Bounded Latency Spanning Tree Reconfiguration

Martino Fornasa, Michele Stecca, Massimo Maresca, Pierpaolo Baglietto

Technical Report

Computer Platform Research Center (CIPI) – University of Genova, University of Padova, Italy

***Abstract*—One of the main obstacles to the adoption of Ethernet technology in carrier-grade metropolitan and wide-area networks is the large recovery latency, in case of fault, due to spanning tree reconfiguration. In this paper we present a technique called Bounded Latency Spanning Tree Reconfiguration (BLSTR), which guarantees worst case recovery latency by accelerating bridge reconfiguration and by eliminating the bandwidth-consuming station discovery phase that follows the bridge forwarding table invalidation due to reconfiguration. BLSTR does not replace the spanning tree reconfiguration protocol (RSTP/MSTP), which remains in control of network reconfiguration, whereas it operates in parallel with it. More specifically BLSTR maintains a copy of the bridge configurations and of the bridge forwarding tables deriving from all the possible single faults, so that at the occurrence of a fault BLSTR is able to activate the appropriate configurations and to load the appropriate forwarding tables.**

***Index Terms*—Ethernet, Carrier Ethernet, Wide Area Network, Metropolitan Area Network, Spanning Tree, RSTP, Bounded latency**

I. Introduction

While in the past Ethernet technology was prevalently adopted in the local domain and in enterprise networks, recently the bandwidth growth deriving from the diffusion of optical transmission has made it convenient to adopt Ethernet technology also in the metropolitan/wide-area domain in and carrier networks. The adoption of Ethernet technology in carrier networks is mainly based on the IEEE 802.1Q standard [3], which introduces the VLAN concept to segregate the traffic related to different services in the user network, on the IEEE 802.1ad standard (Provider Bridge) [4], which introduces the stacked VLAN concept to segregate the traffic related to different customers in the service provider network, and on the IEEE 802.1ah standard (Provider Backbone Bridge) [5], which introduces a separate network associated to a private addressing space to interconnect different Provider Bridge networks. In Provider Bridge technology, which is of particular interest in this paper, the service provider bridges can be classified in two categories, namely that of Provider Edge Bridges (PEB), connected to customer equipment, and that of Provider Core Bridges (PCB), internal to the service provider network (see Fig. 3).

In general terms the Ethernet working model can be summarized by three distinctive features, namely *spanning tree, address learning,* and *flood on unknown.* The *spanning tree* feature denotes the fact that the Ethernet frames are forwarded through an acyclic overlay topology, called *active topology*, which spans all the bridges, i.e., a spanning tree. Ethernet uses the Rapid Spanning Tree Protocol (RSTP) [1] to establish such an overlay topology and the Multiple Spanning Tree Protocol (MSTP) [2], an extension of RSTP, to establish more than one spanning tree instance on the same physical topology to improve robustness and link utilization. The *address learning* feature denotes the fact that the bridge forwarding tables are updated at the reception of each frame by associating the frame source MAC address to the frame arrival port, i.e., by learning the route to a station from the traffic generated by that station. The *flood on unknown* feature denotes the fact that when a bridge receives a frame directed to an unknown MAC address the bridge floods the frame on all its active ports. Both *address learning* and *flood on unknown* require the presence of a spanning tree. In particular *address learning* requires the existence of single bidirectional paths between bridge pairs whereas *flood on unknown* is not compatible with the presence of cycles which would cause endless forwarding loops.

Being a distance-vector protocol [1], RSTP can not provide bounded reconfiguration time in case of faults and more in general in case of network modifications that worsen the network paths because of the well known count-to-infinity phenomenon [13]. In particular it was shown that the RSTP convergence may end up lasting several seconds or even tens of seconds [15]. While such a large reconfiguration latency can be acceptable in enterprise networks, on the contrary it is not compatible with carrier grade services, for which the worst acceptable reconfiguration latency is of an order of magnitude of the tens of milliseconds [7].

Several approaches to reduce Ethernet reconfiguration latency were proposed in the past. A first approach is to devise special techniques for specific physical topologies of large diffusion, such as ring [40][41]. A second approach is to exploit different MSTP instances to perform rapid rerouting of traffic after a fault [31][34]. A third approach is to abandon the Ethernet working model and to replace the spanning tree approach with a link state protocol [6][43]. A detailed discussion is provided in Section VI.

October 25, 2011.

Authors are with the Computer Platform Research Center, University of Genova, University of Padova - Italy (e-mail: {m.fornasa, m.stecca, m.maresca, p.baglietto} @cipi.unige.it).

The Bounded Latency Spanning Tree Reconfiguration (BLSTR) technique [8] proposed in this paper guarantees bounded latency of spanning tree reconfiguration after a bridge fault or after a link fault. BLSTR does not replace the spanning tree reconfiguration protocol (RSTP/MSTP), which remains in control of network reconfiguration, whereas it operates in parallel with it. More specifically BLSTR maintains a copy of the bridge configurations and of the bridge forwarding tables deriving from all the possible single resource faults, quickly propagates fault notifications at their occurrence, deactivates bridge forwarding for a limited amount of time that linearly depends on bridge time synchronization accuracy, activates the appropriate configurations and forwarding tables and activates forwarding again. As a consequence BLSTR not only provides fast reconfiguration but it also eliminates the effect of the bandwidth-consuming flooding needed to fill out the forwarding tables after reconfiguration.

BLSTR follows the direction proposed in [36][37] for the routing domain, according to which routing decisions are taken in a centralized way and then distributed to the network nodes. In the same way as [38] proposes to relay fault information on dedicated packets to spread fault information on a link-state network, BLSTR is based on distributed active fault notification and centralized alternative configuration computation. However, the spanning-tree distinctive features (tree overlay topology, distance-vector approach, root bridge concept, address learning, absence of time-to-live field in frame header) require a dedicated approach.

BLSTR exhibits the following characteristics:

- It is fully compatible with RSTP/MSTP, and as such it can be included in current generation Ethernet bridges as an additional software component.
- Its time critical operation is fully distributed, i.e., each bridge reconfigures itself in case of network faults, and as such it exhibits the same robustness as RSTP/MSTP.
- It guarantees a bounded reconfiguration latency in the order of magnitude of tens of milliseconds on large geographical networks after a bridge fault or after a link fault.
- It leverages the hardware cost reduction, as it requires at most an inexpensive upgrade of bridge memory.

The BLSTR approach is specifically targeted at Provider Bridge networks based on the IEEE 802.1ad standard. Such networks, which are adopted by carriers to provide services in the metropolitan/wide-area domain, take substantial benefit from the bounded reconfiguration latency provided by BLSTR.

The main contributions of the paper are summarized below:

- The first contribution is the idea of precomputing the bridge port configurations and the forwarding tables that would derive from all possible single resource faults and of keeping such configurations and tables in the bridge memories (Section II). While it might be objected that the implementation of such a strategy would require a significant amount of memory, the objective of enabling the deployment of Ethernet technology in carrier-grade telecommunication services by supporting bounded reconfiguration latency is so relevant to justify a memory upgrade in bridges, in particular considering the low cost of memory.
- The second contribution is the idea to have BLSTR work in parallel with RSTP (Section II). The coordination between BLSTR and RSTP is controlled in such a way that after a bridge fault or after a single link fault it is BLSTR that takes care of bridge reconfiguration, whereas in case of multiple simultaneous independent faults (the probability of which is very low), it is RSTP that takes care of bridge reconfiguration in the same way as it would do in absence of BLSTR.
- The third contribution is a technique that supports the notification of single link faults and the identification of bridge faults within a bounded time (Section II).
- The fourth contribution is a technique for synchronizing bridge reconfiguration after fault detection to guarantee the absence of temporary forwarding loops (Section II, Section III). The technique ensures that all the bridges to be reconfigured first switch off forwarding, then activate the new configurations, and finally switch on again forwarding at a time at which all the bridges to be reconfigured have already switched off forwarding, thus avoiding forwarding loops.
- The fifth contribution is a set of algorithms aimed at calculating the spanning tree configuration generated by RSTP in a centralized way (Section IV).
- The sixth contribution is a technique to compute the worst case latency of BLSTR spanning tree reconfiguration. (Section V).

