# Mitigating Some Security Attacks in MPLS-VPN Model "C"

Shankar Raman*, Balaji Venkat†, and Gaurav Raina†
India-UK Advanced Technology Centre of Excellence in Next Generation Networks
*Department of Computer Science and Engineering, †Department of Electrical Engineering
Indian Institute of Technology Madras, Chennai 600 036, India
Email: mjsraman@cse.iitm.ac.in, balajivenkat299@gmail.com, gaurav@ee.iitm.ac.in

***Abstract*—In certain models of inter-provider Multi-Protocol Label Switching (MPLS) based Virtual Private Networks (VPNs), spoofing and replay attacks against VPN sites are two key concerns. MPLS VPN model "C" can scale well with respect to maintenance of routing state when compared with models "A" and "B". But this deployment model is not favoured due to the aforementioned security concerns in the data-plane. The inner labels associated with VPN sites are not encrypted during data transmission. Therefore it is possible for an attacker to spoof or replay data packets to a specific VPN site. We propose a label-hopping technique which uses a set of randomised labels and a method for hopping amongst these labels to address these type of attacks. To reduce the computation time complexity for such algorithms, we propose the use of Timing over Internet Protocol connection and Transfer of Clock (TicToc) based Precision Time Protocol. Simulations show that by using the TicToc protocol, along with the label-hopping technique, we can mitigate spoofing and replay attacks at line-rate. As we address key security and performance concerns, we make a plausible case for the deployment of MPLS based VPN inter-provider model "C".**

***Index Terms*—MPLS; VPN; Model "C"; Label-hopping; Spoofing attack; Replay attack.**

## I. Introduction

Mitigating spoofing and replay attacks in Multi-Protocol Label Switching - Virtual Private Networks (MPLS-VPNs) is a key concern [1]. MPLS [2] technology uses fixed size labels to forward data packets between routers. Specific customer services (for example, Layer 3 (L3)-VPNs based on Border Gateway Protocol (BGP) extensions), can be deployed by stacking the labels. BGP-based MPLS L3-VPN services are provided either on a single Internet Service Provider (ISP) core or across multiple ISP cores. The latter cases are known as inter-provider MPLS VPNs, which are broadly categorised and referred to as models "A", "B" and "C" [3].

Model "A" uses back-to-back VPN Routing and Forwarding (VRF) connections between Autonomous System Border Routers (ASBRs). Model "B" uses exterior BGP (eBGP) redistribution of labelled VPN Internet Protocol version 4 (IPv4) routes from Autonomous Systems (AS) to neighbouring AS. Model "C" uses multi-hop Multi-Protocol (MP)-eBGP redistribution of labelled VPN IPv4 routes and eBGP redistribution of IPv4 routes from an AS to a neighbouring AS. Model "C" is scalable for maintaining routing states and hence preferred for deployment in the Internet [4]. Security issues in MPLS, especially MPLS-based VPNs, continue to attract attention [5].

The security of model "A" matches the single-AS standard proposed in [6]. Model "B" can be secured on the control-plane, but on the data-plane the validity of the outer-most label (Label Distribution or Resource Reservation Protocol label) is not checked. This weakness could be exploited to inject crafted packets from inside an MPLS network. A solution for this problem is proposed in [4]. Model "C" can be secured on the control-plane but has a security weakness on the data-plane. The ASBRs do not have any VPN information and hence the inner-most label cannot be validated. In this case, the solution used for model "B" cannot be applied. An attacker can exploit this weakness to send unidirectional packets into the VPN sites connected to the other AS. Therefore, Internet Service Providers (ISPs) using model "C" must either trust each other or not deploy it [7]. A simple solution to this problem is to filter all IP traffic with the exception of the required eBGP peering between the ASBRs, thereby preventing a large number of potential IP traffic-related attacks. However, controlling labelled packets is difficult. In model "C", there are at least two labels for each packet: the Provider Edge (PE) label, which defines the Label Switched Path (LSP) to the egress PE, and the VPN label, which defines the VPN associated with the packet on the PE.

Control-plane security issue in model "C" can be resolved by using IPSec [8]. The authors propose an IPSec encryption technique for securing the PE of the network. The authors also highlight that the processing capacity could be over-burdened. Further, if IPSec is used in the data-plane then configuring and maintaining key associations could be difficult. If an attacker is located at the core of the network, or in the network between the providers that constitute an inter-provider MPLS VPN, then spoofing is possible. The vulnerability of MPLS against spoofing attacks and the impact on performance of IPSec has been discussed in [9]. If the inner labels that identify packets going towards a L3-VPN site are spoofed, then sensitive information related to services available within the organisational servers can be compromised.

The algorithm previously proposed by us to mitigate spoofing attacks is an $O(N)$ algorithm, where $N$ represents the payload size chosen for hashing [1]. However, using payload to obtain the hash value can encourage replay attacks on a VPN site. It should be noted that the labels used in the

label-hopping algorithm are valid only for a certain period of time. An attacker could resend a valid data packet within this time period. The label-hopping algorithm accepts such packets. Such an attack reduces the network performance as redundant data packets get processed repeatedly. A simple way to solve this problem is to include a sequence number with every packet, but this increases the payload size. Therefore label-hopping with hashing based on payload cannot be used to provide protection against replay attacks.

In this paper, we expand the work presented in [1] in the following ways:

1) We use Timing over IP Connection and Transfer of Clock (TicToc) to achieve label synchronisation and hence mitigate replay as well as spoofing attacks.
2) We show that use of TicToc, hashing and pseudo-random number generators to mitigate replay attacks leads to a constant time ($O(1)$) computational time complexity increase to the algorithm that mitigates spoofing attacks.

Additionally, we show that the computational time complexity of the label-hopping algorithm can be reduced from $O(N)$ to $O(1)$ by using time-based synchronisation techniques like Network Time Protocol (NTP) or TicToc. Such methods will be useful in a real time data transfer scenario in MPLS VPNs as they incur very low processing overhead. The advantage of the proposed scheme is that it can be used wherever MP-eBGP multi-hop scenarios arise. We also show that the proposed method can reduce the burden on Deep Packet Inspection Engines (DPIEs), but can contribute towards more processing time for ISP's billing schemes. As far as we know, no ISP has implemented MPLS VPN model "C". Large scale deployment of this model has been avoided due to security concerns. The methods proposed in this paper make a case for the potential deployment of MPLS VPN model "C' by ISPs.

The rest of the paper is organised as follows. In Section II, we discuss the pre-requisites of the proposed scheme. Section III reviews the label-hopping technique. In Section IV, we present a method by which the computational time complexity can be reduced using TicToc. In Section V, we present algorithms that protect model "C" against spoofing and replay attacks. In Section VI, we present our simulation results. Some of the implementation issues are discussed in Section VII. In Section VIII, we present the impact of label-hopping scheme on two applications; namely deep packet inspection engine and ISP's billing. Section IX outlines our contributions, and highlights some avenues for further work.

