Building Ethernet Connectivity Services for Provider Networks

Eduard Bonada i Cruells

Tesi Doctoral UPF / 2012

Dirigida per
Dra. Dolors Sala i Batlle
Departament de Tecnologies de la Informació i les Comunicacions

$\Sigma\tau ους\ δυο\ μας$

"Si em llevés i fos rei,
reuniria a les corts per incloure en el codi penal
que es prohibeixi a la gent anar pel món
buscant res que no pugui anomenar."
(De la cançó *Flor Groga* de *Manel*)

"If I woke up as king,
I would convene the court to include in the penal code
that people are forbidden to wander through the world
in search of something they cannot name."
(From the song *Flor Groga* by *Manel*)

Acknowledgements

First of all, I want to sincerely thank my advisor. Without her help, her guidance and her support, I wouldn't have been able to complete this work. Her continuous dedication helped me to overtake all the obstacles that I encountered in my way. Thank you Dolors!

Also, specially thanks to my colleagues and friends from the second floor of Tànger building for providing me a lot of support and companionship. Thank you for your technical discussions, coffee breaks, economical lectures, delightful dinners, running sessions, friday after-lunch procrastination, etc. Thank you guys!

Obviously I also want to thank my whole family for supporting me. Surely they are the ones who have believed in me all this time, even more than myself. Thanks a lot. (Evidentment també full agraïr a la meva família tot el support que m'ha donat. Segurament ells són els que més han cregut en mi durant tot aquest temps, fins i tot més que jo mateix. Moltes gràcies!)

Thanks to her. I cannot find enough words to describe my gratitude for being there. This is why I dedicate to her the entire thesis with more than 70.000 words. Anna, $\epsilon\upsilon\chi\alpha\rho\iota\sigma\tau\omega$!

And finally, I want to thank this list of people for the enjoyable moments: Paul, John, George, Ringo; Jarvis, Nick, Candida, Steve, Russell; David, Marc, Dani, Xavi, Alfons; Marcus, Winston, Ben, Ted; Guillem, Arnau, Marti, Roger; Damon , Graham, Alex, Dave; Ernest, Jordi, Jordi; Zach, Perrin, Jason, Nick, Kristin, Paul , Jon, Kelly, Tracy, Ben, Lucy; Joan Miquel, Pau, Pere Manel, Jaume, Joan; Win, Regine, Richard, William, Tim, Sarah, Jeremy; Joan, Candid, David, Lluis, Marina; Joe, Mick, Paul, Terry, Nicky; Ian, Bernard, Stephen, Gillian, Phil, Tom, Peter; Robert, Simon, Roger, Jason, Reeves; Morrissey, Johnny, Andy, Mike, Dale, Craig; Debbie, Chris, Clem, Leigh, Matt, Tommy; Chris, Mike, Lee, Chas, Graham, Daniel; Sid, Steve, Paul, Glen, John; Pere, Joan, Aleix, Lluis; Dani, Joan Enric, Eduard, Ferran; Freddie, Brian, John, Roger; Roger, David, Nick, Richard; David; Brian, Mike, Al, David, Bruce; and Ludwig.

Abstract

The continuous growing demand of data services in residential and enterprise customers has motivated the network providers to deploy more effective networking technologies in their infrastructures. Ethernet's good properties offer providers an opportunity for using it at very large scale replacing the existing ATM/SONET and even in some extent IP networking. However, Ethernet was originally designed for LANs without very strict requirements, and using it as a carrier-grade technology represents a new application that leads to new requirements such as quick recovery, efficient resource utilization, better path selection and provision of path control.

In Ethernet technology the Spanning Tree Protocol (STP) is responsible of establishing the connectivity of the different Ethernet segments in a single interconnected network. The STP creates an active tree on top of the physical topology to avoid potential loops and allows the bridge functionalities to work properly. There are some implications of using STP to prune the physical topology into an active tree. One, selecting a single path to connect two peers eliminates all potential redundancy that might be available through extra links. Two, the communication over this single tree only provides optimal communication paths between nodes in the same tree branch. Three, STP takes tens of seconds to recover from a failure while providers only allow a maximum of 50msec. Although this performance has been improved with the introduction of the Rapid Spanning Tree Protocol (RSTP), it still takes tens of seconds to recover from some crucial situations when the Root of the tree fails because it creates the count-to-infinity problem. And four, the paths that form the tree are elected only based on the link cost while provider networks require more flexibility.

In this thesis we propose a complete solution based on RSTP extensions that addresses each one of the listed shortcomings. Our approach is based on understanding the fundamentals of the single tree protocol to identify the limitations and design the concrete extensions required. First of all, we perform a detailed characterization of RSTP providing a comprehensive description of its operations as well as an analysis of the protocol behavior in the most common situations. The study of the count-to-infinity effect allows us to identify the real causes and a set of unexpected side-effects that the actual reasons of the long recovery.

The deep understanding of the original protocol allows us to design RSTP-Conf as an extension to RSTP that addresses the long failure recovery experienced by RSTP because of the count-to-infinity effect. It is based on a simple yet effective confirmation mechanism that avoids the count-to-infinity detecting whether the Root has really failed or not. We evaluate the performance of RSTP-Conf by means of network simulation and results show that this technique efficiently reduces the recovery from the Root failure to at most twice the propagation delay through the network diameter.

The rest of shortcomings are addressed by expanding the active topology to multiple trees. RSTP-SP deploys an active topology based on configuring one tree per node. This allows operating with shortest-paths in all communications and provides an increase of resource utilization because chances are that a link

is active in at least one of the trees. The main challenge of this multiple tree structure is that it requires a careful selection of such trees in order to keep the symmetrical communication property that Ethernet bridging requires. RSTP-SP extends RSTP and introduces the required changes so as to construct the shortest-paths multiple trees keeping the symmetry requirement. The RSTP-SP performance is compared to IEEE 802.1aq Shortest Path Bridging and results show that that RSTP-SP outperforms SPB in terms of recovery time and outage experienced but the message overhead introduced by RSTP-SP is higher than in the SPB case.

Resum

La contínua i creixent demanda de serveis de dades per part de clients residencials i empresarials ha motivat als proveïdors de xarxes per desplegar tecnologies més eficaces a les seves infraestructures. Les bones propietats d'Ethernet presenten una oportunitat perquè els proveïdors l'utilitzin en els seus desplegaments a gran escala substituint les xarxes existents basades en ATM/SONET. No obstant, Ethernet va ser dissenyada originalment per ser implementada en LANs sense requisits molt estrictes. Que s'utilitzi com una tecnologia a nivell de proveïdor representa una nova aplicació que implica nous requeriments com ara una ràpida recuperació en cas de fallades, una utilització eficient dels recursos, i una millor selecció de les rutes així com la possibilitat de controlar-les.

A la tecnologia Ethernet el protocol d'Spanning Tree (STP) és el responsable d'establir la connectivitat entre els diferents segments. L'STP crea una topologia activa en forma d'arbre que evita que la xarxa tingui cicles i així permetre que les funcions bàsiques d'Ethernet es puguin executar sense problemes. Hi ha però alguns inconvenients d'utilitzar STP per transformar la topologia física en un arbre actiu. Una, la selecció d'un únic camí per connectar cada parell de nodes elimina la redundància que podria estar disponible a través d'enllaços addicionals. Dos, la comunicació a través d'aquest arbre només proporciona rutes de comunicació òptimes entre els nodes de la mateixa branca. Tres, l'STP necessita desenes de segons per recuperar-se d'un error mentre que els proveïdors només permeten un màxim de 50 ms. Tot i que aquest rendiment ha millorat amb la introducció del protocol de Rapid Spanning Tree (RSTP), aquest encara requereix desenes de segons per recuperar-se de la caiguda del node que fa d'arrel de l'arbre ja que l'RSTP pateix l'efecte de "count-to-infinity". I quatre, els camins que formen l'arbre són escollits només en base al cost de l'enllaç mentre que les xarxes de proveïdors requereixen més flexibilitat.

En aquesta tesi proposem una solució completa basada en extensions d'RSTP que aborda cadascun dels inconvenients esmentats. El nostre enfocament es basa en la comprensió dels fonaments del protocol per així identificar-ne les limitacions i dissenyar les extensions concretes que facin falta. En primer lloc, realitzem una caracterització detallada de l'RSTP proporcionant una descripció completa de la seva operació així com l'anàlisi del comportament del protocol en les situacions més comunes. L'estudi de l'efecte del "count-to-infinity" ens permet identificar les causes reals i una sèrie d'efectes inesperats que són concretament les raons per les quals la recuperació de la caiguda del node arrel és tant lenta.

El detallat coneixement del protocol ens permet dissenyar RSTP-Conf com una extensió d'RSTP que s'enfoca a solucionar el problema de la lenta recuperació. RSTP-Conf es basa en un mecanisme de confirmació simple però eficaç que evita el "count-to-infinity" detectant si el node arrel ha fallat realment. L'avaluació de l'RSTP-Conf es duu a terme mitjançant una simulació de xarxes. Els resultats mostren que la tècnica proposada redueix de manera eficient la recuperació a un màxim de dues vegades el retard de propagació a través del diàmetre de la xarxa.

La resta d'inconvenients són solucionats mitjançant l'ampliació de la topolo-

gia activa en diversos arbres. RSTP-SP desplega una topologia activa basada en la configuració d'un arbre per cada node. Això permet operar amb comunicacions que segueixen camins òptims i a més proporciona un increment de la utilització de recursos perquè amb molta probabilitat un enllaç estarà actiu en almenys un dels arbres. El principal repte d'aquesta estructura amb varis arbres és que requereix d'una acurada selecció dels propis arbres per tal de mantenir la propietat de simetria que Ethernet necessita. RSTP-SP estén RSTP i introdueix els canvis necessaris per tal de construir camins òptims i mantenir la simetria. El rendiment d'RSTP-SP es compara amb l'estàndard IEEE 802.1aq Shortest Path Bridging i els resultats mostren que l'RSTP-SP supera SPB en termes de temps de recuperació. Però, la sobrecàrrega de missatges introduïts per RSTP-SP és més gran que en el cas de SPB.

Publications derived from this work

The results obtained from this Thesis derived on the following publications:

⇒ E. Bonada, D. Sala, *RSTP-SP: Shortest Path Extensions to RSTP*. In Proceedings of the IEEE 13th Conference on High Performance Switching and Routing (HPSR'12), June 2012.

⇒ E. Bonada, D. Sala, *Characterizing the Convergence Time of RSTP*. In Proceedings of the Mosharaka International Conference on Communications and Signal Processing (MIC-CSP 2012), April 2012.

⇒ E. Bonada, D. Sala, *Building Ethernet Connectivity Services for Provider Networks (Poster)*. In Proceedings of the 28th International Conference on Computer Communications, IEEE Infocom 2009 Student Workshop, April 2009.

⇒ E. Bonada, D. Sala, *On the Theoretical Bounds of the Spanning Tree Algorithm*. In Proceedings of Jornadas Telecom I+D 2008, October 2008.

⇒ E. Bonada, D. Sala, *Implementation of a L2 Bridge in ns3 (Poster)*. In Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications (SIMUTools 2008), March 2008.

Contents

List of Figures

List of Tables

1.1 Extension of Ethernet into provider networks

In the last years there has been a large increase in the amount of communications generated or received by data consumers [1][2]. First, residential users have increased their data consumption due to applications that offer triple-play services (data, voice and video) or peer-to-peer communications. Also, the new internet based on social networking and multimedia information, together with the ubiquitous access to information thanks to mobile platforms, represents a huge increase of consumed data by these end users. And second, enterprise customers base their business on the networked platforms that seamlessly connect their sites. Networking has actually become a business necessity that improves efficiency and productivity providing instant remote communication, resource and information sharing or higher reliability with remote backups. In consequence, enterprise customers need to be connected at every place and every moment, and this results into a high traffic increase between the different physical locations of the enterprise.

In addition, this increase of traffic comes together with more strict requirements that customers demand. Video or voice communications impose strict requirements for a proper experience (minimum bit-rate, maximum latency, maximum jitter, etc.). In the case of enterprises, a reliable transmission of data is the most important aspect. Long outage situations where an employee cannot connect to the central server in the headquarters are not acceptable.

Traffic flows between residential end-users and multimedia servers or the communications between distant enterprise sites are transported beyond the private *Local Area Network* (LAN) through a network infrastructure formed by Service Providers. Figure 1.1 illustrates a scheme with the three different regions that comprise this network infrastructure: *Access Network* (Access), *Metropolitan Area Network* (MAN), and *Wide Area Network* (WAN). As opposed to the LANs, which are private and administered by the same owner (either residential or business), these three regions are typically owned and managed by the network providers and the connectivity through these networks is offered as a service for a recurring payment.

The customer LANs directly connect to the Access segment using some physical connection owned by the provider (the Access region is also known as last/first mile or local access loop). The Access infrastructure connects to the MAN. The MAN region refers to the network that covers a metropolitan area, usually span-

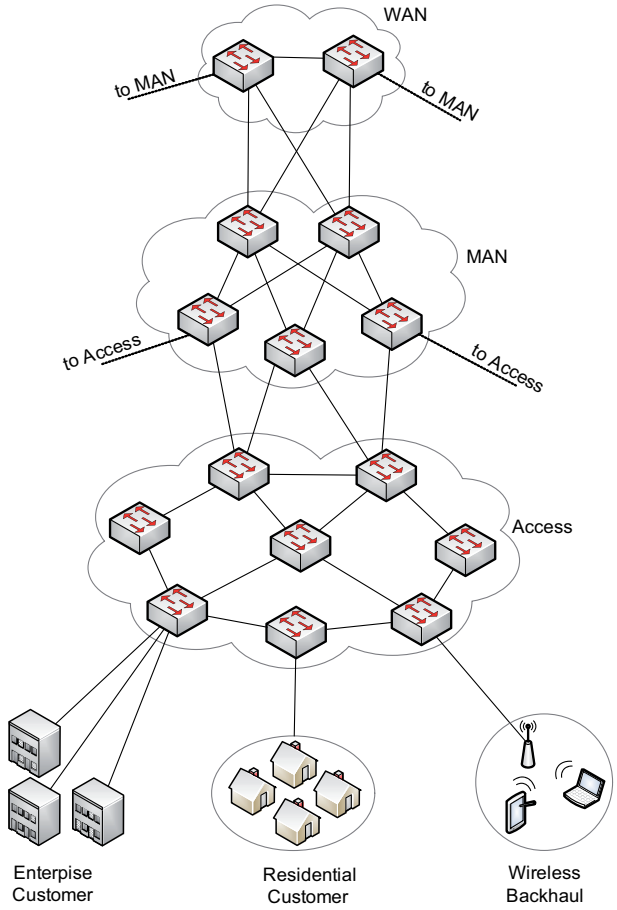


Figure 1.1: Network infrastructure of Service Providers

ning a city and its surrounding areas, and interconnects many entities including several Access regions, other MANs and a few WANs. A WAN refers to the network that covers a larger geographic area and interconnects several MANs.

Delivering connectivity services in the provider networks is substantially different than delivering them in the LAN. First, the increase in the geographical coverage (hundreds of meters to thousands of kilometers) leads to an increase on the scale of the network (tens of end-users to tens of thousands). Second, regarding the bandwidth, not only this must be larger in the provider environment, but the operating model changes from a dedicated use of resources in the LAN to the aggregated model with shared resources in the provider infrastructure. Third, the service scope in the LAN was reduced to a few applications for specific user needs, while in the provider networking numerous services are needed to address the needs of a broad range of customers. Fourth, the management in the LAN is relatively simple as it is composed of a few connections spread over a small area.

In the provider environment, managing thousands of remote users subscribed to different services increases complexity and requires of more sophisticated mechanisms. Additionally, strict *Service Level Agreements* (SLA) are negotiated in the provider environment because of mission-critical applications demanded by customers. And fifth, resiliency in LANs is not very critical because problems can be fixed quickly. However, this aspect becomes of high importance in the provider networks as critical unresolved failures highly impact the provider revenues and its long-term competitiveness. This is why providers require a maximum outage time of 50 milliseconds.

The initial attempt to provide data communication in provider networks was to use the existing telephony infrastructure deploying solutions based on *Time Division Multiplexing* (TDM) technologies and *Synchronous Optical NETworking* (SONET) rings. As data-based communications started to grow, packet-based technologies such as *X.25*, *Frame Relay*, or *Asynchronous Transfer Mode* (ATM) were also deployed. However, the TDM-based SONET solution became the dominant transport infrastructure.

As the amount of traffic keeps growing and the service requirements become more and more strict, providers are reconsidering the technology used in their deployments. These circumstances have positioned Ethernet data networks as one of the best candidates for replacing the existing ATM/SONET infrastructures [3][4]. Providers want to build their new data networks in plain Ethernet operating at layer 2 instead of replacing them by IP routers that contain more complexity and require higher expertise for management. The bandwidth flexibility of Ethernet (granularity of 1Mbps) in front of the limited steps of SONET technology (1.5M, 45M, 155M, 622M, 1.25G, 2.48G, 10G) is one of the main reasons of the intended replacement. Also, the manufacturing cost of Ethernet devices is very low compared to SONET. In addition, the continuous innovation of Ethernet technology implies a good strategic decision.

The adoption of Ethernet technology for provider networking actually represents a win-win situation for both the customer and the provider. The following list briefly describes the main points that position Ethernet as a good candidate.

- *Ubiquity.* One of the main reasons is that Ethernet is everywhere: already 95% of the network interfaces deployed use Ethernet technology [5] and 98% of all enterprise data traffic starts and ends their journey on Ethernet ports [6]. Since the traffic is generated in an Ethernet interface of the LAN, it makes sense to avoid any conversion or encapsulation to another technology that just introduces overhead.

- *Converged interface.* The convergence of all services into a single interface simplifies connectivity and equipment required in the customer premises, optimizes the bandwidth balancing across services and allows for the quick addition of new services without the need to install extra equipment. The converged interface also represents an advantage for providers because it allows for a simpler management and it is easier to deliver the services

optimally, which is turn results into an increase of revenue because of the better use of resources.

- *Bandwidth flexibility.* The Ethernet bandwidth scalability and high granularity allows a pay-as-you-use model where the customer only contracts the bandwidth it really needs. This results into an increase of revenue in the provider because many customers that had to upgrade to a much higher and expensive bandwidth might not do it.

- *Reduced Cost.* Ethernet devices are cheaper than other technologies (due to its mass production) and this allows for a reduction of the cost of the equipment in the customer premises. In addition, in-house expertise after several decades of Ethernet as the dominant LAN technology also avoids additional and costly training. The providers also observe a reduction in the cost of the service delivery (39% reduction in CAPEX and 44% in OPEX based on a MEF study [6]).

However, this provider environment definitely represents a new application for the Ethernet technology originally designed to operate in LANs. One of the main provider's concerns is the scalability of the technology. First, the provider networks extend to infrastructures of thousands of nodes while Ethernet was designed for LANs with few nodes. And second, the failure recovery mechanisms of Ethernet are too slow to operate in such new application. Another aspect that must be addressed is the support for *Quality of Service* (QoS). Since Ethernet is being considered as a platform for convergence where different services are delivered together, it should be able to support different applications with different requirements (such as latency sensitive video and voice applications). This is actually lacking because Ethernet was originally designed for LANs where a best-effort performance was enough.

Provider networks are managed infrastructures that require high availability and really need to provide the required performance. This means that the behavior of these networks must be configurable and efficiently administered. The *Operation, Administration and Maintenance* (OAM) capability of Ethernet needs to be improved as well. Moreover, in very large extensions there can be different interconnected networks from different providers. This implies that some traffic of a concrete provider may probably cross the network of another provider to get its destination. Since the new services provided now require strict quality constraints, providers that transmit others traffic need to ensure these requirements as well. The SLA that providers negotiate include all these performance restrictions in the contract, therefore the network must ensure the abilities to provide this desired performance and service control capabilities.