II. Technique Description

A. Architecture and Operating Principles

BLSTR requires the cooperation between a central platform and a distributed platform.

The *BLSTR Central Platform (BLSTR-CP)* is responsible for offline operations and does not intervene at the moment at which a fault is detected. On the contrary it maintains an up-to-date image of the network, which includes the network topology, the topologies of the spanning tree instances (assuming that more than one spanning tree instance is in operation as provided by MSTP), the association of the end stations to the Provider Edge Bridge ports, and the alternative bridge configuration for every possible single resource fault. The alternative configurations correspond to those which would derive from RSTP. During normal operations, the alternative bridge configurations are kept up to date by the BLSTR-CP and forwarded to all bridges to be used to reconfigure the bridges in case of fault.

The *BLSTR Distributed Platform (BLSTR-DP)* consists of components hosted in the bridges and takes care of immediate reaction to faults. In particular a bridge, upon fault detection on one of its ports, injects a Fault Notification message into the network. A link fault causes two notification messages

whereas a bridge fault causes one message from each bridge neighbour. Such messages reach all the other bridges in a worst-case time proportional to the network delay diameter (i.e., the maximum time needed by a fault notification message to travel between any pair of bridges in the network). Upon reception of such a message each bridge retrieves the configuration that it is supposed to assume in the network topology deriving from the detected fault and reconfigures itself immediately. Later on, RSTP converges to the same results.

From a software point of view BLSTR is carried out by the joint action of a set of components (described in Section II.A) coordinated by means of a control strategy (described in Section II.B).

B. Components

BLSTR is carried out by the following seven components (see Fig. 1, that shows the components hosted by the BLSTR-CP, and Fig. 2, that shows the components hosted by the BLSTR-DP):

- *Time Synchronization* component *(TS)*;
- N*etwork Image Management* component *(NIM)*;
- *Alternative Network Configuration Management* component *(ANCM)*;
- *Bridge Alternative Network Configuration Management* component *(B-ANCM)*;
- *Fault Notification Distribution* component *(FND)*;
- *Fault Identification* component *(FI).*
- *BLSTR Control* component *(BC).*

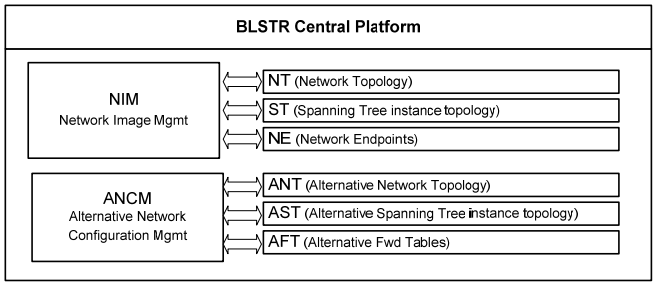

Fig. 1. BLSTR components hosted in the BLSTR-CP.

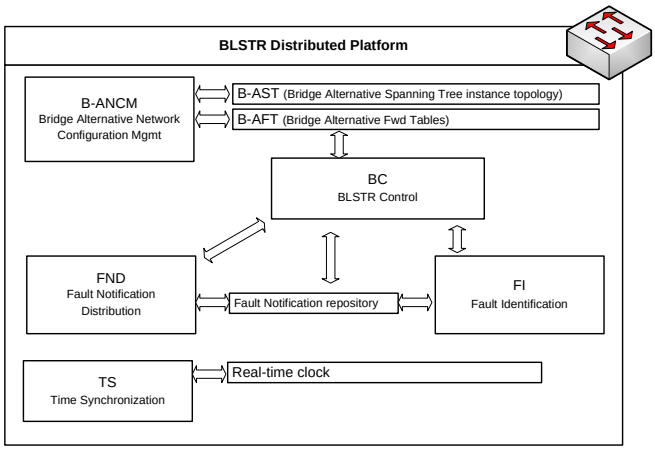

Fig. 2. BLSTR components hosted in the BLSTR-DP.

Time Synchronization component (TS)

The Time Synchronization component, hosted in the BLSTR-DP, aims at keeping the bridge clocks aligned. It may be based on a global time source or on the Network Time Protocol (NTP) [42] as well as on other synchronization protocols and must support worst-case accuracy, i.e., the guarantee that the time-of-the-day of any bridge differs from the actual time-of-the-day by at most T_s.

Network Image Management component (NIM)

The Network Image Management component, hosted in the BLSTR-CP, collects and maintains an image of the network, which includes the following information elements:

- The Network Topology, for example in the form of an adjacency matrix called $NT[N_B][N_B]$, assuming that N_B is the number of bridges in the provider network.
- The Spanning Tree Instance Topologies, for example in the form of an array of N_S Port Status Arrays called $ST[N_B][N_P][N_S]$, each corresponding to a Spanning Tree Instance, where N_S is the number of the active Spanning Tree Instances and N_P denotes the maximum number of ports in a bridge.
- The association between the Network Endpoints, i.e., the MAC addresses external with respect to the network, and the Provider Edge Bridges through which the Network Endpoints are reachable (See Fig. 3), for example in the form of an array of *X* elements called $NE[X]$, where *X* is the number of Network Endpoints (which is variable) and where each element includes the Network Endpoint MAC Address, the Provider Edge Bridge Id, the Provider Edge Customer Port, and the Provider VLAN Id.

The maintenance of the Network Image is supported by appropriate data exchanges (periodic and/or event-driven) between the NIM and the bridges.

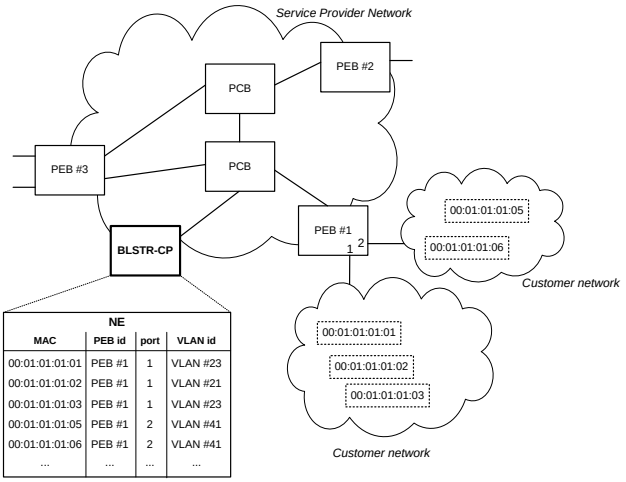

Fig. 3. Network Endpoint table (NE) for a sample Provider Bridges network (PEB: Provider Edge Bridge, PCB: Provider Core Bridge).

