculates and maintains the sum of reserved\_resources (or remaining resources) of each time slot. That is a core node just needs to maintain a variable for each time slot. Suppose that a core node can maintain K time slots’ results, i.e., the aggregated resource reservation window of a core node is K.

3. New resource reservation request succeed only if there are sufficient resources along the path. Resource is reserved in unit of time slot, and at most K time slots. If more than K time slots’ resources are needed, edge node/host can send renewal request before the expiration of K time slots. Edge node/host also can active teardown the resource reservation along the path.

4. Core nodes refresh their aggregated resource reservation windows according to the per-flow resource reservation states maintained by edge nodes.

5. Performance Analysis

5.1. Queueing Delay

We consider forwarding from an LDN node A via an LDN node B to an LDN node C and call the single-hop LDN delay the time between a packet being sent by A and the time it is re-sent by B. This single-hop delay is composed from the A->B propagation delay and the single-hop queuing delay A->B.
As Figure 6 shows, cycle x of Node A will be mapped into cycle y+1 of Node B as long as the last packet sent from A->B is received within the cycle y. If the last packet is re-sent out by B at the end of cycle y+1, then the largest single-hop queueing delay is 2*T. Therefore the end-to-end queueing delay’s upper bound is 2*T*H, where H is the number of hops.

If A did not forward the LDN packet from a prior LDN forwarder but is the actual traffic source, then the packet may have been delayed by a gate function before it was sent to B. The delay of this function is outside of scope for the LDN delay considerations. If B is not forwarding the LDN packet but the final receiver, then the packet may not need to be queued and released in the same fashion to the receiver as it would be queued/released to a downstream LDN node, so if a path has one source followed by N LDN forwarders followed by one receivers, this should be considered to be a path with N-1 LDN hops for the purpose of latency and jitter calculations.

5.2. Jitter

Considering the simplest scenario one hop forwarding at first, suppose Node A is the upstream node of Node B, the packet sent from Node A at cycle x will be received by Node B at cycle y as Figure 7 shows.

- The best situation is Node A sends packet at the end of cycle x, and Node B receives packet at the beginning of cycle y, then the delay is denoted by w;
- The worst situation is Node A sends packet at the beginning of cycle x, and Node B receives packet at the end of cycle y, then the delay= w + length of cycle x + length of cycle y= w+2*T;
- Hence the jitter’s upper bound of this simplest scenario= worst case-best case=2*T.

Next considering two hops forwarding as Figure 8 shows.

- The best situation is Node A sends packet at the end of cycle x, and Node C receives packet at the beginning of cycle z, then the delay is denoted by w’;

- The worst situation is Node A sends packet at the beginning of cycle x, and Node C receives packet at the end of cycle z, then the delay= w’ + length of cycle x + length of cycle z= w’+2*T;

- Hence the jitter’s upper bound = worst case-best case=2*T.
And so on. For multi-hop forwarding, the end-to-end delay will increase as the number of hops increases, while the delay variation (jitter) still does not exceed $2T$.

6. IANA Considerations

This document makes no request of IANA.

7. Security Considerations

Security issues have been carefully considered in [draft-ietf-detnet-security]. More discussion is TBD.

8. Acknowledgements

TBD.
9. Normative References

[draft-ietf-detnet-architecture]

[draft-ietf-detnet-dp-sol]

[draft-ietf-detnet-problem-statement]

[draft-ietf-detnet-security]

[draft-ietf-detnet-use-cases]


[scheduled-queues]

Authors’ Addresses

Li Qiang (editor)
Huawei
Beijing
China

Email: qiangli3@huawei.com