There are several approaches that address these issues in order to prepare Ethernet for provider networking [3]. Some proposals are based on using Ethernet over other technologies already deployed in the infrastructure: copper [7], HFC [8] or SONET[9]. Some others propose to use Ethernet as the base technology but focus on particular regions of the network: *Ethernet Passive Optical Networks* (EPON) [10] in the Access and *Resilient Packet Ring* (RPR) [11] in

the MAN/WAN. However, the natural solution is to evolve Ethernet Bridging with the necessary extensions so as to meet the provider networks requirements. The following section briefly describes the fundamentals of Ethernet Bridging and discusses its advantages and shortcomings as the base of the Ethernet evolution into provider networking.

1.2 Limitations of Ethernet Bridging

The original Ethernet was designed to operate in a shared medium segment within a small coverage area and with a few connected hosts. Ethernet Bridging provides the connectivity extension required to overcome the distance and performance limitations of a single network segment. This is achieved by means of Ethernet Bridges that are used as devices that interconnect the several Ethernet segments. The proper functioning of Ethernet networks relies on enabling the broadcast operation. Therefore, this property is maintained in all Ethernet evolutions: from the single shared segment where all nodes receive all messages, to the modern point-to-point Ethernet networks where bridges flood received frames to all ports. In summary, a bridged Ethernet network provides a broadcast domain with unlimited distance (a broadcast reaches all segments) but with improved performance by making the collision domain of each physical segment independent to the other segments in the network (a collision is isolated). Note that in point-to-point connections the collisions do not occur any more because it is not a shared medium, and hence the collision domain becomes secondary. Section 2.1 provides a more detailed description of the basics of the Ethernet technology.

In Ethernet networking the *Spanning Tree Protocol* (STP) [12] is responsible of establishing the connectivity of the different Ethernet segments in the interconnected network. This connectivity service is driven by the bridging principles based on deploying a plug-and-play architecture. A bridge starts with no configuration and sends all received data frames to all ports except the incoming. From this reception the bridge learns the port that leads to the source address, and subsequent frames sent to this address are directed to this port. All frames with unknown destination continue to be sent to all ports. The broadcast condition guarantees the reception of frames even if the bridge is not aware of the path to the frame destination. This broadcast could result in a continuous flooding if the network has loops (refer to section 2.1 for a more detailed example). Two important aspects of this operation need to be highlighted. One, the broadcast bridging operation only works in networks without loops; hence the spanning tree protocol is used to build a logical tree topology over any physical topology (note a tree has no loops and only one path between any two nodes). As shown in the example of figure 1.2, data communication is then only allowed through the tree links (thick links in figure). And two, the learning operation learns from the incoming frame assuming that the path coming from a node is the same than the path reaching such node. In other words, the path has to be symmetric. Note that the symmetry property comes natural in the tree topology as there is a unique path between each pair of nodes.

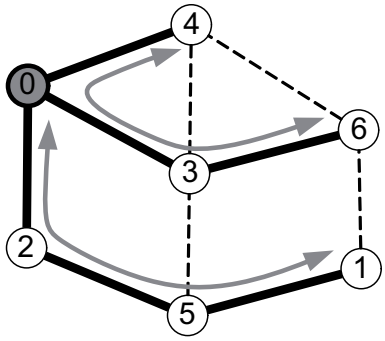


Figure 1.2: Data communications through the single tree

STP is a distance-vector protocol that constructs a shortest-path tree rooted at one of the nodes, the *Root*, which is arbitrarily selected. In the example of figure 1.2, the Root is $B0$ because it has the lower node identifier (note that $B0$ refers to the node with the identifier equal to 0). The convergence of the protocol toward the final tree structure is based on the exchange of information where each node disseminates its own topology view (who the Root is and at which distance). Every node hence receives the information from all neighbors and the best one (lowest Root, or same Root and smallest distance) is selected as the path to the Root. If this iteration is repeated every-time a neighbor notifies an update, the topology views of all nodes eventually converge to the shortest-path tree because all nodes select their best paths to the single Root (hence forming the shortest-path branches).

The original STP has been updated by the *Rapid Spanning Tree Protocol* (RSTP) [13], which is the the current technique specified in the standard. The main reason for such update is that STP takes tens of seconds to detect that the tree is completed because it is based on over-dimensioned timers. RSTP reduces the recovery times in most situations to the order of milliseconds because it introduces a reactive mechanism based on acknowledgements from the next node down the branch of the tree. In addition, RSTP nodes keep alternate paths to the Root through different neighbors and, in case of failure, the takeover to the new path is immediate. However, trusting in these alternate paths results into undesired consequences in the event of the Root failure because RSTP suffers count-to-infinity [14][15]. In this situation, some nodes distribute stale topology views (about the failed Root). Then, the messages about the new Root start looping chasing the messages about the old Root. This leads into an endless forwarding of protocol messages unless the behavior is stopped. As other distance-vector protocols, RSTP uses a hop counter in the protocol messages to detect them and mitigate the count-to-infinity. This however results into low convergence when the Root fails reaching recoveries of tens of seconds in large topologies. Chapter 3 provides a more detailed description of RSTP operation including the count-to-infinity behavior.

Regarding providers necessities, there are some implications of using RSTP

to prune the physical topology into an active tree. One, a single tree activates a single path to connect two peers and this eliminates all potential redundancy that might be available through extra links. Redundancy in provider networks is important in order to meet robustness requirements and increase global network capacity (a network with N nodes can only have N-1 active tree links that belong regardless the total number of links in the physical topology). Two, the communication over this single tree only provides good (optimal) communication paths between nodes in the same tree branch, but poor (sub-optimal) communication paths between nodes in different branches. In fact, the Root of the tree is the only node that observes shortest paths communication to all nodes as the path selection has been made with the criteria to be optimal to this 'elected' node. Three, the original STP takes too long (tens of seconds) to reconstruct the tree after the failure of a network element. This slow recovery was enough in small LANs without tight requirements but it remains too far from the 50ms maximum bound allowed in provider networks. RSTP only provides quick recoveries in the event of single failures (45ms in a network of 100 nodes); when the Root fails the convergence is larger because of the count-to-infinity (24 seconds in a network of 100 nodes). This time is still far from the 50ms that provider SLAs require as the top bound for network recovery times. And four, the branches that define the tree (and hence the data communication paths) are elected only based on the link cost. This needs to be enhanced as in provider networks there is a need to drive the path selection from additional perspectives: different nodes in the network, considering QoS or other SLA requirements.

The *Multiple Spanning Tree Protocol* (MSTP) [16] was introduced as a framework to manage traffic segregation and provide QoS in Ethernet networking. MSTP allows to assign different trees to different VLANs and introduces the possibility to apply load balancing and traffic engineering. In turn, MSTP represents an indirect improvement of the previous drawbacks because it allows the activation of a link either in one VLAN or another. Each one of the trees of MSTP is still managed as an RSTP instance, therefore MSTP inherits all advantages and shortcomings of RSTP in terms of recovery time. Although MSTP introduces a high flexibility in which trees are constructed, the main problem is the complex configuration because each port of each bridge needs to be individually set to operate in the corresponding VLANs. However, MSTP still deploys sub-optimal paths within each of the trees.

A simple way to deploy optimal paths with any pair of nodes is to configure as many trees as nodes. The idea is to extend the active topology into one tree rooted at each node of the network. Note that if each node uses its own tree to introduce its own data traffic, shortest-path communications are achieved between each node (Root) and the rest of nodes. The use of multiple trees also allows for an increase on number of active links because chances are that one link is used in one tree or another. This is what the last evolution of the family of the spanning tree protocols proposes. The 802.1aq *Shortest Path Bridging* (SPB) task group is currently defining an evolution that operates with shortest paths between any pair of nodes [17].

The main challenge of this shortest-path multiple-tree active topology is that

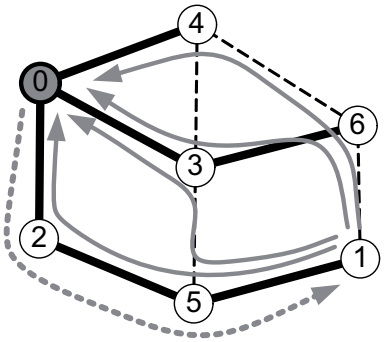


Figure 1.3: Potential lack of symmetry

the communication paths are selected as if they were unidirectional because a different tree is used in each direction. This requires a careful selection of such trees because the bridge learning function requires symmetrical paths. In the topology of figure 1.3, the tree rooted at the node $B0$ has configured the branch $B0$-$B2$-$B5$-$B1$. In the SPB framework, this implies that the messages from $B0$ to $B1$ follow this path. Because of symmetry reasons, the path from $B1$ to $B0$ should be $B1$-$B5$-$B2$-$B0$. However, the tree rooted at $B1$ could select any of the following shortest paths to $B0$: $B1$-$B5$-$B2$-$B0$, $B1$-$B6$-$B3$-$B0$, $B1$-$B6$-$B4$-$B0$ or $B1$-$B5$-$B3$-$B0$. Using a distributed protocol such as the spanning tree imposes a challenge because different trees might select different, and hence not symmetrical, branches. Section 5.1 elaborates on this issue.

This is why SPB introduces the link-state approach moving away from the distance-vector algorithms of the current spanning tree protocols. In general, link-state algorithms are more flexible in terms of potential extendibility but at expenses of an increase in computational complexity. Additionally, this change of path selection paradigm permits the avoidance of the RSTP recovery problems and introduces flexibility to make path selections based on more metrics rather than just the link cost. There is however a big concern on the impact that a change of path selection paradigm might have. Ethernet is a successful technology that has always based its evolution on extensions that kept its simplicity philosophy, a plug-n-play operation and a simple management. For instance, the spanning tree protocols are based on conceptually complex algorithms but very simple in operation. Introducing a link-state protocol to construct the trees involves implementing a more complex algorithm and this might impact Ethernet's success in the upcoming years.

1.3 Problem statement and solution approach

In this work we present a study that analyses in detail the previous shortcomings and describes the design and evaluation of the corresponding solutions to address them. We propose a complete solution that:

- Allows for quick recoveries, meeting provider networks requirements, in all failure situations
- Makes use of available redundancy so as to use all network links
- Provides optimal path communication between all pairs of nodes maintaining the symmetry requirement

Our approach is based on the extension of the current spanning tree protocols in order to keep the original distance-vector approach. The key point to identify the origin of the shortcomings is to first understand the fundamentals of the protocol operation. We therefore base our study on how RSTP constructs and maintains the single active tree. With the analysis of the single tree protocol we are able to understand the operation, characterize its performance and identify the limitations. This allows us to design *RSTP-Conf* as the necessary extensions to avoid the long recovery time in the scenarios where the Root of the tree fails causing RSTP to experience the count-to-infinity effect. The basis of the solution is the implementation of a simple yet effective confirmation mechanism that avoids the dissemination of messages about the failed Root, hence avoids the count-to-infinity.

The extensions that RSTP-Conf introduces only address the recovery time issues as the updated protocol is still based on the construction of a single active tree. We address the redundancy and path optimality shortcomings with the design of *RSTP-SP* (SP stands for Shortest-Path). RSTP-SP extends the RSTP protocol to be shortest-path for all communications. As in SPB, RSTP-SP configures one tree rooted at each node to operate with shortest-paths. The difference is that RSTP-SP keeps the distance-vector approach of RSTP instead of moving to a link-state framework. This way, we maintain the advantages of the original distance-vector approach only introducing the necessary extensions to RSTP so it can construct the shortest-path multiple trees.

1.4 Thesis contributions

To summarize, the contributions of this thesis are the following:

- Detailed characterization of the standardized RSTP providing a comprehensive operation description and a performance evaluation by means of network simulation. This study has allowed us to understand the propagation-based model of the spanning tree protocols and how this affects its performance. In addition, this has also allowed us to (1) understand the causes of the long recovery in RSTP when it suffers count-to-infinity and (2) identify unexpected consequences.
- Design and evaluation of RSTP-Conf as the necessary extensions to RSTP in order to resolve the long recovery problems in the scenarios where the Root of the tree fails (avoiding count-to-infinity). RSTP-Conf is based on a simple yet effective confirmation mechanism that avoids the use of the failed Root information before it is really confirmed by the same Root.

- Design and evaluation of RSTP-SP as the necessary extensions to RSTP to operate with optimal communication paths. The core of the RSTP is maintained and RSTP-SP only requires the updates of very concrete operations. RSTP-SP performance is also compared to the link-state solution in SPB.

- Implementation of the protocol modules (RSTP, RSTP-Conf, RSTP-SP and SPB) in the ns3 network simulator.

1.5 Methodology

We base our study on identifying the nature of the problem before proposing a solution. When analyzing distributed protocols, clearly understanding the propagation of the information is essential to identify the concrete operational disadvantages and propose the right solutions. This is why we first deeply analyze the behavior of the spanning tree protocol when constructing the tree. Understanding the propagation model has allowed as to (1) comprehend its performance in terms of convergence time and message overhead and (2) extend the propagation operation to deploy multiple trees.

An important limitation of Ethernet networks is the backwards compatibility with legacy technology. This imposes additional restrictions in the design of the proposed solutions such as keeping the same Ethernet frame format, maintaining the broadcast operation or preserving the plug-n-play property. Therefore, the proposed solution must fit into this Ethernet framework.

In order to achieve the objectives we follow a methodology based on analysis, design and validation of protocols by means of simulation. In the initial analysis phase we have performed an in-depth study of the problem reviewing the standardized protocols in order to understand the operation and identify benefits and disadvantages of the different protocols. The simulation platform has been very helpful in this initial phase, as it has been used to deeply study the details of the protocols operation. This has allowed us to identify the details that result in the global performance behavior observed. A key aspect of this detailed analysis has been the exhaustive observation of the simulator traces that provide the step-by-step protocol evolution. This methodology allows to verify the correct implementation of the protocol but also to detect particular cases that result in behaviors difficult to identify only studying the protocol operation from the theoretical perspective. For instance, identifying the causes and consequences of the count-to-infinity effect can only be done observing the details of each one of the messages transmitted.

We have also chosen the simulation as the evaluation platform because it provides enough information to evaluate the protocols under study. The simulation also allows us to perform sensitivity analysis to deeply study the protocols in order to identify the key elements and to characterize operation and behavior. To do so we have used a complete set of running scripts that allow configuring different simulation parameters. This also permits an automatic execution of several simulations varying particular details of the scenarios considered. In addition,

the use of these scripts permits massive repetitions of any (and many) particular input configuration.

It is very important to be rigorous on the implementation of a simulation platform. First, the design and implementation includes the use of modeling and simulation theory and software engineering practices. And second, the implementation is verified by means of exhaustive "sanity checks" performed at different points of the simulation that test the valid configuration of the simulation variables at that instant. This can be done in a simulation platform because, although we are actually testing a distributed protocol, the simulator actually has access to all the information.

1.6 Thesis outline

After introducing the framework of the problem and motivating its study, the remainder of the thesis is organized as follows.

- Chapter 2 includes general background information that extends the description already provided in the introduction. First, a review of the Ethernet technology explains its evolution from its origins with the single shared segment to the modern point-to-point bridged networks. And second, an overview of path-selection techniques presents and compares the two main paradigms: link-state and distance-vector.

- Chapter 3 provides the description of the standardized RSTP. This includes a comprehensive review of the protocol operation and a description of the behavior in the most common scenarios such as the initial construction of the tree as well as the critical case of the Root failure that leads to the count-to-infinity scenario.

- Chapter 4 contains the literature review of different proposals that address the same or similar problems. The description of the IEEE SPB proposal is then extended in chapter 5 because it directly compares to RSTP-SP and hence a further explanation is provided.

- Chapter 6 includes the characterization of the RSTP tree construction. It presents a comprehensive description of the propagation effect of the protocol operation. This has allowed us to clearly understand how the distributed information evolves, and hence it has been possible to derive a theoretical bound for the protocol convergence time. An evaluation of the RSTP performance by means of simulation is also included to confirm the propagation analysis.

- Chapter 7 presents the further study of the count-to-infinity effect with the identification of the unexpected side effects that delay even more the recovery of the Root failure. While the count-to-infinity problem is known we are not aware of a comprehensive description of the phenomenon and its effects as the one presented in this thesis. The design of RSTP-Conf as

an extension to avoid count-to-infinity effects is presented in this chapter as well as the performance evaluation comparing the original and extended protocols.

- Chapter 8 describes the RSTP-SP as the extensions of RSTP to deploy the multiple trees operating with optimal paths and presents a comparative evaluation of RSTP-SP and SPB.
- Finally, in chapter 9 the main conclusions of this thesis are outlined and some future work is presented.

2.1 Ethernet Bridging

Ethernet refers to the networking technology being used in LANs for the connection and communication of personal computers, printers, servers, and other devices. Ethernet technology comprises (1) the physical interface that interconnects the devices, (2) the frames being used for such action, and (3) the protocols employed to communicate between these devices (communication, signaling and control).

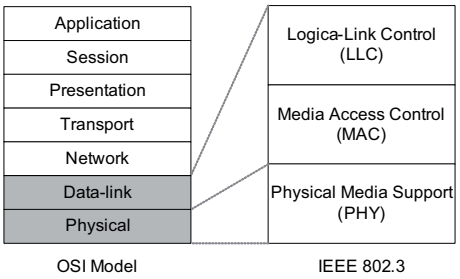
2.1.1 Basics of Ethernet

Ethernet [18] is defined by the IEEE 802.3 standard and its specification extents to both hardware and software aspects of the design (figure 2.1(a) shows the OSI layers correspondence). The Physical layer specifies the physical interface on the devices connected to the LAN (*Network Interface Card*, NIC) and the corresponding cabling. Each NIC has a unique static address, assigned by the manufacturer, referred as its MAC or Ethernet Address and based on a flat-addressing space of 6 bytes.

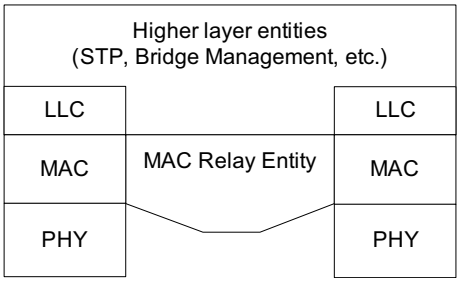
The OSI Data-Link layer is split into the *Media Access Control* (MAC) and *Logical-Link Control* (LLC) sub-layers. The MAC sub-layer defines the medium-independent capabilities built on top of the physical layer and is in charge of the data encapsulation of the Ethernet frame and the management of the medium access. The LLC sub-layer provides the interface between Ethernet and the upper layers and it focuses on the multiplexing and de-multiplexing of frames transmitted and received.

The IEEE 802.3 defines the basic Ethernet frame as shown in table 2.1. The maximum size of the frame is of 1526 bytes (corresponding to a total payload of 1500 bytes) and the minimum is 64. Larger payloads than the limit are split into different frames, and padding is added if total length is less than the minimum. The VLAN functionality [16] can be optionally added with the introduction of the VLAN identifier between the SA and the Type/Length field.

In terms of communication techniques between directly connected devices, Ethernet enables *half-duplex* transmission (transmitting in one direction at a time over a shared physical medium) as well as *full-duplex* transmission (simultaneously transmitting in both directions). The original Ethernet was design



(a) Ethernet device



(b) 802.1 Ethernet Bridge

Figure 2.1: Architectural reference of an Ethernet port and an Ethernet Bridge

Table 2.1: Ethernet frame format

Name	*Description*	*Bytes*
Preamble	Alternating pattern of ones and zeros that indicates to the receiver that a frame is coming. Also used for synchronization of bit-level parsing.	7
Start-of-frame delimiter	Indicates to the receiver the start of the new frame.	1
DA	Identifies the MAC address of the device to receive the frame.	6
SA	Identifies the MAC address of the sending device.	6
Length/ Type	Indicates the number of bytes in the payload or the type of frame.	2
Data Payload	Actual data being carried by the Ethernet frame.	46-1500
Frame Check Sequence	Cyclic redundancy check value that is used to validate the frame and ensure it has not been corrupted.	4

supporting the first half-duplex shared medium option, although nowadays most Ethernet connections are point-to-point and operate in full-duplex.