## II. Pre-requisites for the label-hopping scheme

We briefly review the network topology for model "C", the PE configuration and the control-plane exchanges needed for the proposed scheme.

### A. MPLS VPN model "C"

The reference MPLS-eBGP based VPN network for model "C" as described in [10] is shown in Figure 1, along with the control-plane exchanges. A legend for Figure 1 is given in Table I. The near-end PE ($PE_{ne}$) and far-end PE ($PE_{fa}$) are connected through the inter-provider MPLS based core network. The VPN connectivity is established through a set of routers from different AS and their ASBRs. In the VPN, MP-eBGP updates are exchanged for a set of Forward Equivalence Classes (FECs). These FECs, which have to be protected, originate from the prefixes behind $PE_{ne}$ in a VPN site or a set of VPN sites.

### B. PE configuration

Various configurations are needed in the PEs inside the Autonomous Systems (AS) to implement the label-hopping scheme. These are listed below:

1) A set of "$m$" algorithms that generate collision-free labels (universal hashing algorithms) are implemented in the PEs. Each algorithm is mapped to an index $A = (a_1, a_2, \cdots a_m), m \geq 1$. Ordering of the algorithms must be the same in the PEs. If the PEs used are from different vendors then a standardised set of algorithms must be used.
2) The bit-selection pattern is used by the PE. This helps in determining the bits chosen for generating the additional label. This additional label plays a role in avoiding collision in the hash values.
3) $PE_{ne}$ is configured for a FEC or a set of FECs represented by an aggregate label (per VRF label). For each FEC or a set of FECs, a set of valid labels used for hopping, $K = (k_1, k_2, k_3, \cdots k_n), n > 1$ and, $k_i \neq k_j$ if $i \neq j$, is configured in $PE_{ne}$. This helps in selective application of the schemes for the FECs. In the case of bi-directional security, the roles of the PEs are reversed.

### C. Control and data-plane flow

Initially, set $K$ and the bit-selection pattern used by the PEs are exchanged securely over the control-plane. Optionally an index from $A$, representing a hash-algorithm, could also be exchanged. We propose that only the index is exchanged between the PEs, as it enhances the security for two reasons. First, the algorithm itself is masked from the attacker. Second, the algorithm can be changed frequently, and it would be difficult for the attacker to identify the final mapping that generates the label to be used for a packet. Figure 1 depicts this unidirectional exchange from $PE_{ne}$ to $PE_{fa}$.

Once the secure control-plane exchanges are completed, we apply the label-hopping technique. $PE_{fa}$ forwards the labelled traffic towards $PE_{ne}$ through the intermediate routers using the label-stacking technique (Figure 2). The stacked labels along with the payload are transferred between the AS and ASBRs before they reach $PE_{ne}$. Using the label-hopping algorithm $PE_{ne}$ verifies the integrity of labels. Upon validation, $PE_{ne}$ uses the label information to forward the packets to the appropriate VPN service instance or site. This data-plane exchange from $PE_{fa}$ and $PE_{ne}$ is depicted in Figure 3. A legend for Figure 3 is given in Table I. Figure 3 also shows how the labels for the packets are specified when the data packets flow from CE2 to CE1. In the figure, the L3

header network address is $172.16.10.1$ whose gateway is CE1. We now present the label-hopping scheme.

| Abbreviation | Description |
|---|---|
| AS | Autonomous Systems |
| ASBR | Autonomous System Border Router |
| CE | Customer Edge Routers |
| LDP: L1-L4 | Label Distribution Protocol with link labels |
| NH | Next Hop |
| PE | Provider Edge Routers |
| POP, V1 | Label between AS1 and $PE_{ne}$ |
| VPN | Virtual Private Network |

TABLE I: Legend for Figures 1 and 3

## III. LABEL-HOPPING TECHNIQUE

Once a data packet destined to the $PE_{ne}$ arrives at the $PE_{fa}$ a selected number of bytes from the payload is chosen as input to the hashing algorithm. The resulting hash-digest is used to obtain the first label for the packet. The agreed bit-selection pattern is then applied on the hash-digest to determine an additional label, which is then concatenated with the first label. Once $PE_{ne}$ receives these packets it verifies both the labels.

The implementation steps for the control-plane at the $PE_{ne}$ and $PE_{fa}$ are given by Algorithm 1 and Algorithm 2. The implementation steps for the data-plane at the $PE_{fa}$ and $PE_{ne}$ are given by Algorithm 3 and Algorithm 4.

A brief explanation of these algorithm follows:

Algorithm 1 exchanges four attributes, namely

1) the acceptable Forward Equivalence Classes (FECs),
2) valid and acceptable labels for each of the FECs,
3) the pointer or instance to the hash algorithm, and
4) the bit selection pattern to be used, with the $PE_{fa}$ using a secure control-plane exchange.

Step 3 of Algorithm 1 assumes that the function *CP-SendPacket()* sends secure encrypted data packet to $PE_{fa}$.

**Algorithm 1** Control-plane $PE_{ne}$ algorithm

**Require:** FEC[] Forward Equivalence Classes, K[] valid labels, A[i] hash algorithm instance, I[] the bit-selection pattern chosen for the inner label.

```
1: Begin
2: packet = makepacket(FEC,K, A[i], I);
3: CP-SendPacket(PE_fa, MP-eBGP, packet);
4: End
```

Algorithm 2 receives the secure packet, decrypts it and then fills up its tables by extracting the FECs and the label mapping of the FECs. It then selects the hash algorithm based on the instance or the pointer passed by the $PE_{ne}$. These are done in steps $3-7$. We assume that both the PEs implement the same hash algorithms corresponding to the pointers or instances that are passed. Note that this is pre-configured in the routers. The $PE_{fa}$ also gets to know the valid bit selection pattern that is acceptable for the $PE_{ne}$ in step $8$.

**Algorithm 2** Control-plane $PE_{fa}$ algorithm

**Require:** None

```
1: Begin
2: packet = CP-ReceivePacket(PE_ne); // from PE_ne
3: FEC[] = ExtractFEC(packet); // extract FECs
4: K[] = ExtractLabels(packet); // extract the labels
5: selectHashAlgorithm(A[i]); // hash algorithm to use
6: RecordValues(FEC); // information for PE_fa
7: RecordValues(K); // information on the keys
8: RecordValues(I); // bit-selection pattern to be used
9: End
```

Algorithm 3 describes the processing that occurs before the data packets are sent from $PE_{fa}$. Steps $3-6$ in the algorithm checks whether the label-hopping algorithm is enabled for the FEC. If it is not enabled, the algorithm will proceed to exchange data packets without label-hopping. If the label-hopping algorithm is enabled for the FEC, then the hash-digest of the packet, as well as the first and additional labels are generated at steps $7-9$. The data packet is then encapsulated with the labels and sent to the $PE_{ne}$.