Alternative Network Configuration Management component (ANCM)

The Alternative Network Configuration Management component, hosted in the BLSTR-CP, computes and maintains the alternative network configurations associated to all possible resource faults, where a resource is either a bridge or a link. For all the $N_B + N_L$ resources that may fail (where N_B is the number of bridges in the network and N_L is the number of links in the network), the ANCM component computes and maintains the following information elements:

- ($N_B + N_L$) Alternative Network Topologies, for example in the form of an array of $N_B + N_L$ adjacency matrices called $ANT[N_B][N_B][N_B + N_L]$.
- ($N_B + N_L$) Alternative Spanning Tree Instance Topologies, for example in the form of an array of Spanning Tree Instance Topologies called $AST[N_B][N_P][N_S][N_B + N_L]$.
- $N_V \cdot (N_B + N_L)$ Alternative Bridge VLAN Forwarding Tables, for example in the form of an array called $AFT[N_B][N_V][X][N_B + N_L]$ of Forwarding Table entries (MAC, Port Number), where N_V indicates the number of existing VLANs and X indicates the Forwarding Table length.

The algorithms used to compute the alternative network configurations are described in Sections IV.A and IV.B.

Bridge Alternative Network Configuration Management component (B-ANCM)

The Bridge Alternative Network Configuration Management component (B-ANCM), hosted in the BLSTR-DP, keeps the bridge information base aligned with that of the ANCM. In particular the B-ANCM periodically downloads its portion of AST and AFT tables from the ANCM.

More specifically the B-ANCM maintains the following information base in each bridge:

- The ($N_B + N_L$) bridge Alternative Bridge Port Configurations, for example in the form of an array of port states called $B-AST[N_P][N_S][N_B + N_L]$.
- The $N_V \cdot (N_B + N_L)$ bridge Alternative Bridge VLAN Forwarding Tables, for example in the form of an array called $B-AFT[N_V][X][N_B + N_L]$ of Forwarding Table entries, where N_V indicates the number of existing VLANs and *X* indicates the Forwarding Table length.

Fault Notification Distribution component (FND)

The Fault Notification Distribution component, hosted in the BLSTR-DP, aims at supporting the immediate diffusion of Fault Notifications over the network. The diffusion must be completed within a time limit called Worst Case Fault Notification Latency (WCFNL), as we assume that after a fault the network continues to be connected. A technique to compute WCFNL is presented in Section V.

FND relies on a *fault notification repository* array, which contains a copy of the fault notification messages received by the bridge during a reconfiguration period.

The behaviour of FND is described in Sections II.C and III.

Fault Identification component (FI)

The Fault Identification component, hosted in the BLSTR-DP, identifies the failing resource by combining the Fault Notification messages stored in the repository. More specifically, the fault of a link is detected by combining the notifications generated by the bridges located at the link endpoints, whereas the fault of a bridge is detected by combining the notifications generated by all the bridges connected to the failing bridge, as described in Section IV.C. In case of a single resource fault, the FI supports the selection of the appropriate alternative configuration in the B-ANCM. On the other hand, if the FI component in bridge receives a set of fault notification messages that are not compatible with a single resource fault (for example, originated by multiple faults), the bridge must revert to RSTP.

BLSTR Control component (BC)

The BLSTR Control component, hosted in the BLSTR-DP, coordinates BLSTR and RSTP/MSTP to activate the appropriate bridge configuration after a fault.

First of all it must be pointed out that RSTP/MSTP is permanently running on the network, independent of the action of BLSTR. When it is BLSTR instead of RSTP/MSTP that controls the bridge configuration RSTP/MSTP keeps on running in background on a copy of the bridge configuration stored in the bridge memory.

BC moves through a series of states as shown in Fig. 4. During regular network operation BC is in a state called Normal Operation (NO), in which the bridge is ready to receive Fault Notification messages. Upon reception of a Fault Notification message, BC moves to a state called Fault Notification Collection (FNC), in which the bridge collects Fault Notification messages**.** At the expiration of a timer that guarantees that all the Fault Notification messages originating from the same fault have reached all the bridges BC processes the Fault Notification messages collected to recognize the fault and moves to a state called Single Fault (SF) during which forwarding is switched while the new bridge configuration is loaded but not yet activated. At the expiration of a timer that guarantees that all bridges have switched off forwarding BC activates the new configuration and moves to a state called Reconfigured (RC), in which the bridge configuration is maintained at least for a time equal to the longest time interval needed by RSTP to converge.

The aforementioned state evolution changes upon reception of a Fault Notification message that signals the occurrence of a multiple fault, i.e., of a Fault Notification message that does not refer to the same fault to which the Fault Notification

messages already received refer. In such a case BC evolves to a state in which BLSTR is excluded whereas RSTP/MSTP is given the control of bridge reconfiguration. The multiple fault case is discussed in Section III.

C. Control Strategy

In the following we describe the BLSTR control strategy.

Normal Operation (NO)

During regular network operation the bridges are in the state *NO*, and NIM, ANCM and B-ANCM interact with each other to maintain the network information base up to date and aligned. It is worth noticing that such interactions are triggered by network topology changes, which are supposed to happen rarely. In particular NIM maintains the network image, ANCM maintains the alternative network images and calculates the alternative configurations, and B-ANCM maintains the bridge alternative configurations aligned with ANCM. In addition TS maintains the bridge clocks aligned within a threshold T_s from the current time.

Fault Notification Collection (FNC)

The BLSTR fault reaction is based on a rapid distribution of fault information over the network carried out by the FND component.

Let B_i be a bridge in state NO (more precisely, the BC of which is in state *NO*) and let such a bridge detect a fault at time t_f on port P_j (connected to remote bridge B_k)[1]. At fault detection the bridge moves to state *FNC*, and FND performs the following actions:

- it prepares a Fault Notification message that includes the current time-of-the-day timestamp (t_f^*), the bridge Id (B_i), the port ID (P_j) and the remote bridge ID (B_k);
- it enqueues such a message on high priority egress queues on all ports; as such queues are assigned maximum priority, the Fault Notification message experiences a predictable (and small) delay (see Section V for a quantitative analysis);
- it stores the Fault Notification message in the Fault Notification repository.

Let instead B_j be a bridge in state *NO* or in state *FNC* that receives a Fault notification message. The bridge moves to state *FNC* (if necessary) and the message is processed by FND, which performs the following actions:

- it sets (or updates) the t_{OFF}^* timer, at the expiration of which, the bridge moves to state *SF*;
- it compares the incoming message with the ones contained in the Fault Notification repository, which includes all the Fault Notification messages received since the transition to state *FNC*;
- if the comparison gives a positive result then:
  - it discards the repeated Fault Notification message,

  otherwise:
  - it stores the Fault Notification message in the Fault Notification repository;
  - it enqueues the Fault Notification message on high priority egress queues on all bridge ports but the port on which the message was received.

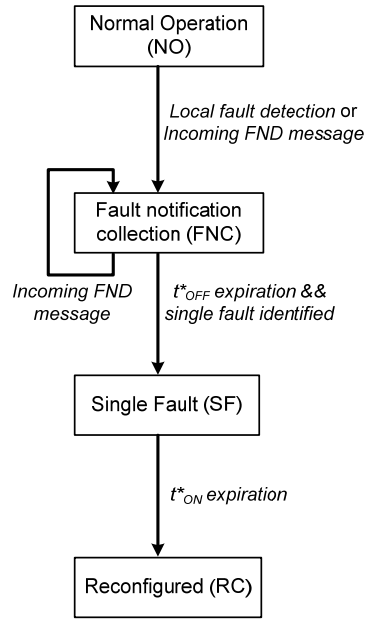

Fig. 4. BLSTR control strategy state transition diagram.