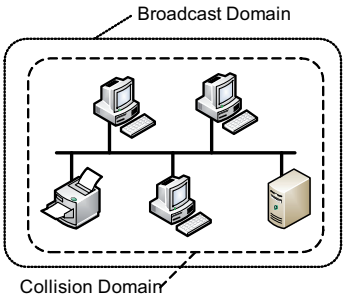
2.1.2 Original Ethernet

Ethernet was originally designed to work in small LANs using a simple bus topology where all nodes were connected to the same segment (see figure 2.2(a)). When for example a computer wants to send a printing request message to the printer in the segment, the Ethernet frame generated at the computer is encoded with the MAC address of the computer interface as SA and the MAC address of the printer interface as DA. The frame is transmitted using the protocol *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) [19] because the devices are connected to a shared medium. In the bus topology, this frame with the printing request is actually received by all devices, but only the MAC sub-layer of the printer interface processes it, and forwards it to its LLC sub-layer, because the frame carries its own MAC address in the DA field.

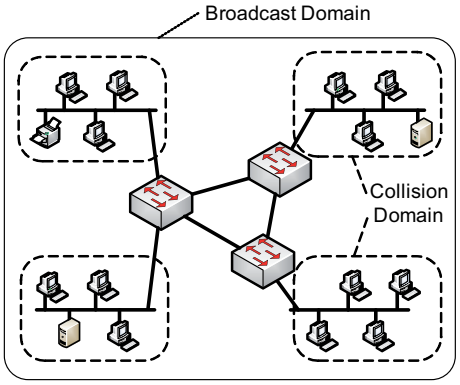
The CSMA/CD is a medium access protocol designed for the original Ethernet. A transmitting device listens to the medium (it actually senses the signal carrier) and, if it is available, it transmits the frame immediately. If it detects that someone else is transmitting, it retries when it is available again. It might happen that several transmitters detect an available channel and start transmitting simultaneously. In this case a collision occurs and, when the transmitting nodes detect it, they stop the transmission and retry the sensing of the medium after a random time. A *binary exponential backoff* algorithm computes this amount of time. If a collision reoccurs, a new backoff time is computed for each device, exponentially reducing the probability of another collision. In this fashion, the CSMA/CD reduces collisions and improves transmission efficiency by 80% compared to other medium access protocols at that time (ALOHA [20]).

Note that Ethernet technology has a broadcast nature on its origins because all traffic in the segment is actually received by all devices (although only processed by the actual destination). More technically, the logical area where nodes can reach each other by broadcast at the Data-Link layer spans the entire segment. This are is known as the *broadcast domain*. This is not efficient in terms of medium capacity, but it is very robust because the reception at destination is guaranteed. In addition, it provides total transparency for end nodes as they do not need to know where the destinations are located; the computer just sends the request and the network ensures it is received.

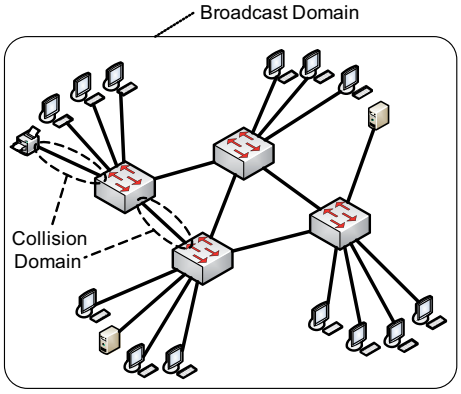
The disadvantage of a shared medium like this one is that concurrent transmissions create collisions that require retransmission of the frames. Although CSMA/CD reduced the amount of collisions, the broadcast nature of Ethernet still presents an added difficulty for a shared medium. In this case, the logical area where node transmission can collide with one another if sent in a shared medium also spans the entire segment (area is known as the *collision domain*). In order to ensure the collision detection, the frame transmission time must be larger than the propagation distance between the two further devices (so the collision is detected before the sender finishes transmitting). For the given minimum frame size (64 bytes), this results in a maximum segment length of 2.5km for 10Mbps, 250m for 100Mbps, 25m for 1Gbps, and so on. This limitation for high speed networks is clearly one of the drawbacks of the original shared medium.



(a) Ethernet segment as a bus topology



(b) Ethernet Bridging interconnecting several bus segments



(c) Ethernet bridging with point-to-point interconnections

Figure 2.2: Evolution of the Ethernet network topologies

2.1.3 Ethernet Bridges

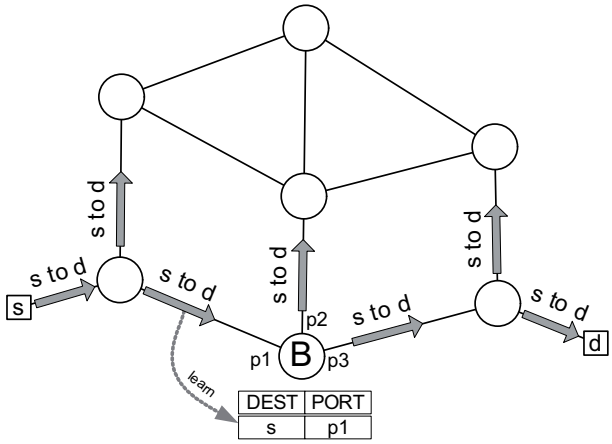
Interconnecting a large number of nodes through the same segment is not feasible as they all would share the same collision domain and hence the data transmission would not be efficient. *Ethernet Bridging* [21] came up in order to solve this problem and provide the connectivity extension required to overcome the distance limitations of a single network segment (see figure 2.2(b)).

An *Ethernet Bridge* is an interconnection device that operates at the Data-Link layer and that is uses to join two or more LAN segments to construct a larger LAN. Figure 2.1(b) shows the layered architecture of an Ethernet bridge connecting two Ethernet segments. Observe how a bridge is actually composed of several ports with Physical and MAC/LLC layers together with (1) a MAC Relay entity that forwards data frames between such ports, and (2) upper bridge clients that include STP and Bridge Management entities.

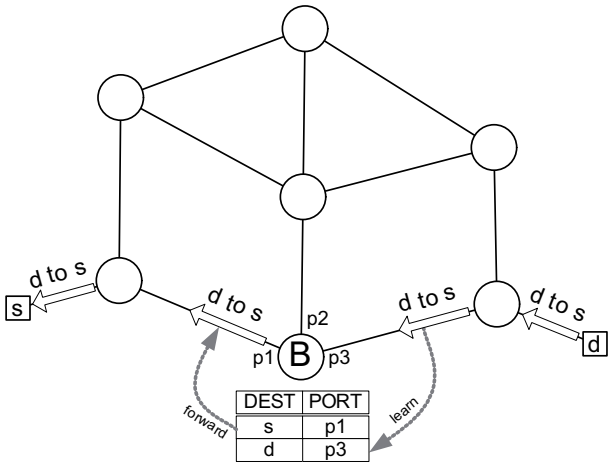
The main function of the bridge is to forward received frames between incoming and outgoing ports. The information that these bridges maintain to correctly forward a message to the next hop is stored in the form of *forwarding tables*. The entries in these tables are composed by the correspondence between a DA and the outgoing port that directs to such destination. This way, the message is forwarded hop by hop from source to destination if each node in the path selects the right outgoing port.

One of the reasons why Ethernet has become such successful is because it is a plug-and-play technology and it does not need any initial configuration to run. This means that the nodes do not need to configure the forwarding table in advance. At the beginning all bridges have empty forwarding tables. When a bridge receives a frame that must be forwarded to an unknown destination, it forwards it to all the outgoing ports except the incoming. As shown in figure 2.3(a), a frame originated at host s and destined to host d is broadcasted (flooded to all ports except the incoming) by the bridge and by all bridges in the network. This unknown frame is transmitted the same as a true broadcast frame (a frame with destination address equal to broadcast). If each bridge repeats this broadcasting operation, the frame reaches all nodes ensuring the correct reception of the frame at destination (host d in the figure).

However, the flooding of all frames results into a not efficient data communication framework. This is why Bridging introduces the automatic construction of the forwarding tables as the network operates: the *learning* functionality. The idea is to avoid the flooding when the forwarding tables have an entry that indicates how to forward the received frame, otherwise it is still flooded. The learning function assumes that the path that a frame came from a source node will be the same path to reach this same node as destination, this is, assumes that the communication is bidirectional. The learning function proceeds as follows and is shown in figure 2.3(a). When a bridge B receives a frame from the end-station s in port $p1$, it realizes that s can be reached through port $p1$. Then, the bridge adds the corresponding entry in the forwarding table linking the host s with the outgoing port $p1$. When later on the same bridge receives a frame that is destined to s (figure 2.3(b)), it forwards it to port $p1$ because the forwarding table



(a) Broadcast forwarding of a frame to a destination still unknown at the forwarding table and learning of the source address (bridge B learns the direction to reach s is port p when it receives the frame that s sends to d)



(b) Unicast forwarding of a frame and with a known destination and learning of the source address (bridge B learns d in port q)

Figure 2.3: Examples of the bridge forwarding and bridge leaning operations

indicates so. With this process the direction of all destinations is learned and the use of flooding is minimized.

Since the bridges build the tables backwards by learning the origin of the data frames that arrive to the node, one of the main requirements is that the path a frame traverses to get from s to d is the same that another frame uses to travel from d to s. Therefore the paths must be symmetrical otherwise the learning functionality would not match the right ports and the forwarding of frames would not be correct. In relation to this, bridges consider that each one of

the entries in the forwarding tables has a lifetime (concept of *aging*). The default value of the expiration is 300 seconds (5 minutes): a shorter value results in more flooded frames because a learnt path expires before it is refreshed; a larger value results in potentially keeping wrong information in the tables for too long.

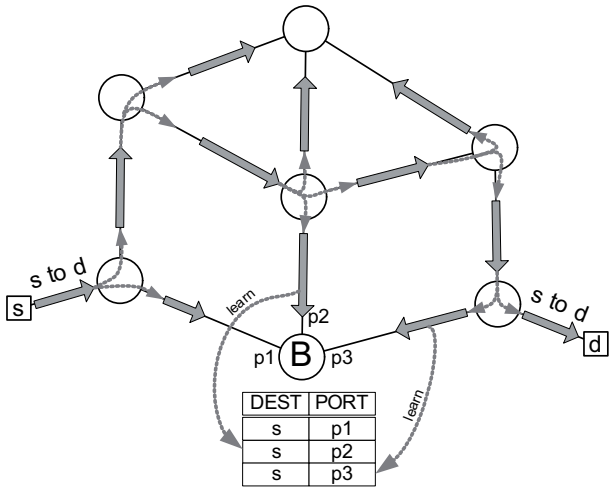
With the forwarding and learning functionalities, bridges regulate the traffic between the segments applying a filtering of traversing Ethernet frames. This filtering does not affect the original broadcast property of Ethernet. It is just a mechanism to avoid the broadcast of frames if it is not needed because the bridges know where to send them. The broadcast nature is kept with the addition of bridges and the broadcast domain still spans the entire network. This keeps the end node transparency as the computer just sends the request and the bridges in the network either broadcast it or direct it to the correct segment. Another advantage of Ethernet bridging is the reduction of the collision domain into each single segment. This is accomplished because bridges isolate the collisions that occur in a particular segment and hence are not noticed in other ports.

Modern Ethernet Bridged networks are deployed using point-to-point links that operate in full-duplex mode (see figure 2.2(c)). The operation of bridges connecting point-to-point links remains the same and they still learn addresses to filter frames and broadcast when needed. This implies that the broadcast domain still spans the entire network and keeps the end-node transparency. Another consequence of the removal of shared segments is that the collisions do not happen any more (actually, these might only occur within each point-to-point link if half-duplex transmission is used).

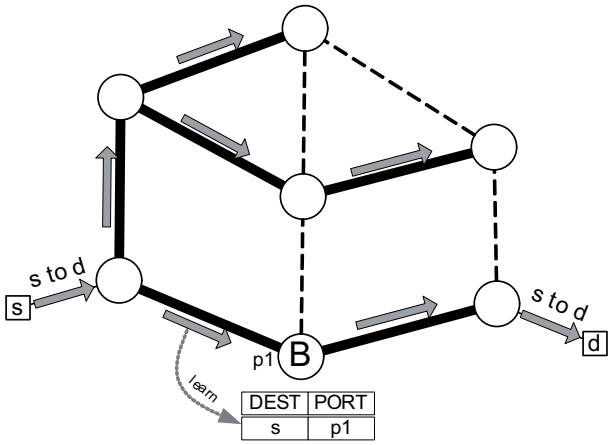
2.1.4 Loop avoidance

As already described, Ethernet has been a broadcast technology from its origins and this property is actually one of the pillars that has been kept in all evolutions. Although the learning functionality minimizes the amount of flooded frames, the broadcast provision must be guaranteed because it ensures the delivery of a frame to the destination even if the network configuration (forwarding tables) does not indicate it. However, the broadcast forwarding becomes a problem if the network has loops.

In broadcasting situations bridges receive a frame in one port and forward it to the rest. If all bridges do the same and loops exist in the network, the number of transmitted frames grows rapidly. This situation leads to a heavy congestion of the resources until saturation and a complete network breakdown. The diagram in figure 2.3(a) showed an example where a frame from s to d is flooded. The diagram in figure 2.4(a) shows the following sequence where bridges continue to flood the frame from host s that is endlessly forwarded. This results into a quick increase of congestion and multiple receptions of the same frame at the destination host d. This situation is called a *broadcast storm*. A broadcast storm also affects the learning functionality as a bridge may receive the same frame in different ports and the learning function would continuously change the learnt entry creating forwarding table instabilities. In the example diagrams, bridge B learns the frame first in port $p1$, then in port $p2$, then in port $p3$, and so on. This



(a) Forwarding table instabilities are caused by broadcast storms.



(b) Pruning the topology into an active tree ensures the proper bridging operation

Figure 2.4: Broadcast storm effects and how the active tree topology avoids it

loop-less constrain cannot be a requirement of the physical topology because the high availability in networks is usually achieved adding redundancy. Note that loops are created with the addition of links between nodes because two points are connected through more than one path.

Ethernet Bridging uses the STP to construct an active logical tree on top of the physical topology in order to eliminate potential loops. As shown in figure 2.4(b), the active topology constructed by STP is a shortest-path tree rooted at one of the nodes, the Root, which is arbitrarily selected. Shortest path branches that connect the Root to each one of the other nodes compose this tree. If only

the links that belong to the tree are used for data communication (the rest are blocked) flooding of frames is not a problem anymore and the broadcast storm behavior is avoided. In addition, pruning the redundant physical topology into a tree ensures a single active path between any pair of nodes, and hence the symmetry property required by bridge learning is also guaranteed.

2.1.5 IEEE 802.1 Spanning Tree Protocols

This section contains a summary of the different standardized spanning tree protocols: STP, RSTP and MSTP.

Spanning Tree Protocol (STP) The STP is the original loop-avoidance technique used in Ethernet bridged networks [12] and it is based on Perlman's algorithm [22]. The STP is a robust and self-configuring protocol aimed at creating the single tree-shaped active topology in the initial LANs. Therefore it is not optimized to provide neither (1) a high utilization of resources (not all links are used because it configures a single tree) nor (2) a quick recovery after a failure. The main reason why STP experiences a slow recovery is because its operation is based on the use of timers. Basically, timers start when there is a topology change and the protocol assumes that all the recovery operation is done when the timer expires. For this reason the timer values must account for the worst case and hence are configured with very large values (order of seconds). This results into unnecessary delays when recovering from failures, which span to several tens of seconds.

Rapid Spanning Tree Protocol (RSTP) The main objective of the RSTP [13] is to decrease the configuration time of the tree topology and hence reduce the outage while a recovery situation. The RSTP must be seen more as an evolution of the STP rather than a revolution. The base of the topology construction in both protocols is the Perlman's algorithm [22] and most of the improvements introduced by RSTP are small changes in the protocol operation. The RSTP introduces three main modifications:

- Ability to quickly reconfigure the active topology using alternate paths to the Root [23]. This reduces the convergence time because a node can independently switch over its path to the Root and other nodes do not even notice the failure.

- Use of independent refreshment messages that allow for earlier failure detection [24].

- The main enhancement aims at removing the timer dependence of STP. Instead of waiting for timeouts to start communicating, RSTP uses a proactive mechanism based on confirmation messages as the topology information is disseminated [25] [26].

Performance analysis [27][28][29][30] show that the 50ms bound is only achieved in topologies of only a few nodes. In addition, since RSTP is a distance-vector algorithm, it suffers the count-to-infinity problem when recovering from the Root failure [14]. The count-to-infinity experienced by RSTP results into large recovery times of the order of tens of seconds.

Refer to chapter 3 for a review of the RSTP protocol including a detailed description of its operation as well as a behavior explanation in common scenarios, such as the initial construction of the tree or failure recoveries.

Multiple Spanning Tree Protocol (MSTP) With the introduction of VLANs into the IEEE 802 framework, the first solution is to use the same single spanning in all VLANs. However, this does not solve the problems of the unused links or the slow network recoveries.

The first approach to multiple instances is made by Cisco with the *Per-VLAN Spanning Tree* (PVST) [31]. This solution allows the use of a distinct tree instance for each one of the active VLANs in the network. However, the main disadvantage of this solution is that a high number of tree instances must be computed and maintained by the bridges.

The MSTP (now included in the VLANs IEEE standard [16]) provides a framework for the use of various tree instances with different VLANs (many VLANs can share the same tree instance). It also introduces the concept of *region*. A region is the set of nodes that have the same configuration. Each one of the nodes in the topology is configured by the network administrator with a set of parameters (name, id, and list of active VLANs). All neighbor nodes that share the same information are within the same region and each region configures its own spanning tree.

One advantage of MSTP is that with the use of multiple trees all links of the network can be used. In addition, with the use of regions MSTP achieves a segmentation of the network and isolates potential failures that only apply to a single region. However, each one of the tree instances still operates as a single RSTP tree and hence neither optimal paths nor sub-50ms convergence is achieved. Moreover, the inclusion of different trees adds complexity and management difficulties that could derive to configuration problems and network misconfigurations.

2.2 Path selection

The main objective of communication networks is to provide connectivity between different nodes so they can exchange data messages. Since these nodes might not be directly connected, the intermediate elements forward the messages between the communicating peers. The sequence of traversed nodes is called the path, and the summation of the costs of all traversed links is the path cost. The set of all paths used for data communication is referred as the active topology, while all existing nodes and links compose the physical topology.

There are different ways to construct these paths [32]. One option is to use a *centralized* method where the physical topology is assumed known by a central

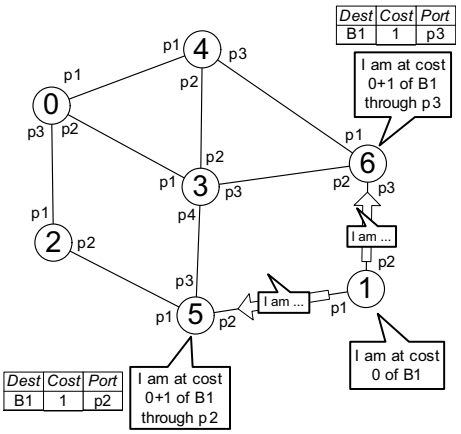
control point that applies any common graph algorithm to locally calculate the paths. These are then disseminated to the rest of nodes so they are also aware of the calculations. Another option is to apply a *distributed* mechanism where network nodes exchange topological information and independently compute the paths.