**Algorithm 3** Data-plane $PE_{fa}$ algorithm

**Require:** None

```
1: Begin
2: packet = DP-ReceivePacket(Interface);
3: match = CheckFEC(packet); // Is the algorithm enabled?
4: if match == 0 then
5:    return; // algorithm not enabled.
6: end if
7: hash-digest = calculateHash(A[i],packet);
8: first-label = hash-digest % |K|;
9: addl-label = process(hash-digest,I)
10: DP-SendPacket(PE_ne, first-label, addl-label, packet);
11: End
```

Algorithm 4 receives the encapsulated packet from $PE_{ne}$. It then determines whether the FECs deploy the label-hopping scheme; see steps $3-6$. In steps $7-11$, the algorithm extracts the labels from the packet and calculates the hash-digest for the packet as well as the inner and additional labels. It compares the calculated values with the extracted values of the labels; see steps $12-17$. If a match exists on the labels sent by $PE_{fa}$, then the packet is considered to be valid. The data packets are passed to the CE after removing the labels that match.

Figure 4 gives a modified version of a sequence diagram for all the four algorithms discussed in this section. This diagram also partially shows the calls executed by the PEs in the control and data-planes.

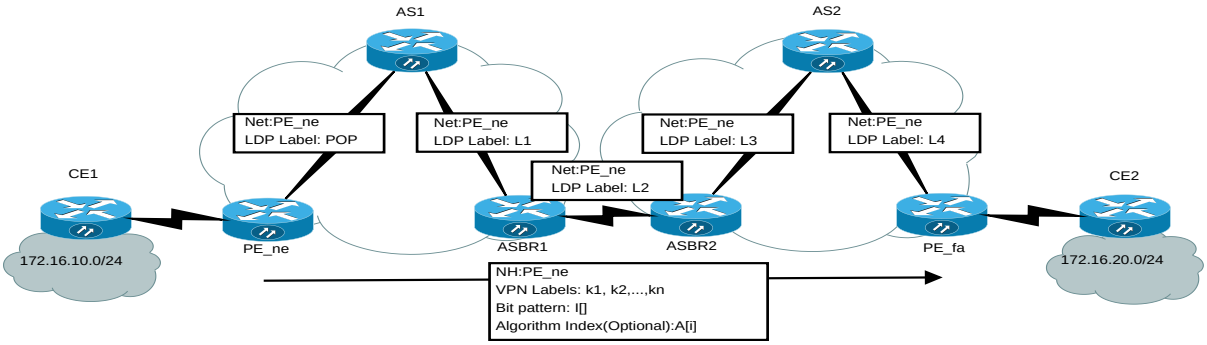


Fig. 1: Control-plane exchanges for model "C" [10]

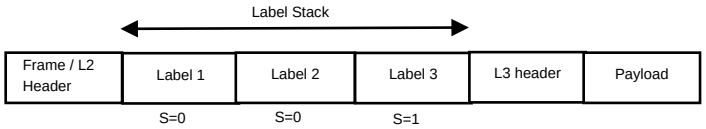


Fig. 2: Label stack using scheme outlined for model "C"

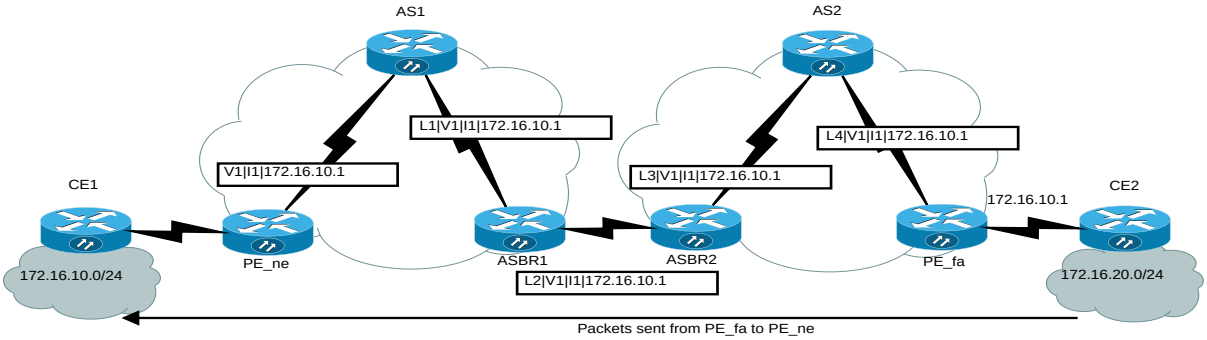


Fig. 3: Data-plane flow for model "C" [10]

**Algorithm 4** Data-plane $PE_{ne}$ algorithm

**Require:** None

```
1: Begin
2: packet = DP-ReceivePacket(Interface);
3: match = CheckFEC(packet);
4: if match == 0 then
5:    return; //no match
6: end if
7: label-in-packet=extractPacket(packet, LABEL);
8: inner-label=extractPacket(packet, INNER-LABEL);
9: hash-digest=calculateHash(A[i],packet);
10: first-label=hash-digest % |K|;
11: additional-label = process(hash-digest,I)
12: if label-in-packet ≠ first-label then
13:    error(); return;
14: end if
15: if inner-label ≠ additional-label then
16:    error(); return;
17: end if
18: DP-SendPacket(CE1, NULL, NULL, packet);
19: End
```

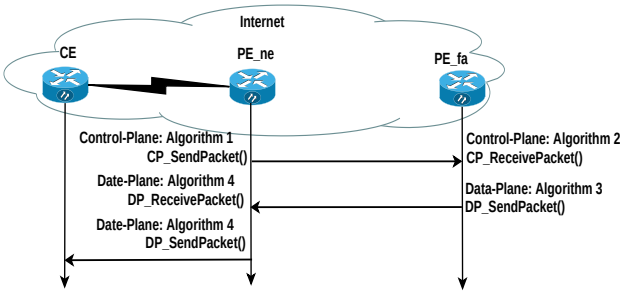


Fig. 4: A modified sequence diagram showing the applicability of the algorithms in the control and the data-plane.

The values in $K$ need not be contiguous and can be randomly chosen from a pool of labels to remove coherence in the label space. Also the algorithms used could be either vendor dependent or a set of standard algorithms mapped the same way by the $PE_{ne}$ and $PE_{fa}$. If the two PEs involved are from different vendors we assume that a set of standard algorithms are used. In order to avoid too many processing cycles in the line cards of $PE_{ne}$ and $PE_{fa}$, the hash-digest is

calculated over a predefined size of the payload. The additional inner label is added to enhance protection against spoofing attacks. With an increased label size, an attacker spends more time to guess the VPN instance label for the site behind $PE_{ne}$. There could be two hash-digests that generate the same label. In this case, the two hash-digests are distinguished using the additional label. Collisions can be avoided by re-hashing or any other suitable techniques that are proposed in the literature [11]. If collisions exceed a certain number, then Algorithms 1 and 2 can be executed with a set of new labels.