Because of the limited time accuracy, the fault detection timestamp value t_f^* included in the Fault Notification message differs from the actual fault detection time t_f. More specifically

$$t_f^* - T_s \le t_f \le t_f^* + T_s \quad (1)$$

The Fault Notification message reaches all the bridges at actual time t_r:

$$t_r = t_f + WCFNL \quad (2)$$

This is the time at which bridges can move to state *SF* to perform reconfiguration in a synchronized manner, as it can be assumed that all Fault Notification messages generated by the fault have reached all the bridges. Let $t_{OFF,i}$ be the actual time at which bridge *i* starts reconfiguration: the following inequality must hold for all bridges:

$$t_{OFF,i} \ge t_r \quad \forall i \quad (3)$$

By combining (1) and (2), (3) we obtain:

[1] A port failure can be caused by a failure on the bridge, by a link failure (e.g. a fiber cut) or by a remote-bridge failure.

$$t_{OFF,i} \geq t_f^* + T_s + WCFNL \quad (4)$$

In other words, when entering state *FNC*, BC must configure a timer to expire at the appropriate time to start reconfiguration. In order to account for the receiving bridge imperfect timer alignment the bridge must set the timer to expire at local time:

$$t_{OFF}^* = t_f^* + 2T_s + WCFNL \quad (5)$$

where t_f^* is extracted from the message whereas $WCFNL$ and T_s are extracted from the bridge configuration, and waits for its expiration.

While in state *FNC,* FND keeps on processing incoming Fault Notification messages and on updating the fault notification repository accordingly. Additionally, if necessary, that is if the t_f^* value included in the message is older than the oldest t_f^* value received so far, the BC reconfigures the timer to expire at $t_{\text{OFF}}^* = \text{oldest } t_f^* + 2T_s + WCFNL$.

At local time t_{OFF}^*, when the timer expires, the bridge moves to state *SF* in order to undertake the reconfiguration of both the Spanning Tree Instances and the VLANs influenced by the fault.

Single Fault (SF)

Because of the limited accuracy of timer synchronization the bridges enter state *SF* at different times. In order to avoid inconsistencies between the bridges still configured according to the network topology preceding the fault and the bridges configured according to the new network topology resulting from network reconfiguration, BLSTR is organized in such a way that for a period of time the network as a whole performs neither MAC Address learning nor frame forwarding.

So, when entering state *SF*, BC undertakes the following actions:

- It detaches the RSTP-driven reconfiguration process from actual bridge reconfiguration. In particular:
  - it redirects the port state updates issued by the RSTP over port state copies;
  - it suspends address Learning and Forwarding for the VLANs assigned to spanning trees subject to reconfiguration.
- It loads the configuration corresponding to the detected fault by performing the following actions:
  - it sets a timer (t_{ON}^*), at the expiration of which the new configuration is supposed to become active;
  - it replaces the obsolete Spanning Tree Instances with the new Spanning Tree Instances retrieved from the B-AST using the failing resource id as a key;
  - it replaces the Forwarding Tables of the VLANs which are influenced by the fault detected with the new Forwarding Tables extracted from the B-AFT using the failing resource id as a key.

At the expiration of timer t_{ON}^* BC moves to state *RC* in order to activate the BLSTR configuration.

We determine the value of t_{ON}^* as follows. We remind that all bridges know t_f^* and have calculated the same t_{OFF}^*. Because of the limited accuracy of clock synchronization, the actual time value $t_{ON,i}$ at which forwarding is resumed for bridge *i* can be written as:

$$t_{ON,i} = t_{OFF,i} + (t_{ON}^* - t_{OFF}^*) \quad (6)$$

To guarantee that the last bridge stops forwarding before the first bridge resumes forwarding, we impose that

$$t_{ON,j} \geq t_{OFF,i}, \ \forall i, j \quad (7)$$

Applying (6) and considering that the maximum time difference between two bridges is $2T_s$ we obtain:

$$t_{ON}^* - t_{OFF}^* \geq 2T_s \quad (8)$$

So, the value of t_{ON}^* timer has to be set to $t_{OFF}^* + 2T_s$

Reconfigured (RC)

Entering state *RC* the bridge resumes learning and forwarding for the VLANs assigned to the spanning trees subject to reconfiguration — notice that the port configuration is the one (derived from BLSTR) that has been set in state *SF*; such a port configuration remains unaltered in state *RC*.

After an appropriate time interval the bridges reset their state by reverting to state *NO*. Such an interval must be long enough to accommodate the worst-case RSTP convergence time and the time needed by BLSTR to recalculate and distribute the alternative configurations for the topology resulting from the fault. The reset process is initiated by the BLSTR-CP, which sends a reset command to all bridges in the network.

III. Multiple Simultaneous Faults

The BLSTR control strategy described in Section II does not take into account the possibility of multiple simultaneous faults, i.e. the faults of multiple bridges or of multiple links that happen in a short time interval. In particular, we consider two successive faults as multiple simultaneous faults if the second fault happens before all the bridges in the network have reverted to Normal Operation after the first fault processing.

As the bridge configuration remains blocked during state *RC*, in the case of multiple simultaneous faults BLSTR may end up worsening the RSTP reconfiguration performance. Such a worsening can be accepted as the probability of such an event is negligible. On the contrary, to handle multiple simultaneous faults appropriately the BLSTR control strategy needs to be improved as described below. The principle is to identify the occurrence of multiple simultaneous faults as soon as possible so as to have all bridges revert to RSTP. The

implementation of such a principle requires the development and the application of a technique that prevents from the occurrence of a situation in which a subset of bridges are controlled by RSTP (the ones that have identified the multiple fault) while another subset of bridges are controlled by BLSTR (the ones that have not identified multiple faults yet) as such a situation leads to inconsistencies and possibly to forwarding loops. The technique consists of having a bridge that detects a multiple fault deactivate forwarding and learning on its ports immediately. Forwarding and learning will be reactivated only at a time at which it can be guaranteed that all the bridges in the network have identified the multiple faults.

The state transition diagram is enriched with the following states, as shown in Fig. 5:

- Multiple Fault (*MF*): accessed immediately upon reception of a Fault Notification message that signals multiple faults. When entering state *MF* the bridge deactivates forwarding and learning.
- Reverting to RSTP (*RV*): accessed at a time at which it can be guaranteed that all the bridges in the network have left the *RC* state.

The bounded-latency behaviour of the fault distribution mechanism assures that, although the Fault Notifications messages may reach the bridges at different times and in different order, it is not possible that two or more bridges detect different single faults during the *FNC* phase. On the other hand, it is possible that a subset of bridges detect a single fault, while another subset detect a multiple fault, leading to an inconsistency between bridges. In any case, eventually a multiple fault notification reaches all the bridges in the network causing their transition to *MF* and the resulting deactivation of forwarding.

At transition from state *MF* to state *RV* port state control is reverted to RSTP and forwarding and learning are enabled; such a transition must take place only after all bridges in the network have recognized the simultaneous multiple fault. This can be obtained by setting a timer t^*_{RSTP} configured as follows:

$$t^*_{\mathrm{RSTP}} = \text{latest}\, t^*_f + 2T_s + WCFNL \qquad (10)$$

At the expiration of timer t^*_{RSTP} the bridge performs the transition to state *RV*. The timer is based on the latest (i.e., the most recent) timestamp among all the Fault Notification messages received because at time t^*_{RSTP} it is certain that at least the same set of messages that caused the multiple fault identification in the given bridge have reached all the other bridges in the network, assuring multiple fault identification and transition to *MF* state on all bridges.