The distributed technique is applied in most networking technologies by implementing protocols that exchange information and compute the paths. There are two main approaches to identify a communication path in a network [33]: *distance-vector* and *link-state*. The objective of both is to select shortest (although not necessarily) paths to a given destination. Finding the shortest path from all nodes to a single destination results into constructing a shortest-path tree rooted at the destination and where the branches are shortest paths.

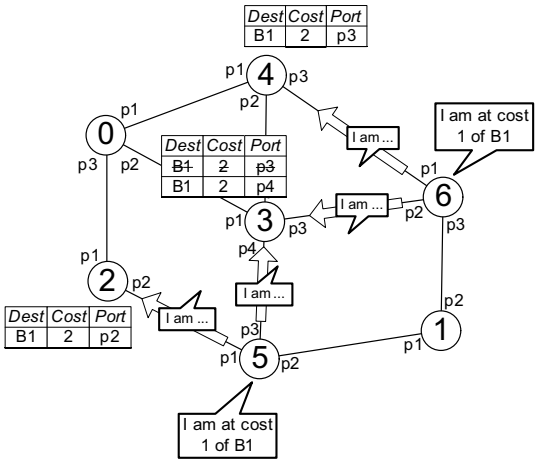
2.2.1 Fundamentals of distance-vector protocols

Common distributed distance-vector (DV) protocols are used to construct the paths and populate the forwarding tables so the data messages are correctly routed. In order to avoid confusion, note that in this section we refer to the distributed DV algorithms and not to the centralized versions like the original centralized Bellman-Ford algorithm [34].

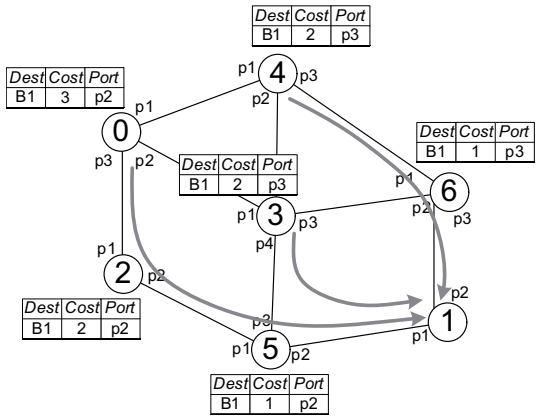
The essence of the DV protocols is the sharing of topological information, basically the distance to a particular destination, between direct neighbors. The processing of messages between neighbors, and the consequent updates, ensures that all nodes eventually realize the shortest distance to each destination. Figure 2.5 shows the steps to construct the paths to destination $B1$. This node starts the procedure by sending a message to its neighbors telling that it is located at cost 0 of itself (in figure 2.5(a) this is represented by the message *"I am at cost 0 of B1"*). The neighbors ($B5$ and $B6$) receive such messages and realize that they are located at cost 1 $(0+1)$ from $B1$. In addition, they also know in which direction $B1$ can be reached as they assume that the port where the *"I am..."* message is received leads to $B1$. Observe in the forwarding tables in 2.5(a) that $B6$ stores that $B1$ can be reached at cost 1 through port $p3$; similarly, $B5$ stores that $B1$ can be reached at cost 1 through port $p2$. Since $B5$ and $B6$ have updated their information on how to reach $B1$, they disseminate the news to the rest of neighbors. In this case, $B5$ sends a message telling that it is located at cost 1 of $B1$ (this is represented by the message *"I am at cost 1 of B1"* in 2.5(b)). $B6$ also sends a similar message announcing it is at cost 1 of $B1$. $B2$-$B3$-$B4$ receive these messages and also update their information about how to reach the destination $B1$. $B2$ and $B4$ set a cost of 2 and the outgoing port as $p2$ and $p3$, respectively. $B3$ has to make an additional decision because it receives both messages from $B6$ and $B5$. Assuming $B6$'s is received first, $B3$ assumes a cost of 2 through its port $p3$. $B3$ later receives $B5$'s message, which also includes a cost of 1 and hence would locate $B3$ at a cost of 2. $B3$ needs to decide which path is selected to reach $B1$: through $p4$ ($B5$) or through $p3$ ($B6$). A tie-breaking condition is decided arbitrarily and different policies can be used. However, the common decision is



(a) B1 starts notifying about itself



(b) Neighbors of B1 update and continue to notify



(c) Final configuration

Figure 2.5: Population of the forwarding tables using a common distance-vector protocol

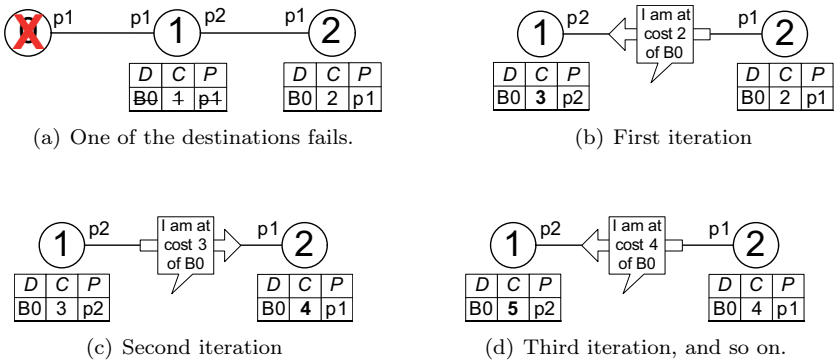


Figure 2.6: Common distance-Vector protocols experience count-to-infinity when one of the destinations fails

to select the path that has an immediate next hop with a lower identifier. In this case, $B3$ selects $p4$ ($B5$) as the port that reaches $B1$.

A node selects a new path and disseminates to the rest of neighbors every time it receives information that updates its distance to $B1$.This leads to a situation when the information about the distance to $B1$ is received in all network nodes. Figure 2.5(c) shows the network state when the entire paths to $B1$ have been already configured in each node. Observe how the active topology including the paths to reach the destination $B1$ is a shortest-path tree rooted at this destination.

The previous example describes the dissemination of distance information from only one destination. Note that in a more realistic example each destination sends its own messages. Therefore, all nodes receive messages from all destinations so they can build the forwarding tables to reach every node. In practice, the created paths represent the construction of one shortest-path tree that has a root at a different destination. In other words, each node (as potential destination) floods its own messages and triggers the construction of its own tree. The constructed paths represent the branches of the destination-rooted shortest-path trees.

The main drawback of the DV protocols is that they experience the count-to-infinity effect in some situations. Figure 2.6 shows a simple example where all nodes have their tables configured telling how to reach node $B0$. This node fails and $B1$ realizes it has no connection to $B0$ any more (figure 2.6(a)). When $B2$ tells $B1$ that it is located at distance 2 of $B0$ (figure 2.6(b)), $B1$ accepts the information and now believes it is located at distance 3 of $B0$. $B1$ now announces the new information to $B2$. This accepts the received information from $B1$ (it actually updates last received) and now $B2$ believes it is located at cost 4 of $B0$ (figure 2.6(c)). This exchange of messages continues and the cost increases with each iteration(figure 2.6(d)). The counting of the cost to infinity is not stopped unless the protocol applies a technique to detect the behavior.

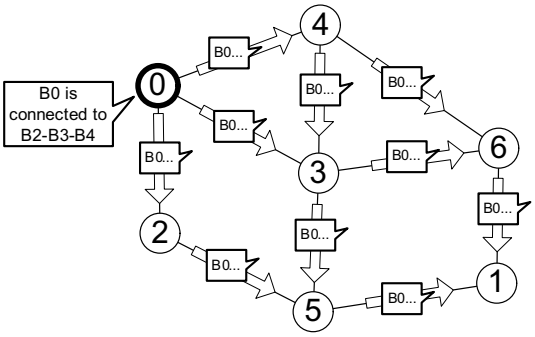
2.2.2 Fundamentals of link-state protocols

As its name indicates, the link-state (LS) approach is based on the dissemination of the link states: whether a network link is available for communication or not. This way, all network nodes construct a database of the state of all network links. In other words, each node is aware of the entire physical topology. With this information available in all devices, it is just a matter of locally executing the same path selection algorithm and constructing the active topology. Observe how this approach extends the centralized computation of paths into a distributed framework by first disseminating the entire physical topology to all nodes. In consequence, the link-state approach inherits the flexibility of the centralized protocols.

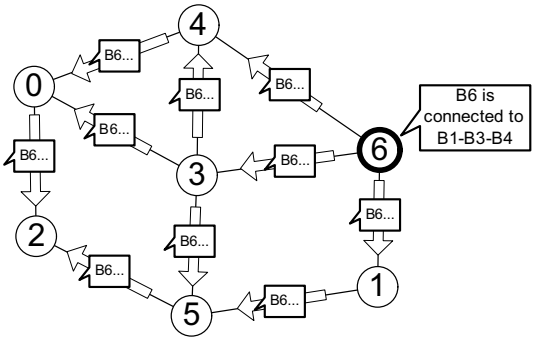
In order to deliver the physical topology to each node, link-state protocols require a topology acquisition mechanism to disseminate the state of the links to all network devices. This mechanism is based on flooding the state of each link in each node. In the example of figure 2.7(a), $B0$ tells to its immediate neighbors that $B0$ is connected to $B2$-$B3$-$B4$. Note that, for example, $B2$ already knows that it is connected to $B0$, but it does not know that $B0$ is also connected to $B3$ and $B4$. The information about $B0$'s connectivity is disseminated to the entire network and hence all nodes learn $B0$'s connections. This procedure occurs in parallel with all nodes (like $B6$ in figure 2.7(b)), and therefore when the dissemination is finished all nodes share a database of link states like the one shown in 2.7(c). Note that this list of connections is one of the possible representations of the physical topology (in graph theory notation this is known as an adjacency list [34]).

Once a node has the complete physical topology, it locally runs a path selection algorithm. The algorithm used to compute the paths is arbitrary but it must be the same in all nodes (all nodes compute the same paths as long as the algorithm is the same). The most common selection is to compute optimal paths using an all-pairs shortest-paths algorithm. The best option is to select Dijkstra [35] because is the centralized shortest-path algorithm that that runs faster. Note that the basic Dijkstra only computes the shortest paths from a particular node to all the rest. Therefore, in order to compute the paths between all pairs, Dijsktra is locally executed as many times as nodes in the network.

The procedure that link-state protocols follow can be graphically seen as a distributed construction of a puzzle. A puzzle piece represents one node and its connections. Therefore there are as many puzzle pieces as nodes in the network. The objective is to share copies of the pieces with all network elements so each node can independently construct the whole puzzle, which represents the entire physical topology. Each node floods its own puzzle piece, so all nodes eventually receive all of them. When each node completes the puzzle, the path selection algorithm is applied.

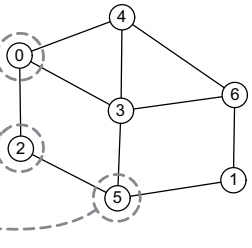


(a) Node B0 disseminates its own connections



(b) Node B6 disseminates its own connections

Node	Connections
B0	B2-B3-B4
B1	B5-B6
B2	B0-B5
B3	B0-B4-B5-B6
B4	B0-B3-B6
B5	B0-B2-B3
B6	B1-B3-B4



(c) Physical topology constructed in each node

Figure 2.7: Population of the forwarding tables using a common link-state protocol

2.2.3 Performance overview

The main difference between the two paradigms is how they operate to construct the trees. In the distance-vector algorithms, the nodes realize how to reach a particular destination comparing information from the immediate neighbors (the best path announced is selected and disseminated so other neighbors can also select it). Differently, in link-state algorithms nodes first use a topology discovery mechanism to acquire the entire physical topology. Then, each node individually, and locally, calculates the paths. The differences in operation between the distance-vector and the link-state result in performance differences as well [33]. These are outlined following:

- *Convergence Time.* LS generally outperforms DV in the time required to create the paths because DV protocols suffers count-to-infinity in some situations. In LS the only limitation is the propagation and processing delays. However, the performance can become quite even when avoiding count-to-infinity situations.

- *Computation.* LS are more demanding than DV. The local computation of the entire paths depends on the complexity of the protocol used, usually Dijkstra. On the other hand, DV protocols do not need to make any complex calculation because they only compare received messages with previously received information.

- *Bandwidth.* Since DV rely on message dissemination as the only way to iterate the protocol, bandwidth required is higher than in LS protocols.

- *Extensibility.* It is easier to add functionalities in LS protocols because the paths are actually computed using a local algorithm. DVs operate totally distributed, what might result into a limitation in terms of flexibility.

Depending on the network technology one approach might be preferred than the other. For example, in the case of IP-routing the link-state protocols like IS-IS [36] have beaten the distance-vector ones like RIP [37]. The reason is that a router is an intelligent device that can route packets based on different metrics and policies. This requires a flexible mechanism such as the link-state approach.

On the other hand, Ethernet Bridging relies on the active shortest-path tree. The simplicity of Ethernet bridges has derived to the selection of distance-vector algorithms to construct this single tree. This is why the original STP, and its evolution RSTP, is a distributed distance-vector protocol that constructs the single shortest-path tree rooted at the Root. Since the objective is to construct a single tree the most effective solution seems to be the use of a distance-vector algorithm that constructs only one shortest-path tree to reach only one destination (this destination would be the Root of the tree). On the contrary, using a link-state algorithm to construct only one tree seems inadequate because sharing the entire network topology for constructing just one tree results inefficient.

This chapter provides a detailed description of RSTP as the current standardized technique used to construct the single tree in Ethernet bridged networks [13]. Understanding RSTP is a key point to comprehend the limitations of the standard approach and at the same time to set the basis of the proposed extensions.

RSTP must be seen as an evolution of the original STP [12]. Both protocols configure a single-rooted tree and their principles come from the distance-vector algorithm defined by Perlman [22]. The main differences between STP and RSTP are on the direction of improving the convergence time of the protocol by introducing small operation changes. The current chapter describes the RSTP operation as it is the currently standardized technique. However, the protocol fundamentals are also applicable to STP.

This chapter is organized as follows. First, section 3.1 describes the elements and variables that compose the state of the nodes running RSTP. The details of the protocol operation in response to different events is detailed in the form of pseudo-code in section 3.2. Sections 3.3 and 3.4 review the RSTP behavior in two common scenarios (the initial tree configuration and a failure recovery, respectively) in order to comprehend the protocol operation. Finally, section 3.5 reviews the consequences of the Root failure that causes the to count-to-infinity.

3.1 Protocol elements

RSTP creates the shortest-path active tree electing one of the nodes as Root of the tree and configuring shortest-path branches connecting all other nodes. Each node has a unique identifier, the *BridgeID*, that is used to determine several aspects of the tree shape.The node with the lowest BridgeID is elected as Root of the tree (in the example of figure 3.1 the Root node is $B0$ because it has the lowest identifier (0). The BridgeID is used in tie-breaking mechanisms to decide among two equal shortest-path branches.

3.1.1 Distributed port activation

RSTP is a distributed protocol where each node executes the same operation in order to locally decide their position within the tree. The combination of the individual decisions results in creating a spanning tree from the global network perspective. In terms of tree/non-tree links, this means that each node decides whether its own ports are active or not, and only those links with two active ports

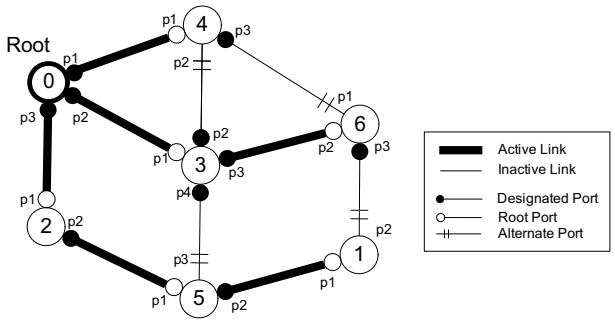


Figure 3.1: Tree rooted at bridge B0 with active/inactive links and port roles

at both sides are active from the data forwarding perspective. RSTP controls whether this property with the port variable *state*: a port is in *Forwarding* state if it is active and transmits data frames; a port is in *Discarding* state if it is inactive and blocks the data transmission. Note that the protocol operation is based on setting the ports, so there is no link information per se.

The bridge operation uses the state of the ports in conjunction with the forwarding table. The data transmission effectively happens in the active tree as follows. A bridge forwards a data frame to the port learnt in the forwarding table if the destination address is known. On the other hand, if the destination address is not known the bridge broadcasts the frame to all ports except the incoming port. However, only those ports in Forwarding state actually forward and receive the data frame while the blocked port discards all of them. Since the blocked ports do not accept data frames, these ports can never learn any path to any source address and hence a blocked port is never added in the bridge forwarding table by the learning operation. Therefore, it is guaranteed that data transmission will never be directed to blocked ports in any case (regardless if the destination address is known or not).

3.1.2 Port roles

RSTP decides whether a port is active or inactive determining the *role* of the port. The role defines the responsibility of this port in the link that connects to the neighbor. There are three possible port roles:

- The *Root port* is the port that connects the bridge upwards the Root (in the tree of figure 3.1, $p1$ of $B5$ is the Root port). The Root port is active because it is actually the path that connects the bridge to the branch that leads to the Root.

- The *Designated port* is the port that connects the bridge downwards the leaves ($p2$ of $B5$). The Designated ports are also active because they are also part of a branch and provide connectivity to child neighbors.

- The *Alternate port* is the port that provides an extra connection to the Root (a part from the Root port) but it is temporarily blocked because the redundancy would lead to forwarding loops ($p3$ of $B5$). Note that the Alternate ports are the locations where loops are broken.

Note that the active links of the tree are those that have a Root port on one side and a Designated port on the other. Since both ports have active roles, the transmission of data in these links is complete (links drawn with thick lines). Differently, a blocked link that is not part of the tree is defined as having a Designated port on one side and an Alternate port on the other. In this case, the Designated port is still active, but the data transmission in this link is blocked by the inactive Alternate port (links drawn with thin lines in the figure). In this case, what is really silence (blocked) is the Alternate port as the link still has transmission activity from the Designated. Since data communication can only use the active links, a frame sent from $B0$ to $B1$ follows the path $B0$-$B2$-$B5$-$B1$. This is a shortest-path connection because the Root bridge is the source. However, a frame from $B6$ to $B1$ follows the path $B6$-$B3$-$B0$-$B2$-$B5$-$B1$, a clearly inefficient path.

3.1.3 Priority vectors

A bridge selects the roles of its ports comparing topological information about how far from the Root each neighbor is located. Because of the distributed nature of RSTP, this topological information is actually exchanged between neighbors. This allows each bridge to know at which distance from the Root each neighbor stays and hence elect: the port that provides the best path to the Root (the Root port); the ports where the own path is better than the neighbor's path (Designated ports); and the ports where the neighbor's path is better than the own path (Alternate ports).

These comparisons are actually implemented using *priority vectors* containing topological information about how to reach the Root from a particular location. These vectors are a set of values that relate to the distance to the Root of the tree and are defined by the following fields.

- The *Root* (r) indicates the BridgeID of the Root node.
- The *Cost* (c) is the distance to this Root.
- The *Bridge* (b) is the BridgeID of the node that owns this vector.
- And *Port* (p) stores the PortID of the port that owns this vector.

For an easier reference to the vector fields, from now on we refer to them with the notation *[r:c:b:p]*.