### A. Illustration

We now illustrate the label-hopping scheme. In Figure 1, using Algorithms 1 and 2, a set of labels are forwarded from $PE_{ne}$ to $PE_{fa}$. The roles of $PE_{ne}$ and $PE_{fa}$ are interchanged for reverse traffic. Figure 2 shows a packet from the data-plane for model "C" with the proposed scheme. In Figure 2, "Label 1" refers to the outermost label, while "Label 2" refers to the label generated from the hash-digest and "Label 3" refers to the additional label that is generated as shown in Algorithm 3. This additional label, denoted by S, has a bottom of stack bit set; see Figure 2. These labels are stacked immediately onto the packet and the path labels for routing the packets to appropriate intermediary PEs are added. Figure 3 also shows these path labels used by the data packet to reach $PE_{ne}$.

Note that the labels that are exchanged need not be related with the services offered by the VPNs. A separate mapping can be maintained internally by the PEs. When the packet passes through the core of an intermediary AS involved in model "C", or through the network connecting the intermediary AS, the intruder or the attacker has the capability to inspect the labels and the payload. However, the proposed scheme prevents the attacker from guessing the right combination of the labels as the labels change with every data packet.

### B. Computational time complexity

The computational time complexity of the algorithms executing at the control-plane is $O(1)$. The data-plane algorithms have a computational time complexity of $O(HashPacketSize)$. The packet size chosen for hashing could either be $64$ or $128$ bytes. Further control-plane exchanges are less frequent than the data-plane exchanges. In terms of processing, hashing small data sizes may not be an issue but frequently hashing every data packet increases the processing time. Hence, it would be of interest to reduce the computational time complexity of the data-plane to $O(1)$.

The most time consuming step in the data-plane algorithms is the hashing of data packets. We show how this hashing step can be removed by using the Timing over IP connection and Transfer of Clock (TicToc) [12] based Precision Time Protocol Label Switch Path (PTP-LSP) [13]. We discuss some important aspects of this algorithm.

## IV. TicToc based label-hopping

If we use the TicToc based PTP LSP then a pre-calculated set of distinct values $d_{ijk}$ for a specific time slot $i$, FEC $j$ and a label index $k$ could be exchanged over the control-plane periodically. These discrete values can then replace the hash values calculated in Algorithms 3 and 4 thereby improving speed-up. In this case, a few of the values from $d_{(i-1)jk}$ and $d_{(i+1)jk}$ must overlap with $d_{ijk}$ forming a sliding window of distinct values. The sliding window is necessary to account for any latency in the clock information. In case $|d_{ijk}|$ is large then we can transfer a random seed for generating pseudo-random numbers $R_{ij}$ which generates $k$ values for every time instant $i$ and FEC $j$. The algorithm for generating the pseudo-random numbers must, a-priori, be known to $PE_{fa}$ and $PE_{ne}$. The sliding window of labels with the distinct values for three consecutive time slots is given in Figure 5.

The ports of the PEs must be configured to enable the functioning of the TicToc protocol. The rest of the configuration of the PEs is similar to the label-hopping schemed discussed in Section II-B.

As before, for each FEC or a set of FECs, a set of valid labels used for hopping in the initial time slot $i$ is exchanged. These labels $D = (d_1; d_2; d_3; \therefore d_n)$ where $n \geq 1$ and, $k_i \neq k_j$ if $i \neq j$ are then configured in $PE_{ne}$. For the set of labels $D$ time slices $TS = (TS_1; TS_2; TS_3 \therefore TS_n)$ are also exchanged. These time slices can be periodically changed and a new set of $TS$ ranging from $TS_1$ to $TS_n$ can be exchanged after a time duration of $TS\_Exchange\_Interval$ from time to time.

The complete sets of algorithms are given in the Appendix. The algorithm given for the control and the data-plane have a constant computational time complexity $O(1)$ while achieving the same objective of mitigating spoofing attacks. The main reason behind using TicToc is to synchronise the labels based on time. We could even consider the use of currently existing Network Time Protocol (NTP) [14] instead of TicToc to synchronise the labels. NTP is widely used to synchronise a device to Internet time servers or other sources. However, such a discussion is beyond the scope of this paper.

It should be noted that the algorithms protect against spoofing attacks. Replay attacks are still possible on systems implementing these schemes. In the next section, we show that Algorithms 1, 2, 3, and 4 can also be modified to mitigate replay attacks.

## V. Mitigating replay attacks

In replay attacks, a valid data packet is replayed or delayed. Since the previous algorithm uses three consecutive time slots, an attacker can replay the packets within three time slots. In the hashing based algorithms the packet can be replayed many times until the labels are valid. Algorithms proposed in the previous sections cannot detect such attacks. Therefore to mitigate replay attacks we introduce a random seed. This random seed, henceforth referred as *Rseed*, generates pseudo random numbers which are used as the label for the time slots. We now discuss the modified algorithms in detail.

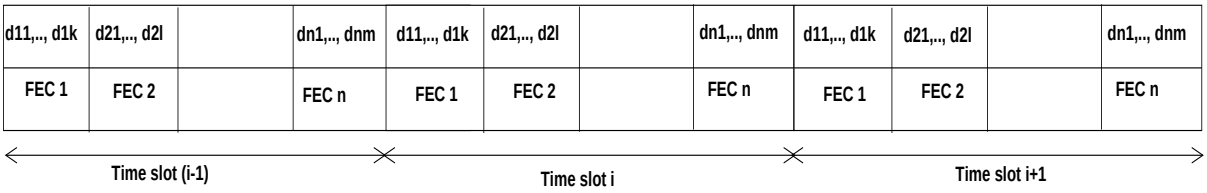


Fig. 5: Total distinct valid keys for a particular time slot $i$, various FECs $j$ in a TicToc based protection against spoofing attacks. In case the keys are too large then a random seed which can generate the keys $d_{ijk}$ can be exchanged. In this case, $PE_{fa}$ and $PE_{ne}$ must know the random number generation algorithm a-priori.

### A. PE configuration

PEs that implement this scheme need extra configuration details in addition to those discussed in Section II-B. This includes the algorithm for pseudo random number generator, the random seed to be exchanged as well as the configuration of ports for implementing the TicToc protocol.