The process of reverting to normal operation from state *RC* (in the single fault case) or from state *RV* (in the multiple fault case) is initiated by the BLSTR-CP, that sends a reset command to all bridges. However, a Fault Notification message arriving during such a process might leave the network in an inconsistent state, as the fault can reach bridges that have already performed the transition (being in *NO* state) and bridges that have not yet performed the transition.

So, suppose that the BLSTR-CP issues a reset command to instruct bridges to revert to *NO* at time t^*_{NO}. In order to avoid inconsistencies we use the t^*_{NO} value as a reference: if a bridge receives a fault notification message having a timestamp that is less than t^*_{NO}, it cancels the reset command. If the cancel operation takes place in a bridge that already performed transition (being in state *NO* or *FNC*), the bridge immediately moves to *MF* state. The above comparison is performed between two timestamps, so its outcome is deterministic and independent of bridge clock accuracy.

The multiple fault control strategy pseudocode is provided in Fig. 6.

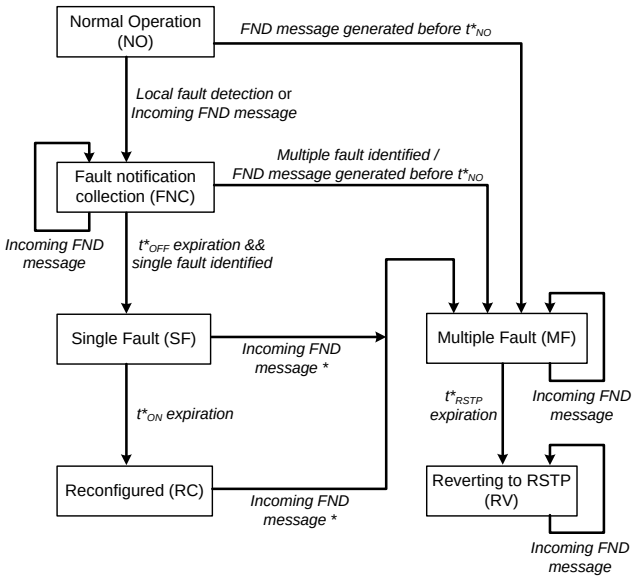

Fig. 5. BLSTR multiple resource fault control strategy (* A FND message arriving to bridges in *SF* or *RC* denote always a multiple fault).

```
function
fault_notification_received_or_fault_detected(msg
) {
    if (msg is in the repository)
        return;
    store_in_repository(msg);
    propagate(msg);
    if (multiple fault identified || msg.ts <
t*NO) {
        state = MF;
        t*OFF = t*ON = t*NO = 0;
        disable_forwarding();
        disable_BLSTR_configuration();
        t*RSTP = newest_ts + WCFNL + 2 * Ts;
    } else if (state == NO) {
         oldest_ts = msg.ts;
         t*OFF = oldest_ts + WCFNL + 2 * Ts;
         state = FNC;
    } else if (state == FNC) {
         if (msg.ts < oldest_ts) {
             oldest_ts = msg.ts;
             t*OFF = oldest_ts + WCFNL + 2 * Ts;
         }
    }
}

function timer_expiration(timer) {
    if (timer == t*OFF) {
        state = SF;
        disable_forwarding();
        t*ON = t*OFF + 2 * Ts;
        enable_BLSTR_configuration();
    } else if (timer == t*ON) {
        state = RC;
        enable_forwarding();
    } else if (timer == t*RSTP) {
        state = RV;
        enable_forwarding();
    } else if (timer = t*NO) {
        state = NO;
    }
}

function reset_command_received(cmd) {
    t*NO = cmd.t*NO;
}
```

Fig. 6. BLSTR multiple resource fault control strategy pseudocode. The `enable_BLSTR_configuration()` function detaches the RSTP from actual port configuration and configures ports according to the detected single fault. The `disable_BLSTR_configuration()` function gives back port control to RSTP. The `propagate()` function enques the fault notification message on all the bridges ports but the port on which the message has been received.

IV. ALGORITHMS

We present here the BLSTR algorithms. In order to simplify the presentation, we discuss only the single spanning tree/single VLAN case. The extension to the MSTP and multiple VLANs is trivial.

A. Centralized spanning tree calculation and port role assignment

RSTP is a distributed algorithm that calculates a deterministic spanning tree over an arbitrary topology of bridges. We discuss here an algorithm to calculate the same spanning tree active topology resulting from the distributed computation described in the 802.1D standard [1] in a centralized way. Such an algorithm is used by the ANCM component to compute centrally an Alternative Spanning Tree that would become active as a consequence of a resource fault, for all possible bridge or link faults.

In RSTP the spanning tree overlay topology is implemented by means of per-port per-VLAN state variable called *port role*. RSTP port roles are the following:

- Root: The root port is associated with the link toward the root bridge and belongs to the active topology.
- Designated: The designated ports are associated with the links toward the leaves of the spanning tree and belong to the active topology.
- Alternate**:** The alternate ports indicate alternative paths to the root bridge and do not belong to the active topology.
- Backup: Backup ports are present only with shared-media LANs and do not belong to the active topology.

The algorithm processes an alternative network topology (i.e., one element of ANT), which includes the 802.1D bridge parameters (bridge id, ports id, port path costs) and produces an alternative spanning tree instance expressed as a set of port roles (AST) for every bridge in the network.

We consider a network composed only by point-to-point links, whereas we do not consider the case of shared media LANs which is obsolete. Under such a hypothesis, we model the network as a weighted graph, in which the nodes correspond to the bridges while the edges correspond to the links. The link weight is the spanning tree *port path cost* parameter, which by default is inversely proportional to the link data rate. The only exception is represented by 'parallel' links, i.e. multiple links between the same pair of bridges[2].

The algorithm, which calculates the same active topology as the one obtained by the RSTP protocol, is based on the Dijkstra algorithm, with some modifications needed to emulate the RSTP behaviour. The algorithm consists of the following steps:

- Step 1. Root bridge selection: Every 802.1D-bridge has a unique bridge identifier (ID), which is composed by the bridge address and a user-manageable priority field. The root bridge is the bridge with the smallest ID on the network. As bridge addresses are unique, the root bridge selection is deterministic.
- Step 2. Shortest-path identification: The shortest path from the root bridge to every other bridge is calculated. This is known as single-source shortest path problem, addressed by the well known Dijkstra algorithm. The graph representing all such paths is a spanning tree. However, as there may exist multiple minimum-cost spanning trees based on the root bridge, the algorithm must select the bridge with the smallest ID as the next node to visit.
- Step 3. Port roles assignment: The last step consists of assigning port roles. For each link that is part of the spanning tree, the port on the highest-ID bridge takes the

[2] In case of parallel links, i.e. multiple links between the same pair of bridges, only one of the link will be part of the active topology. In such a case, the tie-breaker is represented by the comparison of port IDs.

designated role and the port on the lowest-ID bridge takes the root role. For other links, the port on the highest-ID bridge takes the designated role whereas the others take the alternate role.

B. *Bridge per-VLAN forwarding table calculation*

We discuss here an algorithm to pre-calculate the forwarding tables of each provider bridge. The pre-calculation of the forwarding tables for all destinations aims at avoiding the flooding deriving from forwarding table reset, thus greatly reducing the bandwidth consumed after spanning tree reconfiguration.