The priority vectors are one of the essential elements in the protocol operation as RSTP decides the port roles by comparing different vectors. A priority vector is considered better than another if it has a lower Root; or same Root and a

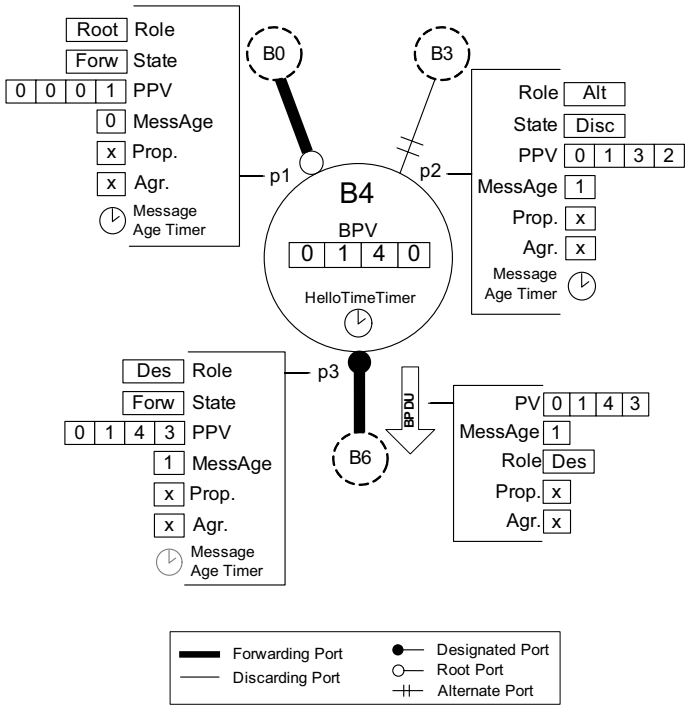


Figure 3.2: Diagram indicating the bridge and port variables stored by bridge B4 once the tree is configured

lower Cost; or same Root and Cost and a lower Bridge; or same Root and Cost and Bridge and a lower Port. For example, the vector [0:2:3:2] is better than the vector [1:0:1:2] because the Root field of the former is smaller (0<1); but the same vector [0:2:3:2] is worse than [0:1:2:2] because they both have the same Root but the latter contains a lower Cost.

Observe that the four vector fields really have a topological meaning that defines the tree structure. First, the Root field is used in the first step of the comparison as an initial distinction, before any other topological metric is considered, because it indicates which node is the Root and determines where the tree is rooted at. This results in vectors containing a lower Root always being considered better regardless the Cost, Bridge or Port values. The other three fields are topological metrics used to decide between different paths to the Root. Comparing between vectors of the same Root is really comparing between two paths to reach such Root. The distance to this node, the Cost field, is the used to decide for the vector that indicates a smaller distance to the Root. This actually results in configuring the shortest-path branches. If two vectors have also the same Cost, the node identifiers stored in the bridge field, first, and the port identifier in the port field, then, are used as tie-breakers. This last two fields also determine the shape of the tree because they actually elect between different equal cost shortest-path branches.

Table 3.1: Global Protocol Parameters

Name	*Description*
MaxAge	Maximum value for the MessAge field of BPDUs in order to limit the time a frame can exist in the network. The value of this parameter recommended by the standard is 20.
HelloTime	Time between periodical dissemination of BPDUs. The default value of this parameter recommended by the standard is 2sec.

Each node keeps one vector at the global bridge level, the *Bridge Priority Vector* (BPV), and one vector per port, the *Port Priority Vectors* (PPV). The BPV vector stores topological information that defines the tree topology from the bridge perspective. The BPV[r:c:b:p] fields contain the BridgeID of the Root of the tree (r), the cost to reach this Root (c), and the own BridgeID (b) (the port field is not used in the BPV and is set to 0). Figure 3.2 shows the variables stored by $B4$ of the previous example tree of figure 3.1. The BPV contains [0:1:4:0] because the Root is $B0$, $B4$ is at cost 1 of this Root, and the own identifier is $B4$.

Differently, the port vectors contain the topology information of the Designated port in that link (which might be either in the local bridge or in the neighbor). It is defined as the BridgeID of the Root of the tree (r), the Cost to reach this root (c), the BridgeID of the designed port of the link (b), and the portID of the designed port of the link (p). In the example of $B4$, the Root port in $p1$ and Alternate port in $p2$ store vectors of the corresponding neighbors ([0:0:0:1] and [0:1:3:2] from $B0$ and $B3$) while the Designated port in $p3$ stores a vector of the own bridge ([0:1:4:3]).

In practice, RSTP decides the port roles comparing the BPV and PPV's using the tie-breaking rules described previously. The port with the best PPV is elected as Root port as it determines the shortest-path branch that leads to the Root. In the example, $B4$ selects $p1$ because the vector [0:0:0:1] is the best among those stored in the ports. The selection of Designated and Alternate ports depends on comparing BPV and PPVs. Those ports that have a PPV worse than the BPV are elected as Designated. A BPV better than a PPV means that the bridge is closer to the Root and hence the port leads to the leaves (as Designated ports do). In $B4$, $p3$ is Designated because the BPV is better than the PPV. Otherwise, if the PPV is better than the BPV, this means that the port is closer to the Root than the bridge (which actually means that the neighbor is closer to the Root) and hence the port is set to Alternate because the port of the neighbor is the Designated. Note that an Alternate port stores a PPV, that is better than the local BPV but it is not as good as the PPV of the Root port. $B4$ selects $p2$ as Alternate port because its PPV is better than the BPV.

3.1.4 Additional variables

Although the priority vectors represent the core of the protocol operation, the bridges also keep other variables (included in tables 3.1, 3.2 and 3.3).

Table 3.2: Bridge Variables

Name	*Description*
BridgeID	Unique identifier of a bridge.
Bridge Priority Vector (BPV[r:c:b:p])	The Bridge Priority Vector contains the information that defines the tree topology from the node perspective. It is defined as the BridgeID of the Root of the tree, the cost to reach this Root, and the own BridgeID (the port field is not used in the BPV and is set to 0).
HelloTime Timer	All bridge nodes must generate periodic BPDUs to disseminate the bridge information to the neighbors in order to refresh the priority vectors. The HelloTimeTimer keeps track of the period to generate the BPDUs and its value is set to HelloTime.

Table 3.3: Port Variables

Name	*Description*
State	The state defines the operation of the ports regarding data transmission. A port in Forwarding state is active transmitting and receiving data frames while a port in Discarding state is inactive and not allowed to transmit or receive any data frames transmitted in the link. The Root port and the Designated ports are active and hence they are in Forwarding state. Note a Designated port is always in Forwarding state and hence it forwards data frames to inactive "blocked" links of the tree. However these data frames are not received by any node as neighbor ports Alternate and stay in Discarding state.
State	The role defines the responsibility of this port in the link that connects to the neighbor. There are three types of port roles. Root role: this port leads to the Root and receives the best BPDUs for the bridge to disseminate to the rest of the network; designate role: this port defines an active link down to the tree (to the leaves) and it is responsible to send the BPDUs to this link; Alternate role: this port is an extra port up to the tree (to reach the Root) but temporarily not in use, hence the port is not responsible to forward BPDUs to it.
Port Priority Vector (PPV[r:c:b:p])	The Port Priority Vector (PPV) contains the topology information of the Designated port in that link (which might belong either to the local bridge or to the neighbor). It is defined as the BridgeID of the Root of the tree (r), the cost to reach this Root (c), the BridgeID of the designed port of the link (b), and the portID of the designed port of the link (p).
MessAge	The Age of the last BPDU processed in this port.
MessageAge Timer	When a port does not receive BPDUs for the duration of 3 HelloTime's it assumes the connection to the neighbor is broken and the port initiates the reconfiguration of the tree. The MessageAgeTimer keeps track of this silence period and its value is set to 3xHelloTime.
Proposal/ Agreement	This flag indicates that the port must send a BPDU marked as 'proposal', initiating the handshake with the neighbor, or marked as 'agreement', terminating the handshake with the neighbor

- The *MessAge* is stored in each port and it contains the number of hops that the information in the PPV has traversed since it was first generated in the Root. It represents the age of the vector measured in hops and it is used to detect messages that endlessly circulate (see section 3.5 for further details on this issue). The Root port of $B4$ stores a MessAge of 0 because it is the vector directly received from the Root. However, the Designated port of $B4$ stores a MessAge of 1 computed as the MessAge of the Root port plus an increment of 1 representing one hop. If, hop after hop, the MessAge reaches its maximum value, MaxAge, the information in the vector is not considered anymore and the tree is reconfigured in the node.
- One of the most important changes that RSTP introduces is a mechanism to set Designated ports to Forwarding state but ensuring that no temporary loops are created. This is done by the proposal-agreement handshake executed between a Designated port and the port in the neighbor. The port variables *proposal* and *agreement* control such port activation. The proposal/agreement flags in the example of $B4$ are all reset because the handshake has already concluded. Section 6.2 provides a further description on how this handshake works.
- The bridges periodically send messages refreshing and confirming the vectors that they disseminate. The *HelloTimeTimer* is run by each bridge and manages this periodical transmission of messages every *HelloTime* seconds.
- In relation with this periodical refreshing, each port maintains the *MessageAgeTimer* to detect a lack of message reception. This timer is configured to 3xHelloTime and its timeout due to no refreshing results in the expiration of the port vector, which is removed and the makes the node to reconfigure the tree.

3.1.5 Bridge Protocol Data Units (BPDU)

The nodes exchange information, essentially the priority vectors, by means of Bridge Protocol Data Units (BPDU). Table 3.4 includes the details of the BPDU frame format. The BPDUs are encoded in Ethernet data frames. RSTP is a broadcast protocol and BPDUs are not sent to any particular destination but to a reserved address that indicates "all bridges" (01:80:C2:00:00:00). Thus, BPDUs are not forwarded between ports and only live in the link where they are transmitted. At most, a received BPDU in an input port might trigger the generation of another BPDU to an output port. The LLC is the bridge client that receives such BPDU and redirects it to the spanning tree protocol entity that will process the message.

The payload of the Ethernet frame is the actual BPDU that contains RSTP information and has a fixed length of 35 bytes. The fields are described following:

- The first three fields (*Protocol Identifier*, *Version*, *Message Type*) are used to distinguish the protocol version and are introduced to keep compatibility with past and future spanning tree protocols.

Table 3.4: BPDU Frame Format

Name			*Description*	*Bytes*
Ethernet encapsulation	*SA*		Management MAC address of the bridge	6
	DA		Reserved address to indicate "all bridges" (01:80:C2:00:00:00). Therefore it is a broadcast protocol and BPDUs are not sent to a particular destination. BPDUs are consumed at the next bridge hop and only live in the link they are produced. These frames never bypass a bridge but instead the bridge consumes and reacts to it producing a corresponding BPDU for the following neighbors when needed.	6
	Length/ Type		Indicates length of the BPDU.	2
	LLC		Indicates the bridge client that processes the received frame and delivers it to the STP client (DSAP:42; SSAP:42; Cntrl:03).	3
BPDU Fields	*Protocol id.*		Protocol identification fields indicating that the BPDU received corresponds to an RSTP instance.	2
	Version			1
	Message Type			1
	PV	*Root*	Contains the vector of the transmitter port defined as the BridgeID of the Root of the tree (r), the cost to reach this Root (c), the BridgeID of the transmitter bridge (b), and the portID of the transmitter port (p).	8
		Cost		4
		Bridge		8
		Port		2
	MessAge		The Age of the information that the BPDU conveys. It is defined as the number of hops the BPDU has traversed from the Root where it is initially created.	2
	Flags	*Role*	It encodes the role of the transmitting port	1
		Prop.	It indicates that the transmitter port is initiating a proposal-agreement handshake in order to transition to Forwarding	
		Agreem.	It indicates that the transmitter port is terminating a proposal-agreement handshake so the receiver port can transition to Forwarding	
	MaxAge		The global parameters are distributed by the Root so all nodes use the same values. The ForwardDelay is used by STP and kept in RSTP for backwards compatibility	2
	HelloTime			2
	ForwardDelay			2
Frame Check Sequence				4

- The priority vector of the transmitter port is included in the following 4 fields (Root, Cost, Bridge, Port). Note this is the most important information conveyed in the BPDU as it allows disseminating the own vectors to the neighbor nodes so they can compare and elect port roles.

- The *MessAge* field conveys the corresponding port variable. Note that, as a hop counter, the MessAge is incremented every time a BPDU reaches a bridge.

- The *flags* are used to encode the role of the transmitting port as well as to include the proposal and agreement flags (used in the handshake to activate a Designated port).

- The last three fields (*MaxAge*, *HelloTime*, *ForwardDelay*) encode the value of the global parameters to configure timer values. These are included in the BPDUs because the Root of the tree disseminates the values of these parameters so all bridges use the same.

3.2 Protocol operation: events and procedures

The distributed nature of RSTP results into an implementation based on the response of the protocol in the occurrence of different events. These events are classified into those that occur at bridge level (*Turn-on Bridge* and *HelloTime-Timer expiration*) and those at port level (*BPDU reception* and *MessageAge-Timer expiration*). Each one of these events triggers operations that are carried out by different procedures that are also distinguished between those that affect the entire bridge (*BecomRootBridge*, *ConfigureTree*, *PortRoleSelection* and *Port-StateTransition*) and those that only relate to a particular port (*ExpirePortInfo* and *SendBPDU*).

The interconnection between events and procedures is depicted in figure 3.3. The diagram shows an overview of the internal node operation in response to the four events listed (black circles in the figure). Without going into the details of each procedure (next sections describe in detail the pseudo-code of each block), it is easy to observe that the general operation of RSTP is based on response to the reception of BPDUs. A message with a better vector updates the current information and this triggers a tree reconfiguration (port roles and port states) and finally results into a dissemination of BPDUs announcing the new state, which in turn is received in neighbors that also update and disseminate, and so on.

Besides this action of reception-update-dissemination, there are other events that trigger protocol actions. First, when bridges turn-on all nodes become the Root of the tree and start disseminating their own BPDUs. These messages are actually the start of all BPDU receptions with the corresponding update and dissemination. This configuration with several Roots is clearly a transient situation and section 3.3 describes in more detail how the protocol behaves in this situation. Second, the expiration of the MessageAgeTimer in a port also triggers

the reconfiguration and dissemination of new state because the expired vector is removed. And third, the HelloTimeTimer periodically triggers the dissemination of BPDUs to refresh the vectors.

The following sub-sections provide the detailed description of each one of the blocks in the diagram in the form of pseudo-code. Sections 3.2.1 and 3.2.2 include the operation executed in response to Bridge events and port events, respectively. Sections 3.2.3 and 3.2.4 describe the operation of Bridge and Port procedures that are called by the previous events operation. Finally section 3.2.5 contains the operation of several auxiliary sub-routines.

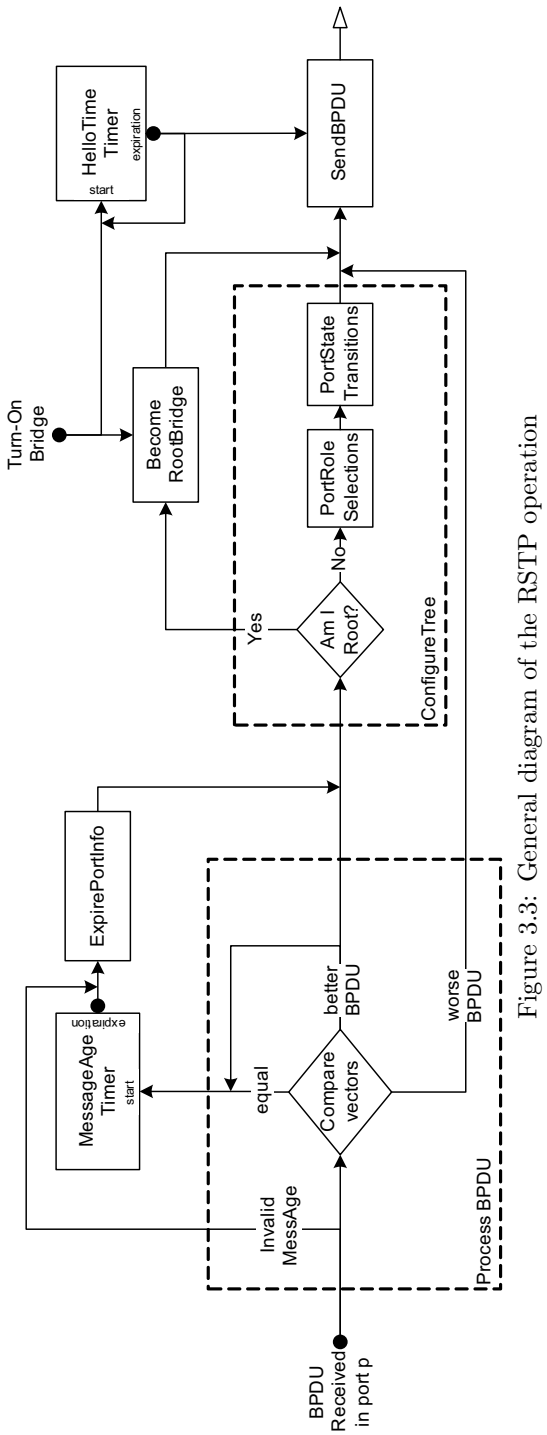


Figure 3.3: General diagram of the RSTP operation

3.2.1 Bridge events

A. Turn-on Bridge	***/* It is triggered at each node at network start-up. All bridges are transiently configured as Roots of the tree */***
1. BecomeRootNode()	*/* All variables are set using own bridge state as no other information is not available yet */*
2. Start HelloTimeTimer	*/* The bridge schedules the transmission of the first periodical BPDUs */*

B. HelloTimeTimer Expiration	***/* Each bridge individually periodically disseminates BPDUs to maintain the tree topology alive */***
1. SendBPDU() to Designated ports	*/* Only Designated ports send BPDUs down the tree */*
2. Start HelloTimeTimer	*/* The timeout for the next periodical dissemination is scheduled */*

3.2.2 Port events

C. BPDU Received in port p	***/* It is triggered when a BPDU is delivered to the RSTP instance by the LLC inferior layer */***
1. if (MessAge in BPDU is lower than MaxAge)	*/* Only BPDUs with a valid MessAge are assumed to carry fresh information and hence only these are processed */*
2. rcvdInfo = CompareVectors(PV of BPDU, PPV of p);	*/* The operation depends on whether the received vector updates the information stored in the receiving port */*
3. if (rcvdInfo is BETTER && transmitter is Design.) || (rcvdInfo is WORSE && transmitter is the parent)	*/* Receiving a BPDU that updates the port vector implies a change of information and hence the tree must be revised. A worse BPDU from a parent is considered better because it updates the last received by the same parent */*
4. Start MessageAgeTimer in port p	*/* The timer that controls the age of the received information in port p is restarted */*
5. $[r : c : b : p] = BPDU.PV[r : c : b : p]$	*/* This is a temporary local variable that stores the priority vector received in the BPDU */*
6. SetPV(PPV of port, r, c, b, p)	*/* The received information updates the port so the vector and the MessAge in the BPDU are stored as the new information in the port */*
7. Set MessAge of port equal to MessAge of BPDU	
8. ConfigureTree()	*/* An updating vector requires a tree reconfiguration */*
9. if(BPDU is proposal)	*/* A better BPDU with a proposal from a Designated port indicates that the local bridge has to reply with an agreement */*
10. PortActivationHandshake(p, 'agreem', 'start')	
11. if (rcvdInfo is WORSE && transmitter is Design.)	*/* A worse BPDU from Designated indicates the neighbor is wrong and a BPDU with own information is replied so it can be updated */*
12. SendBPDU(p)	
13. if (rcvdInfo is EQUAL && transmitter is Designated)	*/* Repeated BPDU reaffirms the state already stored thus it restarts the MessageAgeTimer */*
14. Start MessageAgeTimer in port p	
15. if (rcvdInfo is WORSE && transmitter is not Designated && agreement in BPDU)	*/* A BPDU received from a Root or Alternate port that carries an agreement concludes the proposal-agreement handshake. This results into the port p transitioning its state to Forwarding */*
16. Set State of port p to Forwarding	
17. else	*/* The BPDU is not processed if the MessAge is higher or equal than MaxAge (indicates that the information in the port is not valid anymore). The tree is reconfigured to match the new information. */*
18. ExpirePortInformation()	
19. ConfigureTree()	