### B. Control and Data-plane flow

As given in the TicToc based algorithms in the Appendix, the control-plane exchanges involve constructing a PTP LSP for deriving the clock at the $PE_{ne}$ and $PE_{fa}$ for the forwarding direction. Each pair of $PE_{ne}$ and $PE_{fa}$ knows the PTP port and corresponding PTP LSP used for the traffic. The PTP LSP is intended for providing the clocking between a pair of $PE_{ne}$ and $PE_{fa}$. The clock or time-stamp derived from this PTP LSP is used in the data-plane to determine the valid label at that time instant. Upon validation, $PE_{ne}$ uses the label information to forward the packets to the appropriate VPN service instance or site.

Once a data packet destined to the $PE_{ne}$ arrives at the $PE_{fa}$ the first-label is chosen using K, TS, and *Rseed*. A selected number of bytes from the payload is chosen as input to the hashing algorithm. The agreed bit-selection pattern is then applied on the hash-digest to obtain an additional label, which is then concatenated with the first label. Once $PE_{ne}$ receives these packets it verifies both the labels. Note that in case hashing is not preferred we could use the predetermined set of labels as discussed in the Appendix. The details of the algorithm to mitigate spoofing as well as replay attacks are described below.

The PTP port number and the related PTP LSP information are assumed to be configured before any information exchange in the data-plane. Algorithm 5 forwards the FECs, their associated keys, a set of valid time slices, a random seed and the bit selection pattern for the inner labels from $PE_{ne}$ to $PE_{fa}$. The packets are exchanged using Multi protocol External BGP (step 3).

Algorithm 6 runs at the $PE_{fa}$. It receives the packet and extracts information related with FECs, labels, time slices, random seed (steps $2-6$) and records them (steps $8-12$). It selects the hash algorithm based on the instance and uses the random seed value to generate pseudo-random numbers. It is assumed that both $PE_{ne}$ and $PE_{fa}$ will use the same pseudo-random number generator.

**Algorithm 5** Control-plane $PE_{ne}$ algorithm

**Require:** FEC[] Forward Equivalence Classes, K[] valid labels, TS[] valid time slices, A[i] hash algorithm instance, I[] the bit-selection pattern chosen for the inner label, Random seed *Rseed* which is used for generating the index into set K (set of labels), PTP port and PTP LSP information.

1: Begin
2: packet = makepacket(FEC,K, TS, A[i], I, Rseed);
3: CP-SendPacket($PE_{fa}$, MP-eBGP, packet);
4: End

**Algorithm 6** Control-plane $PE_{fa}$ algorithm

**Require:** None

1: Begin
2: packet = CP-ReceivePacket($PE_{ne}$); // from $PE_{ne}$
3: FEC[] = ExtractFEC(packet); // extract FECs
4: K[] = ExtractLabels(packet); // extract the labels
5: TS[] = ExtractTimeSlices(packet); // extract the time slices
6: Rseed = ExtractRandomSeed(packet); // extract the *Rseed* value.
7: selectHashAlgorithm(A[i]); // hash algorithm to use
8: RecordValues(FEC); // information for $PE_{fa}$
9: RecordValues(K);
10: RecordValues(TS);
11: RecordValues(I); // bit-selection pattern to be used
12: RecordValue(Rseed);
13: End

Algorithm 7 is implemented by $PE_{fa}$. Steps $2-6$ identify the current time slot. The keys for this time slot have already been exchanged in the control-plane. The algorithm works only if the label-hopping is enabled on the FECs. If the label hopping is enabled steps $13-26$ are executed. In step 13, the hash value of the packet is calculated. Steps $14-23$ manages the time slots. We assume that there are $n$ time slots. If all the time slots are completed, we wrap around to time slot 0. A random number is generated and a key for the particular time slot is selected in step 24. The additional labels are created based on the previous identified bit pattern. The packet is then forwarded to the $PE_{ne}$ (see steps $25-26$).

**Algorithm 7** Data-plane $PE_{fa}$ algorithm

**Require:** None

```
1: Begin // One time initialisation
2: CurrentTimeSliceIndex = 0;
3: CurrentMasterClock = PTP LSP Master Clock Timestamp;
4: CurrentTimeInstant = CurrentMasterClock;
5: NextTimeInstant = CurrentMasterClock
                     + TS[CurrentTimeSliceIndex];
6: End
7: Begin // repeated for every data packet
8: packet = DP-ReceivePacket(Interface);
9: match = CheckFEC(packet); // Is the algorithm enabled?
10: if match == 0 then
11:    return; // algorithm not enabled.
12: end if
13: hash-digest = calculateHash(A[i],packet);
14: if CurrentTimeInstant ≤ NextTimeInstant ((+ or -) configured seconds) then
15:    // do nothing;
16: else
17:    CurrentTimeSliceIndex++;
18:    if CurrentTimeSliceIndex == n then
19:       CurrentTimeSliceIndex = 0; // check to wrap around
20:    end if
21:    CurrentTimeInstant = NextTimeInstant;
22:    NextTimeInstant = CurrentTimeInstant
                         + TS[CurrentTimeSliceIndex];
23: end if
24: first-label = K[GenerateRandom(Rseed) % |K|];
25: addl-label = process(hash-digest,I)
26: DP-SendPacket(PE_ne, first-label, addl-label, packet);
27: End
```

Algorithm 8 has to take care of lead or lag in the clock. Since there could be a time-lag between sending and receiving packets, $PE_{ne}$ has to maintain three random seeds. These include the random seed for the previous time slot and the current time slot. In case the time-slots have already wrapped once, the future random seed of the time slot is also stored. Steps $15-33$ takes care of this activity. The else part in steps $17-23$ stores the previous, the current and the next random seed (if it exists). The hashing should be applied on the packets and then the correct label must be chosen based on the random seed values. Steps $2-5$ does a one-time initialisation for the time slot. Functionality of steps $12-14$ and $35-42$ have been discussed in Algorithm 4.