We suppose that the MAC addresses of the end-stations attached to a customer edge port on a provider edge bridge are locally learned and collected by such a bridge, and such information is transmitted to the NIM and stored in Network Endpoints data structure (NE). Every NE entry includes the following fields:

- External Endpoint MAC Address (`MAC`),
- Provider Edge Bridge Id (`peb_id`),
- Provider Edge Customer Port (`peb_port`),
- Provider VLAN Id (`VLAN_id`).

As said before, for simplicity we do not consider here the VLAN.

The input of each run of the algorithm is a NE entry, and the output is a row of the forwarding table of each bridge in the network. So, for every NE entry, the algorithm performs the following steps:

- Step 1. Provider edge bridge entry: The algorithm creates a forwarding table entry for the bridge identified by `peb_id`. The new entry will be the following:

  (`MAC, peb_port`)
- Step 2. Tree-climb: The algorithm starts considering the bridge identified by `peb_id`, and climbs the tree up to the root bridge, creating a forwarding table entry for each traversed bridge. The tree is traversed entering by port `d_port` and going out by the root port. The new entry will be the following:

  (`MAC, d_port`)
- Step 3. Other bridges: Finally, the algorithm sets a forwarding table entry for all non-traversed bridge. Such an entry will be the following:

  (`MAC, root_port`)

It is certainly possible to improve the above algorithm, considering that any algorithm delivering the same results can be used. For example, it is possible to aggregate all MAC addresses pertaining to the same spanning tree and to the same port. However, the per-VLAN forwarding table calculation is performed offline thus it doesn't influence the online critical path represented by the reconfiguration phase.

C. *Fault identification criteria*

During the Fault Recovery phase, every bridge receives a number of fault notification messages, as more than one bridge may detect the fault. Such messages are stored by the FND in the Active Fault Notification repository. We discuss now the fault identification criteria employed by the FI in order to activate the correct alternative configuration in case of single resource fault. There are two cases: the fault of a link and the fault of a bridge.

The fault of a link is detected by the two link terminating bridges, which generate a fault notification message. In Fig. 7, for example, the link between bridge 222 and bridge 444 fails, causing the generation of exactly two fault notification messages, with B_i and B_k values reversed.

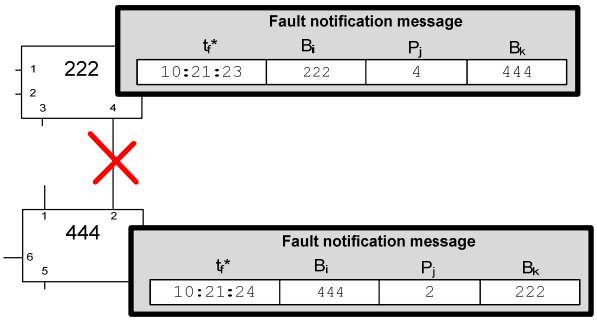

Fig. 7. Link fault. The link between bridge 222 and bridge 444 fails, causing the generation of two fault notification messages.

The fault of a bridge is detected by all the neighbours of that bridge as multiple faults. So, the fault of a bridge causes the generation of at least one fault notification message. All the fault notification messages have the same values in the B_k field. In Fig. 8 bridge 222 fails. As a consequence, bridges 111, 333 and 444 will issue a fault notification message for every link to bridge 222.

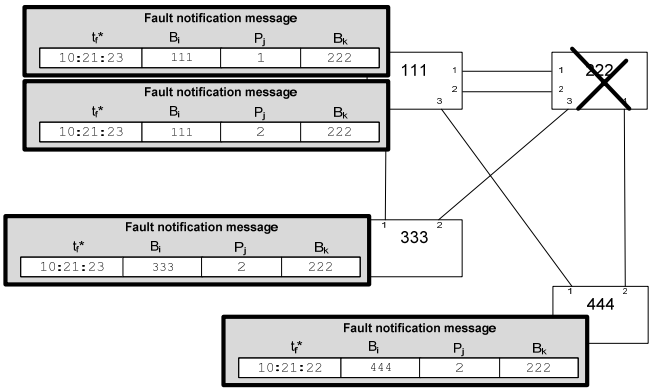

Fig. 8. Bridge fault. Bridge 222 fails, causing the generation four fault notification messages.

V. Performance Analysis

In this section we analyze the BLSTR performance.

First we want to obtain the single-fault BLSTR worst-case fault recovery time (T_{MAX}), i.e. the time interval from fault (t_f) to last bridge reconfiguration ($t_{ON,i}$) in the event of a single fault. T_{MAX} can be written as:

$$\begin{aligned} T_{MAX} &\underset{(6,8)}{=} \max_i(t_{ON,i} - t_f) \\ &\underset{(4)}{=} \max_i(t_{OFF,i} + 2T_S - t_f) \\ &= \max(t_f^* + 5T_S + WCFNL - t_f) \\ &= WCFNL + 6T_S \end{aligned} \quad (9)$$

The value T_s depends on the clock synchronization accuracy, while the value of *WCFNL* can be determined as follows.

We consider a network composed of N_B bridges linked by N_L links; the worst-case network diameter will be $N_B - 1$. The crossing time for the *i*-th bridge can be written as:

$$t_{CR}^i = t_P^i + t_R^i + t_{TX}^i + t_Q^i \quad (11)$$

where:

- t_P^i is the propagation time on *i*-th link, which depends on the geographical length of such a link.
- t_R^i is the processing time on *i*-th bridge, which depends on hardware characteristics.
- t_{TX}^i is the fault notification message transmission time. Being s_{FN} the size of such a message and r_i the rate of the output link, we have: $t_{TX}^i = s_{FN}/r_i$.
- t_Q^i is the fault notification message queuing time. Due to the higher priority of fault notification message, t_Q^i is the sum of the transmission time of a full-sized data frame (s_{MTU}/r_i) plus the transmission time of other fault notification messages that could have been already enqueued in the output queue. We distinguish two cases:
  - Link fault. In case of link fault, two fault notification messages are generated by the two bridges that terminate the link. So in the worst case t_Q^i in the worst case can be written as:

$$t_{Q,WC}^i = \frac{s_{MTU}}{r_i} + \frac{s_{FN}}{r_i}$$

  - Bridge fault. In case of fault of a bridge having N_P ports, a fault notification message is generated from each neighbour bridge. So, in the worst case t_Q^i can be written as:

$$t_{Q,WC}^i = \frac{s_{MTU}}{r_i} + (N_P - 1)\frac{s_{FN}}{r_i}$$

In case of link fault, WCFNL can be written as:

$$WCFNL = \max_{(\text{all paths})} \sum_i t_{CR}^i = \max_{(\text{all paths})} \left(\sum_i (t_P^i + t_R^i) + \sum_i \frac{2s_{FN} + s_{MTU}}{r_i} \right) \quad (12)$$

Analogously in case of bridge fault WCFNL can be written as:

$$WCFNL = \max_{(\text{all paths})} \sum_i t_{CR}^i = \max_{(\text{all paths})} \left(\sum_i (t_P^i + t_R^i) + \sum_i \frac{N_P s_{FN} + s_{MTU}}{r_i} \right) \quad (13)$$

In most realistic environments we can safely assume that the propagation and processing terms will dominate the expression. For example, with a MTU of 1500 B and a link capacity of 1 Gbps, we obtain a MTU/r value of 12 μs. Applying such an assumption, we can express T_{MAX} as:

$$T_{MAX} = 6T_s + WCFNL \approx 6T_s + \max_{(\text{all paths})} \sum_i (t_P^i + t_R^i) \quad (14)$$

i.e. the maximum recovery time in case of single fault is the sum of six times the clock synchronization accuracy plus the network delay diameter, including in the network delay diameter the notification message queueing time. To give an example, if we assume a synchronization of T_S =1 ms, BLSTR can provide sub-50 ms recovery latency in a network with a delay diameter up to 44 ms.