D. MessageAgeTimer expiration (or physical failure detection) in port p	***/* It is triggered when a port expires the MessageAgeTimer or the physical layer detects a failure in the port */***
1. ExpirePortInformation(p)	*/* The aging out of a port information means that the stored vector is not valid anymore and hence it cannot be trusted */*
2. ConfigureTree()	*/* The tree is reconfigured to consider the changes in the information stored in the expired port */*

3.2.3 Bridge procedures

E. BecomeRootBridge()	***/* This routine resets all the information configuring all variables indicating that the bridge is the Root */***
1. SetPV(BPV, BridgeID, 0, BridgeID, 0);	*/* The BPV is reset using own information: Root is the own BridgeID; the cost to the Root is zero as it is the distance to itself; the bridge field is the own BridgeID; the port field is set to 0 */*
2. for each port 3. SetInactiveDesignatedPort(port,0) 4. SendBPDU(port)	*/* All ports are made Designated so they can disseminate BPDUs down the tree. These are initially inactive (Discarding state) waiting for the handshake to activate them. The MessAge is 0 because the Root initiates the counting. The first BPDU of this bridge as the new Root of the tree is sent */*

F. ConfigureTree ()	***/* The tree configuration is triggered when there is a change in the port vectors because the tree needs to be revised*/***
1. Select the Root Bridge	*/* The Root Bridge is the lowest BridgeID the local node is aware of: the lowest value stored in the Root field of the Root and Alternate port vectors (PPV[r]) */*
2. if (I am the Root Bridge)	*/* If the Root (lowest BridgeID in the ports) is the bridge itself, it must arise as new Root because it is not aware of any other better Root */*
3. BecomeRootBridge()	
4. else	*/* If the Bridge is Root, all ports are Designated. Otherwise, the port roles need to be elected based on the port vectors */*
5. PortRolesSelection ()	*/* The roles of each port are selected based on the new information in the port vectors */*
6. PortStateTransitions ()	*/* The states of each port are updated depending on the new roles selected */*
7. for each port p	*/* The new tree configuration is disseminated to the rest of neighbors. Designated ports send BPDUs down the tree and Root and Alternates send agreements if they have been proposed by the parent */*
8. SendBPDU(p)	

G. PortRolesSelection ()	***/* A change in a port vector requires a revision of the port roles */***
1. SetRootPort(port with best PPV)	*/* The port with the best PV is elected as Root port because it provides the best path to the Root among all bridge ports. Since the Root port provides the connection towards the Root bridge, the BPV is updated using the PPV of the Root port (See SetRootPort details) */*
2. for each other port p	*/* Once the Root port is selected, and the BPV updated, the rest of roles is decided */*
3. BridgeInfo = CompareVectors(BPV, PPV[p])	*/* The role of each other port is then selected comparing the information stored in the bridge (BPV) and in each port (PPV) */*
4. if (BridgeInfo is BETTER)	*/* If the bridge vector is better than the port vector, the bridge is the Designated node in that link and hence the port is Designated. Designated ports send BPDUs down the tree, hence the MessAge is the Root port MessAge incremented one hop. In this case the vector of the Designated is filled with own bridge information from BPV (See SetDesignatedPort for details) */*
5. SetDesignatedPort (p, Root port's MessAge + 1)	
6. else	*/* If the bridge information is worse than the port information the Designated node is the neighbor thus the port in the bridge is Alternate. The port vector is not updated because an Alternate port stores the information of the Designated in the link (the neighbor) */*
7. SetAlternatePort(p)	

H. PortStateTransitions ()	***/* The transition of the port States really configures the tree from the data traffic perspective */***
1. if (State of Root port is Forwarding);	*/* A Root port is an active port and hence no action is required if the port is already Forwarding */*
2. else	
3. Set Root port to Forwarding state	*/*If the Root port is Discarding, it directly transitions to Forwarding state. This transition can be executed and avoid potential loops if all Designated ports of the bridge are made Discarding at the same time. Note that any potential communication between the link of the Root ports and the links of the Designated ports */*
4. for each Designated port (p)	
5. Set port p to Discarding state	
6. for each Designated port p	*/* Designated ports are checked after the state of the Root port is updated (some Designated ports might be Discarding) */*
7. if (State of p is Forwarding);	*/* A Designated port is an active port and hence no action is required if the port is already Forwarding */*
8. else	*/* A Discarding Root port makes all Designated ports Discarding before it can transition to Forwarding. This results in the Designated ports starting proposal-agreement handshakes with their child neighbors */*
9. PortActivationHandshake(p, 'prop.', 'start')	
10. for each Alternate port p	*/* An Alternate port is inactive and it directly transitions to Discarding state */*
11. Set port p to Discarding state	

3.2.4 Port procedures

I. SendBPDU (port)	***/* The information encoded in the BPDU represents the state information stored in the port that disseminates the message */***
1. SetPV(BPDU, PPV[port])	*/* The priority vector PPV[port],*
2. Set MessAge of BDDU equal to MessAge of port	*the MessAge,*
3. Set Role of BPDU equal to Role of port	*and the role of the transmitting port are encoded in the message */*
4. If (port is Designated && proposal in port is true)	*/* The proposal flag to initiate a proposal-agreement handshake is set if*
5. Set proposal flag of BPDU to true	*the corresponding port variable indicates it */*
6. If (port is Root or Alternate && agreement is true)	*/* The agreement flag to terminate a proposal-agreement handshake is set*
7. Set agreement flag of BPDU to true	*if the corresponding port variable indicates it */*
8. PortActivationHandshake(p, 'prop.', 'stop')	*/* These port variables are cleared once the the BPDU flags are set */*
9. PortActivationHandshake(p, 'agreem.', 'stop')	
10. Send the constructed BPDU to port	*/* Once the BPDU fields are filled, the BPDU is sent to the outgoing port indicating the ethernet encapsulation details to the inferior sub-layer (LLC) */*

J. ExpirePortInformation(port)	***/* A port information expires when the MessageAgeTimer times out and when a BPDU with a MessAge>MaxAge is received */***
1. $[r : c : b] = BPV[r : c : b]$	*/* This is a temporary local variable that stores the three first fields of the bridge priority vector BPV */*
2. SetPV(PPV[port], r, c, b, port)	*/* As a Designated port, it disseminates down the own state, hence the PPV is filled using the bridge information in BPV */*

3.2.5 Auxiliar sub-routines

K. CompareVectors (A, B)	***/* It returns whether vector A is considered BETTER, EQUAL or WORSE than B according to the tiebreak rules */***
1. if (A.Root < B.Root ||	*/* Vector A is better than B if it has a lower Root, */*
2. (A.Root == B.Root && A.cost < B.cost) ||	*/* or same Root and a lower cost, */*
3. (A.Root == B.Root && A.cost == B.cost && A.bridge < B.bridge) ||	*/* or same Root, same cost and a lower bridge, */*
4. (A.Root == B.Root && A.cost == B.cost && A.bridge == B.bridge && A.port < B.port)	*/* or same Root, same cost, same bridge and a lower port */*
5. return 'BETTER'	
6. else if (A.Root == B.Root && A.cost == B.cost && A.bridge == B.bridge && A.port == B.port)	*/* Vectors A and B are EQUAL if all fields are the identical */*
7. return 'EQUAL'	
8. else	
9. return 'WORSE'	*/* A is WORSE than B if none of the previous conditions are met */*

L. SetInactiveDesignatedPort (port, mAge)	***/* Setting an inactive Designated port implies forcing the state to Discarding and start a handshake to activate it */***
1. SetDesignatedRole(p, mAge)	*/* The port variables are updated according to the Designated role of the port (See SetDesignatedRole for details) */*
2. Set State of port p to Discarding	*/* The port state is set to Discarding as a transient situation waiting for an agreement that closes the handshake and activates it */*
3. PortActivationHandshake(p, 'proposal', 'start')	*/* The Designated port is Discarding and hence starts a handshake sending a proposal */*

M. SetDesignatedPort (port, mAge)	***/* Setting a Designated port includes updating the port variables: role, PPV, MessAge and proposal */***
1. Set Role of port to Designated	
2. $[r : c : b] = BPV[r : c : b]$	*/* This is a temporary local variable that stores the three first fields of the bridge priority vector BPV */*
3. SetPV(PPV[p], r, c, b, portID of p)	*/* As a Designated port, it disseminates down the own state, hence the PPV is filled using the bridge information in BPV */*
4. Set MessAge of port to mAge	*/* The value of MessAge is a parameter because different situations result in different MessAge values for Designated ports */*
5. PortActivationHandshake(p, 'agreement', 'stop')	*/* A port selected as Designated does not send agreements down */*

N. SetRootPort(port)	***/* Setting the Root port implies changing the Role and updating the BPV */***
1. Set Role of port to Root	
2. $[r : c : b] = PPV[r : c : b]$	*/* Temporary local variable that stores the three first fields of the Root port priority vector (PPV[Root port]) */*
3. SetPV(BPV, r, c+1, b, 0)	*/* The BPV is updated using the information in the PPV of the Root port (the cost increases) as it determines how the bridge reaches the Root of the tree */*

O. SetAlternatePort (port)	***/* Setting an Alternate port only implies changing the role of the port */***
1. Set port to Alternate role	*/* No vector is updated because it stores the vector of the Designated in that link (the neighbor) */*

P. SetPV (vect, r, c, b, p)	***/* Routine that copies the vector fields (arguments r,c,b,p) into the vector in vect */***
1. $vect = [r : c : b : p]$;	

Q. PortActivationHandshake (port, type, action)	***/* Routine that updates the port proposal/agreement flags */***
1. if (type == 'prop')	
2. if (action == 'start')	*/* The proposal flag of the port is set if the next BPDU needs to be sent marked as*
3. Set proposal flag of port to true	*a proposal to start a handshake asking confirmation to activate a Designated port */*
4. else if (action == 'stop')	*/* The proposal flag of the port is reset if no more BPDUs need to be sent marked*
5. Set proposal flag of port to false	*as a proposal because the handshake has already been started */*
6. if (type == 'agreem')	
2. if (action == 'start')	*/* The agreement flag of the port is set if the next BPDU needs to be sent marked as*
3. Set proposal flag of port to true	*an agreement to conclude a handshake and activate the Designated port*/*
4. else if (action == 'stop')	*/* The agreement flag of the port is reset if no more BPDUs need to be sent marked*
5. Set proposal flag of port to false	*as a proposal because the handshake has already been terminated */*

3.3 Initial configuration of the tree

This section analyzes the behavior of the protocol in a network start-up scenario where all bridges are turned-on (commonly referred as *cold-start*). Although this is a situation that does not occur very often in production networks (only if a global reset is required), it is necessary to study it because it provides the fundamental information to comprehend how the protocol operates when constructing the tree from scratch. Therefore, understanding the behavior during this initial tree construction is crucial to later understand how the protocol recovers from failure situations.

3.3.1 Initialization of bridges

Because of the distributed nature of the protocol, the same initialization is applied to all nodes at network start-up: each node assumes to be the Root regardless of its BridgeID. At the beginning, RSTP nodes do not know the existence of any other so they assume they are the only bridge in the network, therefore their BridgeID is the lowest, and hence they configure themselves as the Root of the tree (event *Turn-on Bridge* described in block A of section 3.2.1).

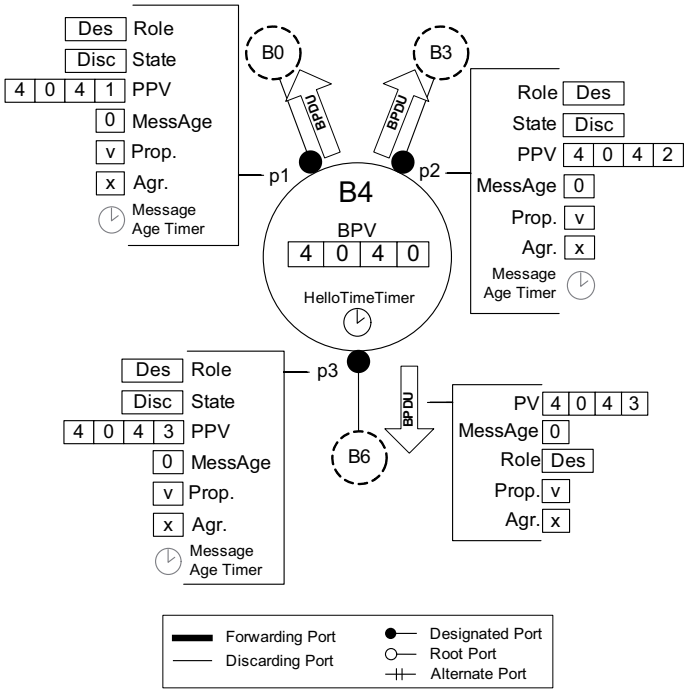
Figure 3.4(a) shows the diagram with the initial $B4$ configuration. Becoming the Root of the tree implies that the vectors contain information about how to reach itself (procedure *BecomeRootBridge* in block E of section 3.2.3). In the example, $B4$'s BPV is set to [4:0:4:0] because $B4$ is the Root and $B4$ is located at distance 0 of itself. As a Root, all ports are branches that lead to the leaves, therefore all ports are Designated. This implies that all port vectors store information about the own Root, [4:0:4:p], so it can be disseminated to neighbors.

It is important to note that the Designated ports are still inactive (Discarding state) and this is why the sent BPDUs are marked as 'proposal' so as to start a handshake with the neighbor with the objective to activate the Designated in the Root bridge. In addition, since the Root port is the origin of its own BPDUs, the hop counter MessAge of each port is set to 0.

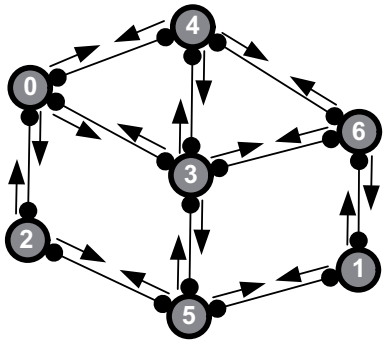
A similar configuration as this one of $B4$ is set into each bridge of the network (only changing the BridgeID of each node). This means that, at the beginning, there are as many Roots as nodes trying to construct the tree rooted at themselves (figure 3.4(b)). Note that this is clearly a transient situation because there can only be one Root and one tree. The protocol operation ensures that all bridges eventually agree on one Root and one tree, and it is concretely these initial messages sent by each bridge that trigger all the process.

3.3.2 Processing of a received BPDU

All nodes are initially configured as Roots and send the corresponding BPDUs. This results in every bridge receiving messages form each neighbor. The operation executed by the RSTP instance at the occurrence of a BPDU reception is detailed in the bridge event *BPDU received in port p* of block C in section 3.2.2. In order to understand the protocol operation when processing a received BPDU we will



(a) Detail of the initial configuration in B4



(b) All nodes send their BPDUs

Figure 3.4: Initial configuration of bridges (all nodes are Roots)

review in detail the reception of the message that $B0$ sends to $B4$. Figure 3.5 shows the diagrams with the entire configuration in all the steps involved.

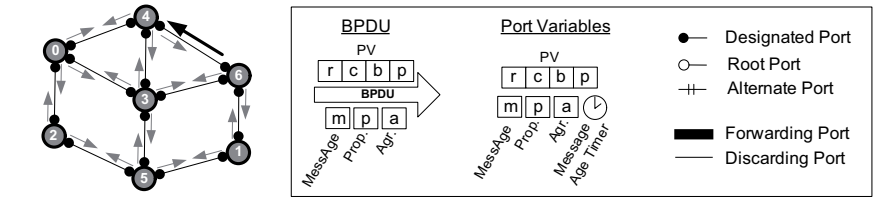
The protocol is based on the state evolution triggered by updates of topological information (vectors) that are received through of BPDUs. Therefore a BPDU that carries a vector with updated information triggers a tree reconfiguration to match the just received new information. This condition is checked comparing the vector in the BPDU with the vector stored in the receiving port (line 2 in block C; note that the detailed comparison rules are described in block K of section 3.2.5). In figure 3.5(a), $B4$ receives a BPDU from $B0$ with the vector [0:0:0:1] (announcing $B0$, itself, as Root) and its port $p1$ stores [4:0:4:1] (believing $B4$, itself, as Root). The received vector is considered better than the port vector because the Root field is lower ($0<4$). As shown in figure 3.5(b), a received BPDU with a better vector implies that new topology information is available in the port (lines 4 to 7 in block C) and hence the tree configuration in the local bridge must be revised (line 8 in block C). In addition, the MessageAgeTimer of the port is started because the vector that this port now stores has been received from the neighbor and the ageing of such information needs to be controlled.

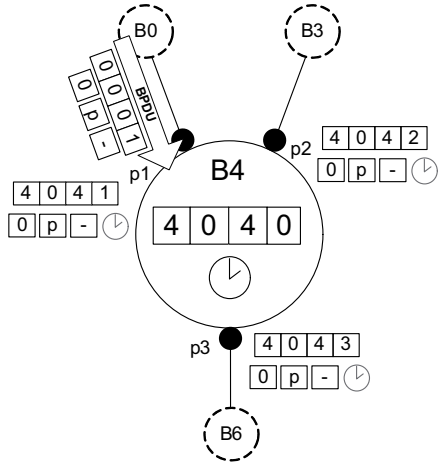
Configuring the tree involves the selection of the port roles based on the vectors information and the transition of the port states depending on the new roles configuration. This operation is detailed in procedures of section 3.2.3: *ConfigureTree* in block F , *PortRoleSelection* in G, and *PortStateTransition* in H. The first step in revising the tree is checking which bridge is the Root based on the available information. This is simply done looking up for the lowest Bridge value stored in the port vectors (line 1 of block F). In the example, the just received vector indicates that $B0$ is the lowest bridge identifier that $B4$ is aware of, hence $B4$ now believes that $B0$ is the Root.

The fact that the selected Root is another node implies that the tree must be configured to match this new situation. The first step is to set the port roles (line 5 of block F), starting by choosing port with the best vector as the Root port (line 1 of block G). In figure 3.5(c) $B4$ selects $p1$ as Root port because it is the only port vector that contains $B0$ as Root, while other ports have the own identifier $B4$. Setting the Root port means configuring the path to the Root, hence the bridge vector BPV is updated with the vector in the Root port. Note that all vector values are copied except the cost that is incremented. Observe that at this point $B4$ already believes that the Root is $B0$ and at distance 1.

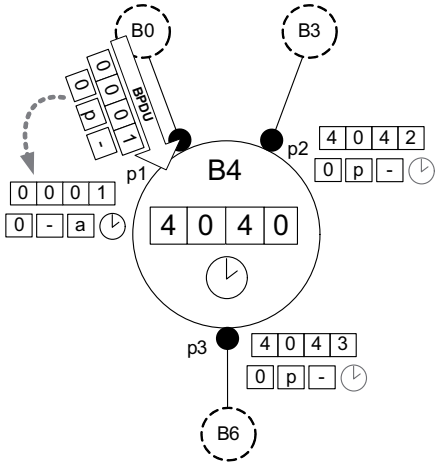
After the bridge realizes its way to reach the Root, through the Root port, it selects the roles of the rest of ports between Designated and Alternate (lines 2 to 7 of block G). As already explained in section 3.1, a port vector whose PPV is worse than the BPV becomes Designated because it leads to the leaves; otherwise it becomes Alternate. In figure 3.5(d), $B4$ selects both ports as Designated because the freshly received vector in $p1$ contains a lower Root. Since the Designated ports store vectors about the local bridge, the BPV is used to update the PPV of each one. The MessAge is also updated setting it to the MessAge in the Root port plus the increment that represents the additional hop from the Root.