The change in the algorithm to randomly pick up a label for the next time slot will help in avoiding man-in-the-middle attackers from synchronising with the time slots. The labels in the previous algorithms are predictable if a large number of packets were observed. The *Rseed* will generate values in lock step with the time slots at both the $PE_{fa}$ and $PE_{na}$. This will

**Algorithm 8** Data-plane $PE_{ne}$ algorithm

**Require:** None

```
1: Begin // One time initialisation
2: CurrentTimeSliceIndex = 0;
3: CurrentMasterClock = PTP LSP Clock Timestamp;
4: CurrentTimeInstant = CurrentMasterClock;
5: NextTimeInstant = CurrentMasterClock
                     + TS[CurrentTimeSliceIndex];
6: Begin // For each packet
7: packet = DP-ReceivePacket(Interface);
8: match = CheckFEC(packet);
9: if match == 0 then
10:    return; //no match
11: end if
12: label-in-packet=extractPacket(packet, LABEL);
13: inner-label=extractPacket(packet, INNER-LABEL);
14: hash-digest=calculateHash(A[i],packet);
15: if CurrentTimeInstant ≤ NextTimeInstant ((+ or -) configured seconds) then
16:    // do nothing;
17: else
18:    CurrentTimeSliceIndex++;
19:    OldRseedIndex = RseedIndex;
20:    RseedIndex = (GenerateRandom(Rseed) % |K|);
21:    NextRseedIndex =
       LookAheadRseedIndex(GenerateRandom(Rseed)%|K|);
22:    RollbackRseed(Rseed by 1);
23:    if CurrentTimeSliceIndex == n then
24:       // check to wrap around
25:       CurrentTimeSliceIndex = 0;
26:    end if
27:    CurrentTimeInstant = NextTimeInstant;
28:    NextTimeInstant = CurrentTimeInstant
                         + TS[CurrentTimeSliceIndex];
29: end if
30: // Check if label used before in the previous, current
31: // or future time slot can be used
32: // Check with OldRseedIndex, RseedIndex
    // and NextRseedIndex
33: first-label = K[RseedIndex (+ or −1)];
34: additional-label = process(hash-digest,I)
35: if label-in-packet ≠ first-label then
36:    error(); return;
37: end if
38: if inner-label ≠ additional-label then
39:    error(); return;
40: end if
41: DP-SendPacket(CE1, NULL, NULL, packet);
42: End
```

prevent an attacker from synchronising with label changes and hence replay attacks could be avoided. The sequence diagram given in Figure 4 is valid for Algorithms 5-8.

Note that the changes to the label-hopping algorithms presented in this section can be applied to the algorithms given in the Appendix. This will ensure that the spoofing and replay attacks are mitigated close to wire speed. We do not discuss the details in this paper.

## VI. Simulation

In this section, we present the simulation results on performance, comparing the various label-hopping technique including deep packet inspection where we encrypt and decrypt the complete packet.

Implementing the label-hopping algorithm for sets of FECs belonging to any or all VPN service instances may cause throughput degradation. This is because the hash-digest computation and derivation of the inner-label / additional inner label calculation can be intensive operations. We therefore compared our technique by choosing a part of the payload as input to our hashing algorithm.

We simulated our algorithm on a $2.5$ GHz Intel dual processor quad core machine. We compared the performance of the label-hopping technique with a deep packet inspection technique where the complete packet was encrypted before transmission and decrypted on reception. The performance of the data-plane level algorithm on $PE_{ne}$ is shown in Figure 6. Simulations without the use of TicToc schemes indicate that we were able to process $10$ million packets per second when we used 64-byte for hashing on a payload of size $1024$ bytes. For a hash using $128$-byte, we were able to process approximately $6.3$ million packets per second. However, with complete encryption and decryption of the packet, we were able to process only about $1$ million packets per second.

The TicToc based algorithms given in the Appendix adds only constant time to the computation. There is an increase between $1-3\%$ in computation time depending on successful identification of label from the correct time slot. Therefore the results in Figure 6 show that the lines lie very closely to wire-speed performance.

When we combine label-hopping, TicToc and pseudo random number generation it adds approximately $2-5\%$ of additional computation time to the label-hopping algorithm. Since the difference in processing speeds is less than $5\%$, the performance with label-hopping lies very close to $64$ and $128$ bit hash lines in the figure. We were able to process approximately $9.6$ million packets for a $64$-bit hash and $6$ million packets for $128$-byte hash.

Based on the simulation results we suggest two solutions that can be implemented:

- The payload based label-hopping can be applied to specific VPN traffic between the PEs which are mission-critical and sensitive.
- The TicToc based label-hopping algorithm can be applied on those VPNs that have high link reliability and require line-rate operation.

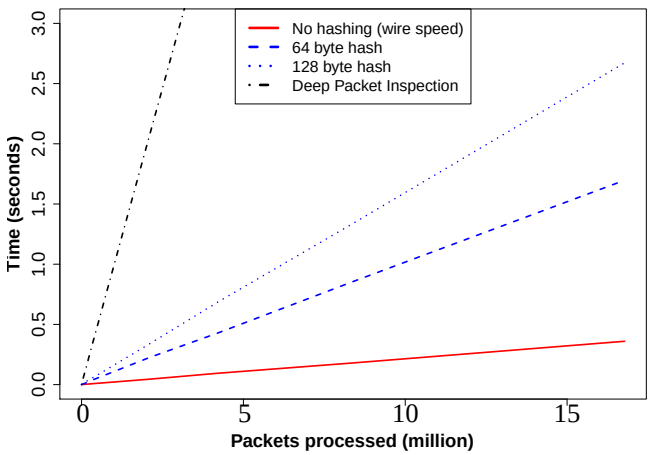


Fig. 6: Performance comparison of complete packet encryption and decryption with a $64$, $128$ byte hash on a payload of size $1024$ bytes. The performance of the TicToc based label-hopping is very close to wire speed. Replay attack prevention adds approximately $2-5\%$ extra time to the algorithms.

In the next section, we consider some implementation issues that must be addressed while using the label-hopping methods.

## VII. Implementation

In the PEs label-hopping can be enabled or disabled by using a look-up table. This look-up table can be used to efficiently implement the algorithms which can be programmed with an on or off bit to indicate whether the label-hopping scheme is deployed. In case the scheme is deployed, then the $PE$s compute inner labels 2 and 3 using Algorithm 8. If the packet is valid it is accepted, else it is discarded.

One concern of the scheme is to have a method to tackle the problem of fragmentation that can occur along the path from $PE_{fa}$ to $PE_{ne}$. We can fragment the packet at $PE_{fa}$ and ensure that the size of the packet is fixed before transmission. The other method is to employ the Path maximum transfer unit (Path-MTU) discovery process so that packets do not get split further into multiple fragments. If packets are fragmented this scheme fails. However, networks employ the Path-MTU discovery process to prevent fragmentation and hence this problem may not exist.

The proposed label-hopping method based on payload expects the same hashing algorithms to be used at both the $PE$s. If the vendors of the $PE_{ne}$ and $PE_{fa}$ are different then interoperability issues must be addressed. In our scheme, we chose the time instant that the packet leaves the $PE_{fa}$ and this time instant serves as the variable component that the attacker cannot decipher. This requires the use of time synchronisation mechanism. This is provided by the PTP LSP.

## VIII. Impact on applications

We briefly discuss the impact of the label-hopping method on a Deep Packet Inspection Engine (DPIE) and ISP billing. We show that by using label-hopping the workload on the DPIE can be reduced but the processing load on the ISP billing mechanism would increase.

### A. Enhancing DPIE performance

Once a spoofed packet is detected at the $PE_{ne}$ an error routine is called (see Algorithm 8). Such packets can be collected and periodically sent to a DPIE. Without the application of this algorithm, all the packets that are received by the customer site have to be sent to the DPIEs for possible attack detection. The proposed technique helps in filtering packets which would otherwise undergo deep packet inspection.