In the multiple fault case, the worst-case reconfiguration time corresponds to the RSTP worst-case reconfiguration time. However, the best-case reconfiguration time can be higher than the RSTP one, being influenced by the reverting to RSTP mechanism. In particular, bridges revert to RSTP at local time $t_{\text{RSTP}}^* = \text{latest}\, t_f^* + 2T_s + WCFNL$, see Eq. (10). So, taking into account the clock accuracy, we can say that all bridges in the network will revert to RSTP by the time:

$$t_{\text{RSTP}} = \text{last\_fault}\, t_f + 4T_s + WCFNL \quad (15)$$

i.e., in case of multiple faults, all bridges revert to RSTP at worst four times the clock synchronization accuracy plus the network delay diameter after the last fault.

VI. Comparison with Previous Work

Surveys on the evolution of carrier-grade Ethernet technologies can be found in [9] and [10]. Another survey, focused on Ethernet resilience mechanisms is presented in [11].

In the following we summarize some previously proposed approaches to cope with Ethernet inefficiencies.

A. Non-arbitrary topologies

A number of approaches exist to handle RSTP slow re-convergence that work in non-arbitrary topologies, such as ring topologies (EAPS [40], ERPS [41]) and parallel link aggregation [39], but they can not be used in arbitrary topologies. On the contrary, BLSTR approach can be applied to arbitrary topologies.

B. Replacing spanning tree with a link state routing protocol

A growing number of proposals in the literature [14][17][18][19][20] and in standardization bodies [6][43] suggest the elimination of spanning-tree based forwarding on Ethernet networks by applying a link state routing protocol to layer-2, i.e., by forwarding frames on shortest-path routes.

In [14] the authors propose to use a link state protocol for forwarding and a distributed directory service providing station location registration in order to eliminate the need of flooding. Using such an approach, a new station connecting to the network has to register itself to the directory service.

SEATTLE [20] proposes a distributed directory mechanism composed by two parts: i) by running a link state protocol, each bridge learns the shortest path to every other bridge in the network, ii) an hash function is used to map end-station information to a bridge. This approach promises to reduce the amount of distributed information, achieving better scalability.

Recently two link state approaches have emerged: the RBridges/TRILL, sponsored by IETF, and the Shortest Path Bridging, sponsored by the IEEE.

RBridges [16] is a campus-level architecture based on a link-state routing in which frames are encapsulated in an additional layer-2 header containing a TTL counter in order to avoid forwarding loop persistence; additionally, station address learning is only performed locally and the obtained information is distributed to the entire network along with topology information. RBridges is specifically targeted to campus-wide and datacenter networks. Such a proposal is under active standard development in IETF Transparent Interconnection of Lots of Links (TRILL) working group.

Shortest Path Bridging (802.1aq) [6] is an amendment to the IEEE 802.1Q standard providing Ethernet frame forwarding on shortest path by using the IS-IS protocol.

Approaches based on link state routing protocol require an extensive modification of the Ethernet working model, based on the elimination of the spanning tree approach in favour of shortest-path routing approach, often involving a modification of the Ethernet frame format based on the inclusion of additional headers. On the contrary, BLSTR maintains the spanning tree-based forwarding approach, and uses network global topology information only to precalculate spanning tree instances and forwarding tables.

C. MSTP-based recovery schemes

Several mechanisms have been proposed to exploit MSTP to perform a *global recovery* scheme [23][24][25][27] [30][31], i.e., to take advantage of the use of two disjoint spanning tree paths (primary and backup) between each pair of border bridges. Such paths are usually pre-computed offline. When a resource (link or node) fails, the ingress node, informed of the fault, switches the traffic on the backup path. Such approaches usually rely on the assignment of a single VLAN to an instance of MSTP, and use the VLAN IDs to select the spanning tree instance. This prevents from using all the available VLAN IDs to classify and segregate customer traffic and poses scalability issues.

On the other hand, in a *local recovery* scheme ([26][28][29][31][32][33][34]), the fault of a resource triggers the rerouting of the traffic on a pre-signalled backup path and such a reroute is controlled by the upstream node with respect to the detected fault. Local recover mechanisms exist in SONET/SDH rings and have been proposed recently in MPLS [35]. Some local recovery approaches ([26][28][29][31]) take advantage of multiple spanning tree instances as backup paths while some others propose different mechanisms: in [32] the authors propose a local recovery mechanism that responds to link fault by locally rerouting the traffic on a path pre-computed via integer linear programming that is not on the spanning tree; the same authors extended the concept to simultaneous double link faults in [33].

The MSTP-based approaches presented above use a multiple spanning tree approach, but they explicitly build spanning tree instances without being compatible with MSTP/RSTP protocols. Additionally, they use VLAN IDs to select switching paths, preventing the use of VLAN tagging to segregate traffic. On the contrary, BLSTR supports as many spanning tree instances as MSTP, as it does not rely on explicit path building, and does not exploit VLAN IDs.

D. Other approaches

In [15] the authors propose *RSTP with Epochs*, a RSTP modification that adds a sequence number to RSTP BPDUs in order to suppress stale topology information from the network thus avoiding the count-to-infinity behaviour. Such an approach aims at resolving a specific issue (count-to-infinity behaviour) and does not address bounded latency or flooding issues.

VII. Conclusion

The fact that RSTP is based on distance-vector leads to high worst-case reconfiguration latency in Ethernet networks because of the possibility of count-to-infinity. Such a high reconfiguration latency is an obstacle to the deployment of Ethernet technology in the carrier domain.

A possible radical approach is to replace entirely the spanning tree scheme with a link-state algorithm, as advocated for example by the IEEE 802.1aq standard.

Instead, in BLSTR we propose a hybrid approach: we maintain a local-information-based spanning tree approach, characterized by the robustness of a distributed system, but we extend it through the addition of a mechanism to pre-compute the bridge configurations that would result from all possible single resource faults so as to be able to activate them immediately upon fault detection. In this way, the performance of network reconfiguration upon fault detection depends on the speed of fault notification distribution and on the accuracy of bridge clock synchronization. In particular, in case of single resource fault the worst-case fault recovery latency is the sum of the network delay diameter plus six times the bridge clock synchronization accuracy, while in case

of multiple faults the worst-case recovery time is the same of RSTP protocol.

Additionally, the pre-calculation of the bridge forwarding tables allows retaining the Ethernet plug-and-play distinctive characteristics, namely the address learning capability and the flood-on-unknown capability, while avoiding the bandwidth-consuming flooding phase needed to re-populate the forwarding tables after their reset.

In RSTP the impact of a fault on network operation depends on the location of the fault in the spanning tree: a fault close to the root bridge has a larger impact with respect to a fault close to the leaves, as only the bridges contained in a subtree rooted at the fault are subject to a root path cost change [12]. A possible improvement to BLSTR will consist of exploring the possibility of adapting dynamically the WCFNL value to the network portion that is impacted by the specific fault in order to improve the reconfiguration time in the average case.