The change in the port roles drives the transition of port states as detailed in block H of section 3.2.3. This procedure aims at transitioning the Root port and

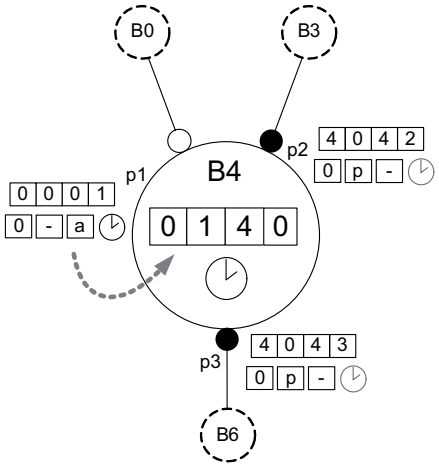




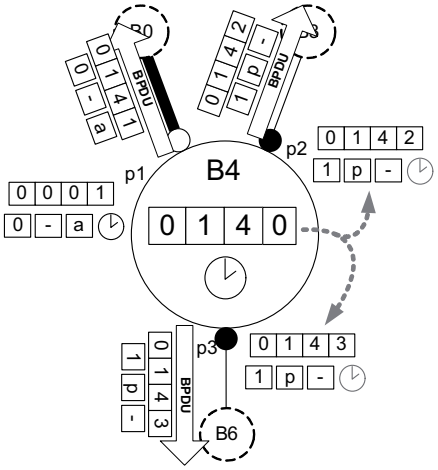
(a) Comparison of received and port vectors



(b) Received is better and hence updates the port information



(c) Selection of the Root port



(d) Selection of other ports and dissemination

Figure 3.5: Protocol operation in the event of a BPDU reception

Designated ports to Forwarding and the Alternate ports to Discarding. However, these transitions need to be safely executed in order to avoid the creation of temporary forwarding loops. First, it is safe to transition a Root port to Forwarding if we ensure that the rest of ports are Discarding. Note that forcing the Designated ports to Discarding creates a blocking "barrier" that breaks any potential transient loop. This is why all Designated ports are moved to Discarding state before the Root port is made Forwarding (lines 3 to 5 of block H). Second, the transition of a Designated port to Forwarding is based in the proposal-agreement handshake. A Discarding Designated port starts a handshake sending a BPDU marked as proposal in order to receive the corresponding agreement from the neighbor and, only then, transition the Designated port to Forwarding. And third, an Alternate port is directly set to Discarding as it does not represent any problem in terms of loop creation. In figure 3.5(d), observe how the Root port is made Forwarding while the Designated ports remain Discarding but are marked to send a proposal in the next BPDU so as to start the handshake with their corresponding neighbors.

The last step of the tree configuration is to disseminate the new state to neighbors so these can also update their port roles and states (line 8 of block F). BPDUs are sent through Designated ports (starting a handshake procedure if the proposal flag is marked) and BPDUs with agreements are sent back to the newly elected Root and Alternate ports (completing the handshake if the agreement flag is marked). In example of figure 3.5(d), $B4$ disseminates the new information by sending a BPDU through the Designated ports to $B3$ and $B6$; it also confirms to $B0$ with a BPDU with an agreement. The reception of this last agreement message in $B0$, results into setting its Designated port to Forwarding state (lines 15 and 16 of block C).

The previous example has detailed the operation of processing a received BPDU with a vector that updates the port vector. There is also the case that the received message carries either a worse (line 11 of block C) or equal vector (line 13 of block C). An example of the first case is the processing of the BPDU that $B4$ initially sends to $B0$ (actually in the opposite direction of the previously BPDU). In this case, $B0$ compares the received and stored vector and realizes that the message does not update the local state, hence there is no change in topology information and the tree does not need to be revised. The only action that $B0$ executes is to reply to $B4$ with a BPDU as $B0$ assumes $B4$ is wrong and hence it has to update the tree. Note that when this replied BPDU is received in $B4$, it will be seen as equal information (second case). $B4$ does not take any action but re-starting the MessageAgeTimer because the equal BPDU actually confirms and refreshes the vector.

An additional particularity of the message processing is how to proceed depending on the value of the hop counter, MessAge, of the BPDU. Recall that the Root node sends its initial BPDUs with a MessAge equal to 0 and that this value is incremented every time the information is disseminated by Designated ports (line 5 of block G). The common message processing is applied always that the received MessAge is lower than MaxAge (line 1 of block C). If otherwise MessAge is larger than MaxAge (line 17 of block C), the message is discarded because it

has traversed too many hops and it is a signal of a potential problem (section 3.5 describes this issue in more detail). In a cold start scenario like this one, this condition is only met if the BPDU originated in a node traverses more hops than MaxAge. An immediate consequence of this restriction is that the network size is limited by the value of MaxAge. To be safe, MaxAge should be set to the number of nodes in the topology. However, the recommended value by the standard is 20.

3.4 Failure detection and recovery

Once the tree has been initially configured, the role of RSTP is to maintain and reconfigure it in the event of failures.

3.4.1 Failure detection

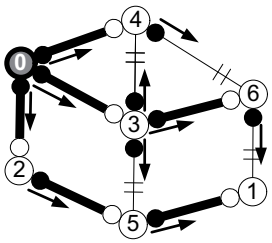
The protocol bases the detection of failures on actually detecting the lack of BPDU receptions. This can be assumed because the RSTP bridges send periodical BPDUs through their Designated ports every HelloTime (event *HelloTime-Timer Expiration* in block B of section 3.2.1). RSTP bridges independently send these periodical messages to their child neighbors (like local independent heartbeats). As shown in figure 3.6(a), the messages are transmitted towards the leaves of the tree from Designated to Root or Alternate ports. These messages are received and are used to refresh the vectors that they store (ports $p1$ and $p2$ of $B4$ in figure 3.6(b)). Note that these periodical messages are considered equal (line 13 of block C) and hence the MessageAgeTimer is restarted et every periodical reception.

If three in a row of these BPDUs are not received (this is 3xHelloTime after the last reception), the timer expires and vector stored in that particular port is removed (event *MessageAgeTimer expiration* in block E of section 3.2.2). This signal is interpreted as a lost of connectivity through that port and hence the information stored in cannot be trusted anymore. A change in the topological information of the port vectors results into a potential change in the tree, so this must be reconfigured if the information in a port ages out. The failure detection can be improved if bridge ports can directly detect a failed link at physical level. This results into an immediate reconfiguration just after the physical detection is notified to the RSTP instance.

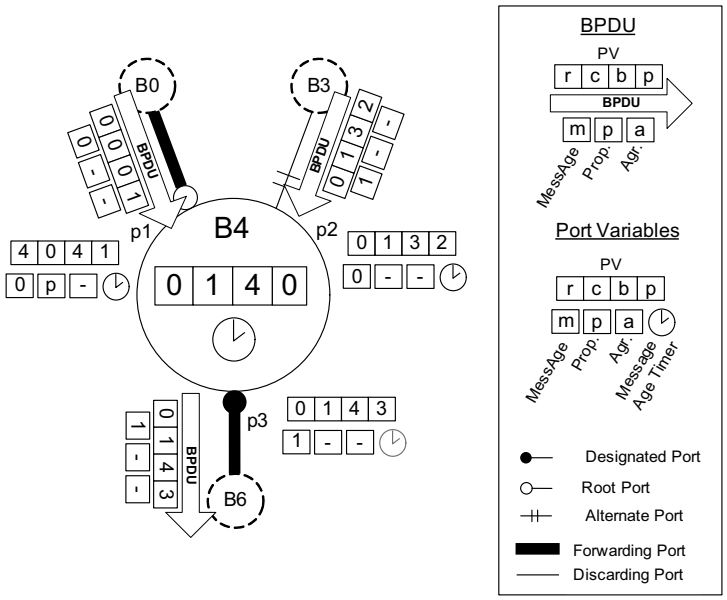
3.4.2 Link failure recovery

The recovery form the failure example shown in figure 3.7(a)) is described following. In order to detail the example, the diagrams in figure 3.8 depict the operation of RSTP to recover the tree after the failure of the link between $B0$ and $B2$.

Since the link has failed, $B2$ stops receiving the refreshment BPDUs from $B0$. However, both the timer expiration due to lack of receptions and the immediate detection at physical level trigger the same recovery operation. $B2$ removes the

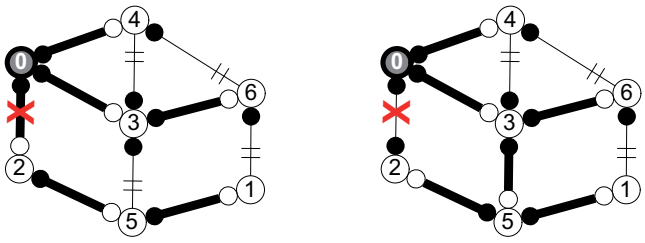


(a) All Designated send the periodical BPDUs



(b) Detail of bridge B4 receiving and transmitting refreshing BPDU

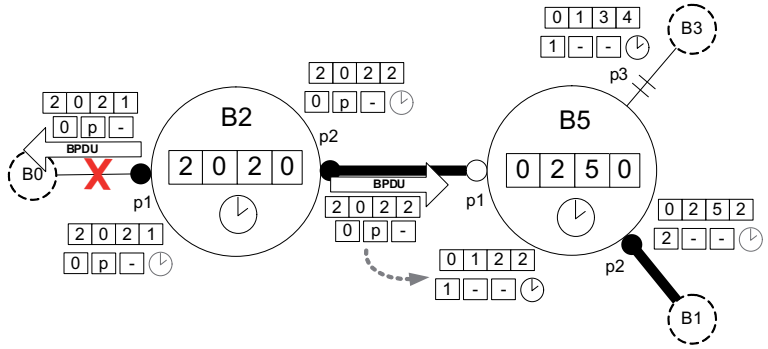
Figure 3.6: Transmission of periodical BPDUs to refresh the vectors



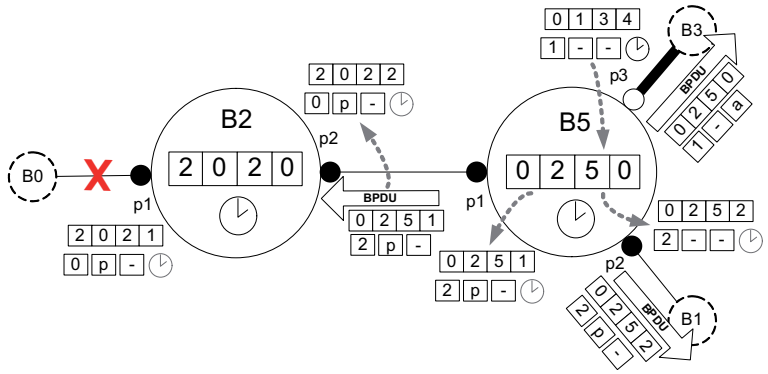
(a) Tree before the recovery

(b) Recovered tree

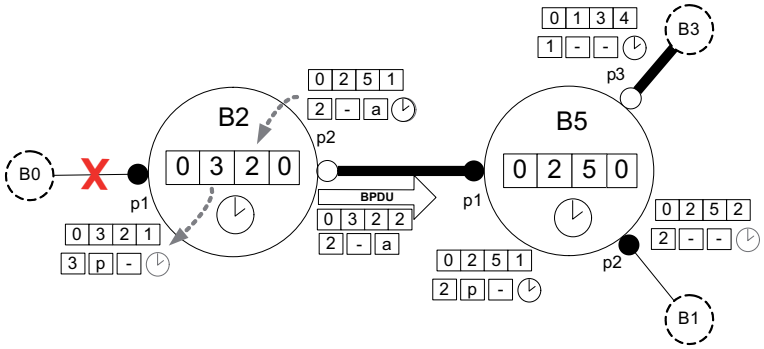
Figure 3.7: Recovered tree by RSTP after a link failure



(a) B2 arises as Root



(b) B5 accepts worse vector and selects old Alternate as new Root



(c) B2 receives vector about B0 and updates

Figure 3.8: Diagram of exchanged messages in RSTP in the event of the B0-B2 link failure scenario

vector stored in the failed port, which was actually its Root port, and when it reconfigures the tree it realizes that the lowest Root it is aware of is itself because it has no Root or Alternate ports left (line 1 of block F). Therefore, $B2$ arises as Root and disseminates its own BPDU (figure 3.8(a)).

$B5$ receives this message and, even if it carries a worse vector (a higher Root), it is considered to update because it is received from the parent (figure 3.8(b)). The parent is always trusted because it provides the path to the Root, so a change in the information that the parent notifies results into a direct update regardless the vector. $B5$ reconfigures the tree and now selects the old Alternate port as new Root port because it now stores the best vector (note that the vector in the old Root has been updated by the message just received from the parent). The change of Root port implies a change on the BPV and a reselection of the rest of roles. $B5$ selects the other ports as Designated, forces them to Discarding to allow the Root port to Forwarding, and disseminates the new information. $B2$ receives the new vector from $B5$ and considers that it updates because it has a lower Root (figure 3.8(c)). The corresponding reconfiguration selects the new Root port and B2 sends back the last agreement that concludes the tree recovery (figure 3.7(b)).

A similar operation of recovery happens in the failure of a non-Root node. The only difference is that different neighbors detect the failure and start the previous process in parallel. If $B3$ fails, several nodes - $B4$, $B5$, $B6$ - reconfigure their trees and disseminate messages. The propagation of these messages follows the same rules and evolves similarly to the single link failure case. A particular scenario appears when the Root fails. In this case the protocol elects a new Root, the node with the second lowest identifier, and configures a new tree. In theory, the Root neighbors detect the failures on their links and each one arises as new Root. This should result into a similar situation as in a cold start where each node is initialized as Root. However, there are some particular conditions that trigger a count-to-infinity behavior: an unstable situation with looping BPDUs that delay the convergence of the protocol after the Root failure. Section 3.5 describes and analyzes in detail this particular situation.

3.5 Root failure consequences and count-to-infinity

As seen in section 3.4, the recovery of the active topology after a link failure, or a non-Root node, requires a few BPDUs to be exchanged and hence the recovery time depends (1) on the detection time and (2) on the propagation delay propagation involved in the transmission of such messages. A different situation occurs when the failed device is the Root of the tree. The distance-vector protocols experience count-to-infinity when a destination fails. Since in the computation of the active tree in RSTP the Root acts as a destination, count-to-infinity appears when this node fails. [14][15].

The count-to-infinity effect is an unstable behavior that leads to the uncontrolled dissemination of BPDUs. This effect delays the recovery time until these messages are removed. As stated in [15], the condition required for count-to-

infinity to appear is that there exists at least one physical loop in the remaining topology after the Root node has failed.

A simple example topology that meets such condition is shown in figure 3.9 (the outer node $B0$ is the Root that fails). The Root $B0$ fails at t_f, hence $B1$ should arise as new Root as it has the lowest of the remaining identifiers. For simplicity reasons, the bridge vectors (represented by boxes next to each node) and the BPDUs (represented by arrows) only show the Root and Cost fields. The rest of fields are not included in the example because the vector comparisons do not reach the third step (comparing Bridge) hence it is not necessary to include them.

$B1$ first detects the failure of its port $p1$ at t_f. $B1$ has no Root or Alternate ports left so it arises as potential new Root and sends BPDUs announcing it (1:0). The colored bridge vector box indicates there has been a change in the Root field (from 0 to 1) and in the cost field (from 1 to 0).

At t_1, $B2$ receives the information about $B1$ on its Root port $p1$. The vector in the BPDU is worse than the locally stored because the message is announcing a Root with a higher identifier ($1>0$). However, the vector is transmitted by the parent, who is always trusted, and hence $B2$ overwrites the previous vector and reconfigures the tree. Since $B2$ has no Alternate ports, it notices that $B0$ is not available anymore and sets $B1$ as the current Root. $B2$ also configures the Cost to 1 as the summation of 0 (received) plus 1 (link cost). $B2$ disseminates the new information to neighbors (1:1).

Also at t_1, $B3$ receives the worse BPDU from the parent $B1$ and also reconfigures the tree. The difference with the previous case is that $B3$ has an Alternate port on $p2$ that provides an extra path to the Root $B0$. Because of this information in the Alternate port, $B3$ still believes $B0$ is alive and selects this Alternate as new Root port. $B3$ is now trusting in the vector that is stored in the old Alternate port, which contains the really failed $B0$ as Root and at cost 3. Even though this is a mistaken decision, $B3$ does not really know that $B0$ has failed hence it disseminates this misleading information (0:3). At t_2, $B3$ receives the message from $B2$ with $B1$ as Root. Since it comes from the parent, it is accepted and disseminated (1:2). Also, $B1$ receives the BPDU from $B3$ about $B0$. This message is considered better, even if it is false information, because it carries a lower Root ($0<1$). $B1$ reconfigures the tree assuming now $B0$ as Root, at cost 4 ($3+1$), and disseminates again (0:4). Observe that the stale information about $B0$ originated by $B3$ at t_1 now reaches $B1$, also mistakenly believing that $B0$ is still alive, and continues its propagation. At t_3, $B2$ receives the BPDU from $B1$ with $B0$ as Root, it accepts it and also disseminates the false vector (0:5). Also at t_3, $B1$ receives the BPDU from $B3$ about $B1$. This is considered worse and discarded because $1<0$. At t_4 and t_5, $B3$ and $B1$ keep receiving the BPDU with $B0$ as Root and of increasing Cost.

Observe how this BPDU with false information is looping in the cycle topology and the cost is incremented hop by hop. Note that the name of the effect refers to the cost field counting to infinity if looping messages are not detected. The count-to-infinity effect does not terminate unless the corresponding distance-vector protocol implements a mechanism that detects the situation and termi-

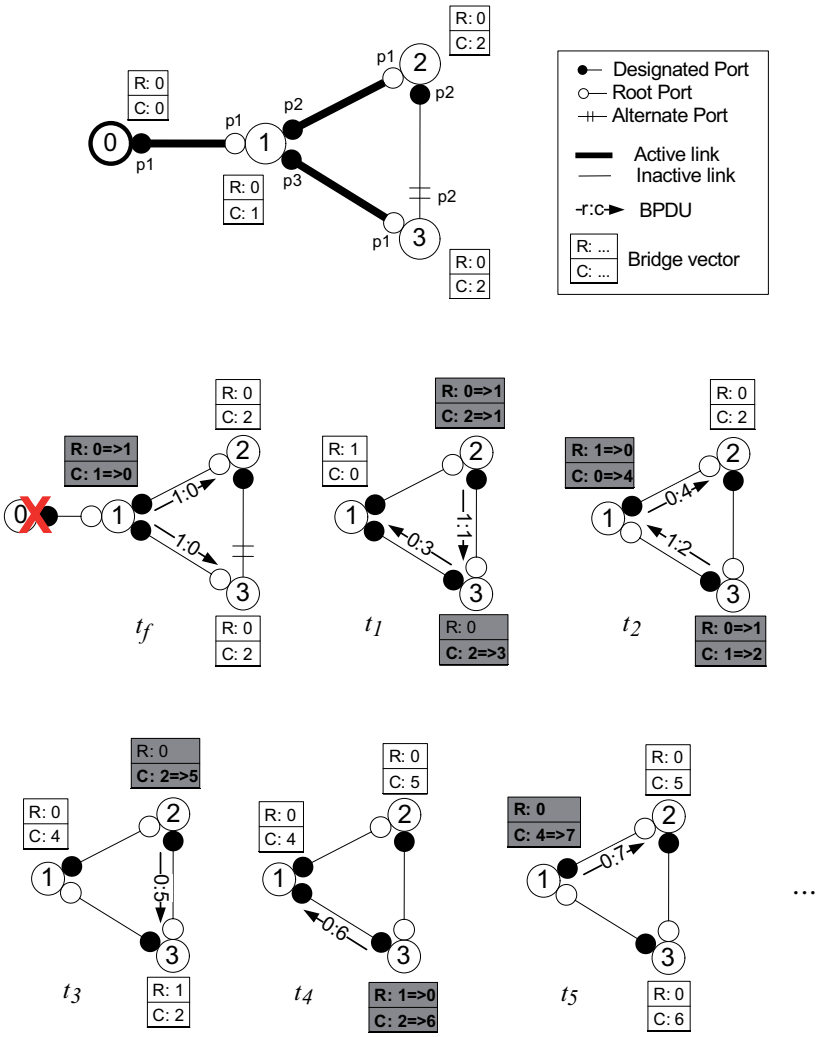


Figure 3.9: Count-to-infinity experienced in the example network with the Root B0 failing at t_f.

nates it. RSTP uses the field MessAge in the BPDUs as a hop counter to detect and discard the looping messages. This hop counter is incremented every time a bridge disseminates a BPDU through Designated ports after receiving one in the Root port. When a node receives a BPDU with a MessAge larger than its maximum value MaxAge (20 by default), the message is discarded and the information in the port is removed. Note that this operation is similar to the response to failure detection in that port. This way, nodes eventually remove all false BPDUs conveying vectors about the failed Root $B0$. Once these have been removed, only messages about the alive nodes are being exchanged, similar to a cold-start

where the wave-front of $B1$ spans the entire networks and arises as new Root.