To mitigate attacks ranging from buffer overflow hack attempts to denial of service attacks such as tear drop attacks, the DPIEs record the packets in a secondary storage for inspection and correlation. The correlation engines also need computation power and memory, for deciphering and raising alarms to the about possible attack on specific VPN or a set of VPN sites. With reduced number of packets sent to DPIEs, the attack detection and correlation can be applied only to these filtered set of packets. It is important to note that if this label-hopping scheme was not adopted, some sort of DPIEs would have to be placed within the customer's network.

Another less preferred method is to have the DPIEs on the PE itself. With label-hopping scheme in place there is no need for having DPIEs on the PEs. The error packets can be spanned or replicated and sent to a suitable cluster of DPIE engines at the customer site for further correlation. An alternate solution for the ISP deploying the PEs could be to provide the first level of warning while the customer's hardware could do the rest of the mitigation and protection. This would be a co-operative solution between the customer and the ISP that reduces the time taken in the event of an attack. The label-hopping scheme would bring an extra level of protection.

### B. ISP Billing

A concern of such a scheme is the billing related aspects for ISPs as the labels change periodically. Most of the ISPs use the labels to bill their customer. The modification to billing can be done as follows: the traffic statistics are collected for all the VPN labels as if they were separate labels. At the egress PEs, statistics are gathered for the data packet and labels coming towards it based on a set of labels that would be exchanged. This data can then be used along with the ASBR statistics (identifying the egress PE by the outer label) for billing purposes. The label-hopping scheme involves more processing for billing by the ISPs.

## IX. Outlook

Today, there is reluctance among service providers to use MPLS VPN model "C" due to security concerns like spoofing and replay attacks. We propose methods to secure the Inter-Provider MPLS VPN model "C" data-plane by preventing spoofing, replay and other unidirectional attacks.

### A. Contributions

In this paper, we proposed a label-hopping scheme for inter-provider BGP-based MPLS VPNs that employ MP e-BGP multi-hop control-plane exchanges. In such an environment without label-hopping, the data-plane is subject to spoofing and replay attacks. Spoofing attacks can be prevented by using the payload-based label-hopping scheme. A combination of label-hopping, TicToc and the use of pseudo-random numbers serve to mitigate replay attacks. The proposed schemes prevent the spoofed or the replayed packets from getting into a VPN site. Simulations show that the use of randomised labels for label-hopping along with TicToc can operate at line-rate. All the proposed schemes are computationally less intensive as compared to other encryption-based methods. One additional advantage, of the label-hopping scheme, is that the workload for deep packet inspection engines can be reduced. However, there would be an increase in computation time complexity for ISP billing. This trade off could be worth considering. We hope that the methods proposed will encourage ISPs to experiment with MPLS VPN model "C".

### B. Avenues for future work

There are some cases where the label-hopping scheme cannot be used. For example, consider Equal Cost Multicast Path (ECMP) scenarios. In this case, a flow arriving at a router could choose any of the available equal cost paths to reach the destination. However, it is advisable that the flows of the same service choose a unique path out of the available equal cost paths. Otherwise, reordering of packets could occur at the receiving end as two equal cost paths may not have the same latency. For any real-time flows, reordering of packets introduces processing concerns at the receiver. The current practice to avoid reordering is to hash the flow labels so that flows of the same service are redirected through a specified unique path.

If flow hashing is done on Label Switching Routers (for pseudowires in RFC 6391), then the labels generated by the label-hopping technique might hash to different paths. Therefore reordering schemes have to be introduced at the receiver end. It is desirable to propose methods to solve this problem in the future.

## Acknowledgements

The authors would like to acknowledge the UK EP-SRC Digital Economy Programme and the Government of India Department of Science and Technology (DST) for funding given to the IU-ATC. The authors would like to thank Josh Rogers, James Uttaro, Robert Raszuk, Tal Mizrahi, Robert Raszuk, Greg Mirsky, Jakob Heitz and Bhargav Bhikkaji for the extensive and useful email discussions.

## References

[1] S. Raman and G. Raina, *Mitigating Spoofing Attacks in MPLS-VPNs using Label-hopping*, Proceedings of the Eleventh International Conference on Networks (ICN 2012), pp. 241–245, ISBN: 978-1-61208-183-0.

[2] Y. Rekhter, B. Davie, E. Rosen, G. Swallow, D. Farinacci, and D. Katz, *Tag switching architecture overview*, Proceedings of the IEEE, vol. 85, no. 12, December 1997, pp. 1973–1983, doi:10.1109/5.650179.

[3] Advance MPLS VPN Security Tutorials [Online], Available: *http://etutorials.org/Networking/MPLS+VPN+security/Part+II+Advanced+MPLS+VPN+Security+Issues/*, [Accessed: 20th December 2012]

[4] M. H. Behringer and M. J. Morrow, *MPLS VPN security*, Cisco Press, June 2005, ISBN-10: 1587051834.

[5] S. Alouneh, A. En-Nouaary, and A. Agarwal, *MPLS security: an approach for unicast and multicast environments*, Annals of Telecommunications, Springer, vol. 64, no. 5, June 2009, pp. 391–400, doi:10.1007/s12243-009-0089-y.

[6] C. Semeria, "RFC 2547bis: BGP/MPLS VPN fundamentals", Juniper Networks white paper, March 2001.

[7] L. Fang, N. Bita, J. L. Le Roux, and J. Miles, *Interprovider IP-MPLS services: requirements, implementations, and challenges*, IEEE Communications Magazine, vol. 43, no. 6, June 2005, pp. 119–128, doi: 10.1109/MCOM.2005.1452840.

[8] C. Lin and W. Guowei, *Security research of VPN technology based on MPLS*, Proceedings of the Third International Symposium on Computer Science and Computational Technology (ISCSCT 10), August 2010, pp. 168–170, ISBN-13:9789525726107.

[9] B. Daugherty and C. Metz, *Multiprotocol Label Switching and IP, Part 1, MPLS VPNS over IP Tunnels*, IEEE Internet Computing, May–June 2005, pp. 68–72, doi: 10.1109/MIC.2005.61.

[10] Inter-provider MPLS VPN models [Online], Available: *http://mpls-configuration-on-cisco-ios-software.org.ua/1587051990/ch07lev1sec4.html*, [Accessed 20th December 2012].

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd edition, MIT Press, September 2009, ISBN-10:0262033844.

[12] Timing over IP and Transfer of Clock Work Group [Online], Available: *http://datatracker.ietf.org/wg/tictoc/*, [Accessed: 20th December 2012].