REFERENCES

[1] "IEEE 802.1D-2004. IEEE Standard for Local and Metropolitan Area Networks - Media access control (MAC) Bridges (Incorporates IEEE 802.1t-2001 and IEEE 802.1w)", Clause 17.

[2] "IEEE 802.1Q-2005 IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Revision", Clause 13.

[3] "IEEE 802.1Q-2005 IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Revision", Clause 6.

[4] "IEEE 802.1ad-2005 IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Revision - Amendment 4: Provider Bridges".

[5] "IEEE 802.1ah-2008 IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment 7: Provider Backbone Bridges".

[6] "IEEE 802.1aq Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks - Amendment 8: Shortest Path Bridging".

[7] ANSI T1.TR.68-2001, "Enhanced Network Survivability Performance".

[8] M. Maresca, "Metodo per l'accelerazione della riconfigurazione in reti informatiche basate su spanning tree", Italian Patent GE2010A000037, April 15, 2010.

[9] J. Sommer, S. Gunreben, F. Feller, M. Kohn, A. Mifdaoui, D. Sass, J. Scharf, "Ethernet. A survey on its fields of application", *IEEE communications surveys and tutorials*, Vol. 12, 2010.

[10] R. C. Sofia, "A Survey of Advanced Ethernet Forwarding Approaches", *IEEE Communications Surveys & Tutorials*, Vol. 11, No. 1, First Quarter 2009.

[11] M. Huynh, S. Goose, P. Mohapatra, "Resilience technologies in Ethernet", *Computer Networks*, Vol. 54, No. 1, pp. 57-78, 2010.

[12] J. Jaffe, F. Mos, "A Responsive Distributed Routing Algorithm for Computer Networks", *IEEE Trans. on Communications*, vol. 30, no. 7, July 1982.

[13] J. J. Garcia-Lunes-Aceves. "Loop-free routing using diffusing computations", *IEEE/ACM Trans. Netw*., Vol. 1, Issue 1, February 1993.

[14] Myers, Eugene Ng, Zhang., "Rethinking the Service Model: Scaling Ethernet to a Million Nodes", in *Proc. of Third ACM Workshop on Hot Topics in Networks (HotNets-III),* San Diego, CA, November, 2004.

[15] K. Elmeleegy, Alan L. Cox, T.S. Eugene Ng, "Understanding and Mitigating the Effects of Count to Infinity in Ethernet Networks", *IEEE/ACM Trans. on Netw.,* Vol. 17, No. 1, February 2009.

[16] R. Perlman, "Rbridges: Transparent Routing", in *Proc. of Infocom 2004.*

[17] R. Garcia, J. Duato, F. Silla, "LSOM: A Link State Protocol Over Mac Addresses for Metropolitan Backbones Using Optical Ethernet Switches", in *Proc. of NCA 2003*

[18] T. Rodeheffer, C. Thekkath, D. Anderson, "SmartBridge: A Scalable Bridge Architecture", in *Proc. of Sigcomm 2000*.

[19] K. Lui, W. Lee, K. Nahrstedt, "STAR: A Transparent Spanning Tree Bridge Protocol with Alternate Routing", *ACM Sigcomm Computer Communications Review*, July 2002.

[20] C. Kim, M. Caesar, J. Rexford, "Floodless in SEATTLE: a scalable ethernet architecture for large enterprises", in *Proc. of Sigcomm 2008.*

[21] K. Elmeleegy, Alan L. Cox, "EtherProxy: Scaling Ethernet By Suppressing Broadcast Traffic", in *Proc. of Infocom 2009*.

[22] G. Mirjalily, M.H. Karimi, F. Adibnia, S. Rajai, "An approach to select the best spanning tree in Metro Ethernet networks", CIT 2008.

[23] S. Sharma, K. Gopalan, S. Nanda, T. Chiueh, "Viking: a multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks", in *Proc. of Infocom 2004*.

[24] J. Farkas, C. Antal, L. Westberg, A. Paradisi, T. R. Tronco, V. Garcia de Oliveira, "Fast Failure Handling in Ethernet Networks", in *Proc. of ICC 2006*.

[25] M. Huynh, P. Mohapatra, "Etherlay: An Overlay Enhancement for Metro Ethernet Networks", in *Proc. of ICC 2006.*

[26] M. Huynh, P. Mohapatra, "Cross-Over Spanning Trees Enhancing Metro Ethernet Resilience and Load Balancing", in *Proc. of Broadnets 2007.*

[27] A. Iwata, A., Y. Hidaka, M. Umayabashi, N. Enomoto, A. Arutaki, "Global Open Ethernet (GOE) System and its Performance Evaluation", *IEEE Journal on Selected Areas in Communications*, Vol. 22, Issue 8, Oct. 2004.

[28] J. Qiu, G. Mohan, K. Chaing Chua, Y. Liu, "Local restoration with multiple spanning trees in metro Ethernet", in *Proc. of ONDM 2008.*

[29] L. Su, W. Chen, H. Su, Z. Xiao, D. Jin, L. Zeng, "Ethernet Ultra Fast Switching: A tree-based local recovery scheme", in *Proc. of ICCS 2008.*

[30] A. F. De Sousa, "Improving Load Balance and Resilience of Ethernet Carrier Networks with IEEE 802.1S Multiple Spanning Tree Protocol", in *Proc. of ICNICONSMCL 2006*.

[31] M. Huynha, P. Mohapatra, S. Goose, "Spanning tree elevation protocol: Enhancing metro Ethernet performance and QoS", *Computer Communications,* Vol. 32, Issue 4, March 2009.

[32] J. Qiu, Y. Liu, G. Mohan, K. C. Chua, "Fast Spanning Tree Reconnection for Resilient Metro Ethernet Networks", in *Proc. of ICC 2009.*

[33] J. Qiu, G. Mohan, K. C. Chua, Y. Liu, "Handling Double-Link Failures in Metro Ethernet Networks Using Fast Spanning Tree Reconnection", in *Proc. of Globecom 2009.*

[34] P.M.V. Nair, S.V.S. Nair, M. Marchetti, G. Chiruvolu, M. Ali, "Bandwidth sensitive fast failure recovery scheme for Metro Ethernet", *Computer Networks,* Vol. 52, Issue 12, June 2008.

[35] P. Pan et al., "RFC 4090. Fast Reroute Extensions to RSVP-TE for LSP Tunnels".

[36] K.-W. Kwong, L. Gao, R. Guerin, Z.-L. Zhang, "On the Feasibility and Efficacy of Protection Routing in IP Networks", *IEEE/ACM Trans. Netw*., Vol. 19, Issue 5, October 2010.

[37] H. Peterson, S. Sen, J. Chandrashekar, L. Gao, R. Guerin, Z.-L. Zhang, "Message-efficient dissemination for loop-free centralized routing", *ACM Sigcomm Computer Communications Review*, Volume 38 Issue 3, July 2008.

[38] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, I. Stoica, "Achieving convergence-free routing using failure-carrying packets", in *Proc. of Sigcomm 2007.*

[39] "IEEE 802.1ax-2008 IEEE Standard for Local and Metropolitan Area Networks - Link Aggregation".

[40] "RFC 3619. Extreme Networks' Ethernet Automatic Protection Switching (EAPS). Version 1".

[41] "ITU-T G.8032. Ethernet ring protection switching".

[42] "RFC 5905. Network Time Protocol Version 4: Protocol and Algorithms Specification".

[43] IETF Transparent Interconnection of Lots of Links WG (TRILL) 'http://tools.ietf.org/wg/trill/.