[13] Precision Time Protocol LSP [Online], Available: *http://www.nist.gov/el/isd/ieee/ieee1588.cfm*, [Accessed: 20th December 2012].

[14] D. L. Mills, *Computer Network Time Synchronization - the Network Time Protocol on Earth and in Space*, CRC Press, 2010. ISBN:1439814635,

## APPENDIX

In this appendix, we show a constant time $O(1)$ algorithm for mitigating spoofing attacks in MPLS VPNs using time synchronized label-hopping. The time synchronization is achieved using the TicToc protocol. We assume that each $(PE_{ne}, PE_{fa})$ pair knows the PTP port and the corresponding PTP LSP used for sending and receiving time information. We now present the four algorithms which removes packet based hashing and uses time-based labels.

Algorithm 9 forwards the FECs, keys associated with the FECs and a set of valid time slices from $PE_{ne}$ to $PE_{fa}$. The packets are exchanged using Multiprotocol External BGP in step 3. For every time slice instant this process is repeated. This is shown is steps $5-10$. The algorithm then wraps around and restarts from TS[0].

**Algorithm 9** TicToc: Control-plane $PE_{ne}$ algorithm

**Require:** FEC[] Forward Equivalence Classes, D[] valid labels, TS[] valid time slices, PTP port and PTP LSP information.

```
1: Begin
2: packet = makepacket(FEC, D, TS);
3: CP-SendPacket(PE_fa, MP-eBGP, packet);
4: Sleep(TS[1]);
5: For every time instant TS[i] starting from TS[2]...TS[n],
6: Begin
7: packet = makepacket(FEC, D);
8: CP-SendPacket(PE_fa, MP-eBGP, packet);
9: Sleep(TS[i]);
10: End
11: End
```

Algorithm 9 shows that $PE_{ne}$ transfers the distinct values at every time instant TS[i]. $PE_{ne}$ has to store the values for three consecutive time intervals to ensure that the latency involved in sending new labels to $PE_{fa}$ is also accounted.

The values sent by the $PE_{ne}$ are extracted from the packet in steps $3-5$. For example, the procedure ExtractLabelsAndAppend(packet) given in Algorithm 10 helps the $PE_{fa}$ use the new labels received at every new time instant from $PE_{ne}$. All the values are recorded in steps $6-9$, for executing the verification activity when a packet arrives.

$PE_{fa}$ initiates the synchronization activity in steps $1-6$ in Algorithm 10. Once a packet is received then the algorithm checks whether the FEC of the packet has label-hopping enabled in steps $9-12$. If enabled, the time instances are checked and a label is chosen for the current time index, added to the packet and sent; see steps $14-27$.

Extra work needs to be done at the level of the data-plane for managing the synchronization so that packets are not rejected. Hence, $PE_{ne}$ can store values of $D$ for three consecutive time slots. $PE_{ne}$ synchronizes itself with the time slots of $PE_{fa}$. This is shown in steps $1-6$ and $14-25$ in Algorithm 12. If the label-hopping algorithm for the packet is enabled, then the

**Algorithm 10** TicToc: Control-plane $PE_{fa}$ algorithm

**Require:** None

```
1: Begin
2: packet = CP-ReceivePacket(PE_ne); // from PE_ne
3: FEC[] = ExtractFEC(packet); // extract FECs
4: D[] = ExtractLabelsAndAppend(packet); // labels
5: TS[] = ExtractTimeSlices(packet); // extract time slices
6: RecordValues(FEC); // information for PE_fa
7: RecordValues(D);
8: RecordValues(TS);
9: End
```

**Algorithm 11** TicToc: Data-plane $PE_{fa}$ algorithm

**Require:** None

```
1: Begin // One time initialization start
2: CurrentTimeSliceIndex = 0;
3: CurrentMasterClock = PTP LSP Master Clock Timestamp;
4: CurrentTimeInstant = CurrentMasterClock;
5: NextTimeInstant = CurrentMasterClock
                     + TS[CurrentTimeSliceIndex];
6: End // One time initialization end
7: Repeat forever
8: Begin
9: packet = DP-ReceivePacket(Interface);
10: match = CheckFEC(packet); // Is the algorithm enabled?
11: if match == 0 then
12:    return; // algorithm not enabled
13: end if
14: if CurrentTimeInstant ≤ NextTimeInstant (configured seconds) then
15:    // do nothing;
16: else
17:    // Move by next TS[i]
18:    CurrentTimeSliceIndex++;
19:    if CurrentTimeSliceIndex == n then
20:       // check to wrap around
21:       CurrentTimeSliceIndex = 0;
22:    end if
23:    CurrentTimeInstant = NextTimeInstant;
24:    NextTimeInstant = CurrentTimeInstant
                         + TS[CurrentTimeSliceIndex];
25: end if
26: label = Choose a label from CurrentTimeSliceIndex of D[];
27: DP-SendPacket(PE_ne, label, packet);
28: End
```

received label is recorded and searched in the array of labels that was already exchanged for that time slot. These activities are shown in steps $9-13$ and $26-30$. If the labels do not match then it is an error and hence the packet is discarded.

**Algorithm 12** TicToc:Data-plane $PE_{ne}$ algorithm

**Require:** None

```
1: Begin // One time initialization starts
2: CurrentTimeSliceIndex = 0;
3: CurrentMasterClock = PTP LSP Clock Timestamp;
4: CurrentTimeInstant = CurrentMasterClock;
5: NextTimeInstant = CurrentMasterClock
                     + TS[CurrentTimeSliceIndex];
6: End // One time initialization ends
7: Begin
8: packet = DP-ReceivePacket(Interface);
9: match = CheckFEC(packet);
10: if match == 0 then
11:    return; //no match
12: end if
13: label=extractPacket(packet, LABEL);
14: if CurrentTimeInstant ≤ NextTimeInstant (configured seconds) then
15:    // do nothing;
16: else
17:    // Move by next TS[i]
18:    CurrentTimeSliceIndex++;
19:    if CurrentTimeSliceIndex == n then
20:       // check to wrap around
21:       CurrentTimeSliceIndex = 0;
22:    end if
23:    CurrentTimeInstant = NextTimeInstant;
24:    NextTimeInstant = CurrentTimeInstant
                         + TS[CurrentTimeSliceIndex];
25: end if
26: // Note that the array D must be 3 times
    // larger in this case
27: first-label = Check whether the current label is in D[]
28: if label ≠ first-label then
29:    error(); return;
30: end if
31: DP-SendPacket(CE1, NULL, NULL, packet);
32: End
```

Some remarks about the algorithms are given below:

- The label size will include that of the additional label used in the label-hopping algorithms based on hashing.
- Due to non inclusion of additional label, bit selection pattern is not needed.
- A mechanism to handle packet losses may be used when the labels desynchronize.
- This algorithm can be implemented real-time and at nearly line-rate.