The cause of the count-to-infinity behavior is the use of stale information stored in Alternate ports. The event that triggers the process is the update of the current Root port when the Root is changing. If the bridge has an Alternate port with information about the failed Root, this becomes the new Root port and messages about the failed Root are disseminated to neighbors. If the bridge does not have Alternate ports, it arises as new Root and the corresponding messages informing of the failure are disseminated to the neighbors. Either fresh or stale messages are received in child neighbors in their Root ports. The key issue is that RSTP accepts all messages from parent even if they are worse than the previous received. Hence, old information keeps moving around untouched. The count-to-infinity clearly results into a larger recovery time after the failure of the Root. The protocol first needs to reach the maximum value of the MessAge field (20 hop delays), and then the new Root needs to disseminate its BPDUs to the entire network (one-way trip delay). In addition, this situation also leads to a high number of unnecessary messaged being transmitted. This results into a remarkable increase of the protocol overhead in the critical case of the Root failure.

§ 4. Review of Proposed Ethernet Bridging Control Protocols

This chapter includes a review of the path selection techniques that aim at improving the performance of Ethernet Bridging networks in terms of provision of connectivity. These can be divided into 3 different groups depending on the nature of the protocols.

MSTP-based proposals MSTP [16] is essentially a framework to deploy multiple tree instances that provides techniques to manage them. There are several proposals based on the use of MSTP that introduce some particularities on which trees are really created and how these are managed. These proposals only use MSTP as a framework to deploy different trees in different VLANs. The mechanism to construct the trees, the spanning tree protocol, is however removed and substituted by offline techniques. The externally computed trees are then configured in the network bridges before the communication starts. Using a multi-tree framework mainly allows for improvements in terms of recovery time thanks to the use of pre-configured backup trees. Other aspects such load balancing and resources utilization are also addressed in a second term.

Routed Solutions One of the pillars of Ethernet bridging is its plug-n-play property due to the automatic MAC learning operation. A different approach is to avoid this automatic and introduce routing techniques where the bridge forwarding tables are configured instead of learned from the network operation. The operation of these proposals is based on using a routing protocol to construct point-to-point paths between all nodes and configure this information into the forwarding tables. These routing mechanism are mainly based on distance-vector or link-state protocols already designed for higher layer networking solutions.

Turn prohibition A different approach to overcome the drawbacks of the spanning tree bridging is based on a new loop-avoidance paradigm called turn-prohibition. Instead of blocking links as the spanning tree protocols do (link-prohibition), these protocols avoid certain turns in the topology. A turn is generally defined by a sequence of 3 nodes (A,B,C) where a message transmitted from A received by B is directed to C. Therefore forbidding a turn means blocking the communication to C when the message comes from A. This way, a higher number of links is used and more optimal paths are constructed.

4.1 Framework and comparison overview

This section provides a general review of the related work summarizing the main characteristics of each group and providing a qualitative comparison. The analysis is divided into three topics: *Performance*, *Properties* and *Operation*. Each one of these and the particular details that we compare are described following. Each comparison topics includes the corresponding table with the details for each proposal. Note that the IEEE 802.1 protocols introduced in section 2.1 are also included in the comparisons to set the reference point of the standardized techniques.

Performance

- **Recovery time**: time to recover the network operation after a failure event described as the main factor that determines the time.
- **Resource utilization**: amount of active links that are really used for data communication.
- **Path optimization**: degree of path optimality (whether the paths are more or less close to optimal shortest-paths).

Properties

- **Path control**: flexibility to decide the nodes and links that a path traverses.
- **Plug-n-play**: whether a node can immediately start transmitting data after it is connected.
- **Backwards compatible**: whether the proposal can coexist with standard techniques.
- **End-node transparent**: whether the end-nodes are transparent to network operation and hence all networking operation is made by the bridges.

Operation

- **Table population**: whether the MAC addresses are automatically learnt from received data frames or configured in-advance (manually or by means of an automatic routing technique)
- **Path selection nature**: whether the paths are selected using a centralized/distributed technique, based on link-state/distance-vector, etc.
- **Active topology type:** whether the constructed active topology is a single tree, several trees, individual point-to-point paths, etc.

- **Protection/Restoration**: whether a technique makes use of a pre-constructed backup topology (protection) or recomputes it at the event of failures (restoration).

Tables 4.1, 4.2 and 4.3 summarize the comparison between the different proposals in terms of performance, properties and operation (respectively). A global conclusion of the review of each of the initially described groups is provided following.

- **IEEE Standards.** The evolution of IEEE 802.1 protocols has always been triggered by the need to improve one of the performance aspects: recovery time (RSTP), resource utilization (MSTP), and path optimality (SPB). However, these enhancements come at a cost with the increase of configuration complexity and a corresponding reduction in plug-n-play property. Also note how the operation is based on bridging fundamentals with a distance-vector protocol except in SPB where a link-state solution is introduced.

- **MSTP-based extensions**. The use of the MSTP framework to deploy multiple trees provides a clear improvement in terms of resource utilization. In addition, it opens the door to provide Traffic Engineering and therefore meet the providers' requirements. However, all this comes with an increase of configuration complexity, mainly due to the use of MSTP as a framework. Most of the techniques propose a protection scheme based on the use of backup trees in order to obtain a quicker recovery time. However, this approach introduces two additional problems. First, switching from tree to tree might result in temporary loops because the path in one tree concatenated to the path in another independent tree might be a cycle. And second, the construction of the primary and backup trees is done at network startup before any communication starts, therefore removing the plug-n-play capability of original Ethernet bridging.

- **Routed approaches**. The techniques based on applying routed protocols show a significant improvement in terms of resource utilization and path optimality. This is because of the construction of individual point-to-point paths between network nodes. This operation does not follow the bridging principles and compromises automaticity aspects such the need to disseminate the end-hosts MAC addresses by means of a routing technique. All protocols rely on a distributed technique and are based on a restoration approach, but there main difference is in the paradigm used in the topology acquisition: distance-vector or link-state. Note that the use of point-to-point communication results into removing the broadcast condition of Ethernet bridging.

- **Turn-prohibition**. This completely different approach introduces some benefits but in general results into not compatible techniques. Moreover, the paths are not guaranteed to be and the resource utilization depends on

the algorithm used to calculate the legal turns. As in routed approaches, the paths need to be calculated by each node after they have received the physical topology graph, so the technique is not plug-n-play and it requires a configuration complexity similar to routed approaches.

Certain global conclusions can be extracted from this literature review in order to put in perspective our approach. First of all, all reviewed solutions actually improve the performance of the spanning tree protocols either in terms of recovery time, number of links used or path optimality. However, how this is achieved introduces additional points to discuss. Many techniques describe high-level add-on solutions that do not really address the fundamental problems we want to tackle in this thesis. Moreover, some proposals use a completely different architecture substituting essential bridging functionality. Since one of our objectives is to keep the fundamental Ethernet operation (principally automatic MAC learning and broadcast condition), we cannot introduce into our solution any technique that removes of substitutes such basic operations. This is the case of the routed approaches that are based on implementing a whole routing mechanism to populate the forwarding tables substituting the fundamental Ethernet automatic learning.

In order to solve the recovery problem of RSTP, due to count-to-infinity, we discard the option to deploy backup trees because this implies a pre-computation of paths that removes the plug-n-play property of Ethernet. Another option would be to use the link-state path selection paradigm because it does not suffer count-to-infinity. We actually further study this option in chapter 5, with a detailed SPB description, and in chapter 8, with a performance analysis comparing with RSTP-SP. In order to solve the count-to-infinity problem we instead opt for a detailed analysis of the RSTP protocol in order to identify its causes and design the concrete extension that avoids it (see chapter 7).

Another aspect of this work that can be borrowed and explored is the use of multiple trees to address the low resource utilization and sub-optimal paths of the spanning tree protocols. The use of an active topology with multiple trees is definitely a good solution to improve topological performance issues like the number of links used and path optimality. As seen in the comparisons, not any set of multiple trees can be used to deploy optimal communications. The active topology composed by one tree rooted at each node in SPB seems to be the best option. We follow this idea in the RSTP-SP proposal where we extend the original single tree RSTP to deploy one tree rooted at each node in order to operate with optimal communication paths (see chapter 8). Note that the proposals based on the routed approach also operate in shortest paths. We however have discarded this option because they substitute the fundamental automatic learning by a configuration technique.

Table 4.1: Performance aspects of related work techniques

Protocol		*Performance*		
		Recovery time	*Resource utilization*	*Path optimization*
IEEE 802.1	*STP*	fixed timer of 50*sec*	Only single tree links	Only single tree links
	RSTP	Propagation from failure down the affected tree branch*	Only single tree links	Only single tree paths
	MSTP	Propagation from failure down the affected tree branch*	Depends on the number of trees	Only single tree paths
	SPB	Propagation from failure to furthest node	Potentially all links (one tree per node)	All Shortest paths (one tree per node)
MSTP - based	*[38]*	Not specified	Links of two single trees	Slightly better than single tree
	[39]	Not specified	Depends on the number of trees	Better than single tree but not optimal
	[40]	local backup tree failover		
	[41]	local backup tree failover		
	[42]	Notification to edge nodes of the tree failover		
	Viking	Notification back and forth to and from the central control		
Routed Approaches	*STAR*	Not specified	Links of single tree plus some point-to-point paths	Paths of single tree plus some point-to-point paths
	OSR	Propagation from failure to furthest node	Potentially all links (point-to-point paths)	All shortest paths (point-to-point paths)
	BRouter			
	LSOM			
	RBridges			
	Smart Bridge			
	SEATTLE			
	ARP-Path	Propagation from failure to source node plus one round-trip	Potentially all links (point-to-point paths)	Close to optimal
Turn Proh.	*Up/ Down*	Not specified	Potentially all links (depends on the turns allowed)	Better than single tree (depends on the turns allowed)
	TBTP	Not specified	Potentially all links (depends on the turns allowed, more than Up/Down)	Better than single tree (depends on the turns allowed, better than Up/Down)

* Count-to-infinity in the event of the Root failure

Table 4.2: Properties of related work techniques

Protocol		*Performance*			
		Path control	*Plug-n-play*	*Backwards compatible*	*End-node transparency*
IEEE 802.1	*STP*	Based on link cost	yes	yes	yes
	RSTP	Based on link cost	yes	yes	yes
	MSTP	Based on link cost	no	yes	yes
	SPB	Based on link cost but possible to extend to other metrics	yes	yes	yes
MSTP-based	*[38]*	Possible to extend to other metrics (paths computed offline)	Requires in-advance configuration of trees	Not specified	Not specified
	[39]			Not specified	Not specified
	[40]			No (remove STP)	yes
	[41]				yes
	[42]				yes
	Viking				yes
Routed Approaches	*STAR*	Based on link cost	yes (spanning tree as initial forwarding)	yes	yes
	OSR	Based on link cost	Requires in-advance configuration of paths	No (remove STP, no automatic learning, no broadcast)	yes
	BRouter	Based on link cost			
	LSOM	Based on link cost			
	RBridges	Based on link cost but possible to extend to other metrics		No (remove STP, no automatic learning)	yes
	Smart Bridge	Based on link cost		No (remove STP, no automatic learning, no broadcast)	yes
	SEATTLE	Based on link cost			yes
	ARP-Path	No (paths depend on ARP messages)			yes
Turn Proh	*Up/Down*	Not specified	Requires in-advance configuration of paths	No (remove STP, no automatic learning, no broadcast)	yes
	TBTP	Not specified			yes

Table 4.3: Operational aspects of related work techniques

Protocol		*Performance*			
		Table population	*Path selection nature*	*Active topology type*	*Protection/ restoration*
IEEE 802.1	*STP*	Automatic learning	Distributed Distance-Vector	Single tree	Restoration
	RSTP		Distributed Distance-Vector	Single tree	
	MSTP		Distributed Distance-Vector	Various trees	
	SPB		Distributed Link-State	One tree per node	
MSTP-based	*[38]*	Automatic Learning	Centralized computation (offline)	2 trees	Not specified
	[39]			Various trees	Protection
	[40]				
	[41]				
	[42]				
	Viking				
Routed Approaches	*STAR*	Additional tables configured by the routing technique	Distributed Distance-Vector	Single tree with additional point-to-point paths	Restoration
	OSR	Configured by routing technique	Distributed Distance-Vector	Point-to-point paths	
	BRouter		Distributed Distance-Vector	Point-to-point paths	
	LSOM		Distributed Link-State	Point-to-point paths	
	RBridges		Distributed Link-State	Point-to-point paths	
	Smart Bridge		Distributed Diffusing Computations	Point-to-point paths	
	SEATTLE		Distributed Link-State	Point-to-point paths	
	ARP-Path	Configured by the ARP-reply	ARP-based	Point-to-point paths (not deterministic)	
Turn Proh	*Up/Down*	Automatic learning	Distributed Diffusing Computations	Turn-prohibition	Restoration
	TBTP	Automatic learning	Not specified	Turn-prohibition	Not specified

4.2 MSTP-based extensions

Pre-configured backup trees Some proposals describe an offline computation of trees which is then assumed to be mapped onto the MSTP nodes for example by means of manual configuration by the network administrator.

This is the case in [38] where the authors propose the use of two edge disjoint trees (two trees that do not share any link). The proposal does not indicate why using two trees is an optimized solution. With the configuration of two trees, the authors focus on improving the resources utilization (active links) and providing better paths than the single tree. The MSTP framework is used to map the different VLANs into one of the two trees. In case of failure, and since the trees do not share any link, note that only one tree is affected and hence only those VLANs that were mapped to such tree loose connectivity. The proposal however does not describe any type of failure recovery. The trees are constructed offline and configured manually into the nodes by the network administrator. The authors consider the user traffic requests to optimize the creation of the trees and the assignment of VLANs to traffic flows [43]. The study only contemplates the topological aspects of constructing the edge disjoint trees but is has not been implemented. These aspects are evaluated numerically. While results show that a higher number of links is used than in the single tree (90% of links), the maximum number of hops stays 66% larger than the case of using optimal paths. In conclusion, this solution only improves the links utilization but it does not provide optimal path communications. In addition, recovery issues have not been analyzed and hence we cannot discern a conclusion on this aspect.

A similar proposal that also describes an offline algorithm to construct precomputed trees is described in [39]. The aim of the proposal is to reduce the recovery time. This solution proposes to construct a primary and a backup tree for each potential link failure. The tree switchover is then applied locally when a node detects a failure on a port. This solution requires that the network administrator configures in advance: the multiple MSTP active trees, the VLANs assigned to each one of the multiple trees, and the multiple backup trees that are used in case a link failure is detected. These tree and VLAN selections are done offline by means of an ILP (Integer Linear Programming) that minimizes the number of trees created to recover from all possible link failures. As in the previous case, the analysis provided is a numerical study based on topological aspects evaluating BW allocated in different parameters configuration. A complete convergence time analysis would require to evaluate the protocol, for example, in a simulation platform.

The proposal in [40] describes a technique based on the use of backup trees. It mainly aims at improving the recovery time while at the same time provides load balancing properties. There is not any limitation on the number of trees to construct and the authors do not specify any rule to decide how many (it is left to the administrator criteria). The protocol operation is based on switching between trees when a failure is detected. The decision to switch tree is taken on a per-frame basis in each one of the nodes. When a frame needs to be forwarded to a failed port, the bridge switches that data flow to the next tree (actually

changing the VLAN). The only requirement is that the trees must be ranked and only switching to a higher rank tree is allowed because switching back and forth between trees might create forwarding loops. One of the problems is that after a long time all flows might be transmitted over the last tree and a network reboot is needed. Another disadvantage is that switching between trees might create local loops because a frame traverses a link in one tree that it has already traversed in one of the previous trees. This might be a problem for keeping the symmetry requirement. The evaluation of the protocol is done by means of simulation and it shows an immediate single link failure recovery because of the local switchover to a higher ranked tree. In conclusion, while the use of backup trees clearly improves the resilience performance, requiring external tools to configure in advance all active topologies is an important limitation in Ethernet networking.

A different approach on how to use the backup trees to reduce the recovery time is presented in [41]. In this case each node stores a database with several trees. The particularity of these trees is that each one will be used if a concrete link failure occurs. That is, for each possible link failure each node stores which is the backup tree to use. This mechanism requires that the link failures be propagated to all nodes so they can switch to the corresponding tree. The calculation of the trees is also done offline before the network starts operating and it is described in [44]. It is based on an ILP optimization that creates as few trees as possible but ensuring that all possible link failures are covered. The evaluation is based on simulation and sub-50ms recoveries are achieved thanks to the use of the backup trees.

Solution at [42] also focuses on improving the recovery time by applying a protection mechanism based on the decisions taken by edge nodes. A set of predefined and static multiple trees distinguished by VLAN ID are set on the network. In doing so, alternative paths of different trees can be used in a failure situation. They implement an operation mode where all the set of trees is initially used by a single VLAN for each tree. The edge nodes need to be configured in advance so they know to which VLAN they assign each traffic flow. The authors describe in [45] the offline algorithm that constructs the best trees. The objective of the algorithm is to configure as many trees as needed in order to ensure at least one alternative disjoint path, in another tree, for each pair of edge nodes (this results in less than 10 in all their evaluations). The proposal implements a lightweight protocol based in keep-alive messages that detects a failure in the paths. Any single link failure detected by an edge node is broadcasted to the rest of edge nodes so they can also switch the VLAN ID and the tree. This results in removing an entire tree for every failure. When a failure is detected in one of the trees, the edge nodes switch the traffic from one VLAN to another in order to switch traffic to a different tree. The evaluation is based on a test-bed and results show that the recovery time only depends on the time to detect the failure and notify it to all edge nodes (from 10 to 60 milliseconds in a network of 8 nodes). However, it lacks plug-n-play property because the trees must be computed a priori and configured in the nodes before any communication. In addition, after several failures a network reboot might be needed if no trees are left for further recoveries.

Viking The Viking solution [46] uses the MSTP framework to deploy multiple trees and it is based on the application of an intense network monitoring together with a centralized computation of paths. We highlight it from the previous proposals because it provides a complete practical solution.

Its objective is to reduce the time required to recover from failures and provide load balancing. As in previous solutions, Viking is based on the use of primary and backup paths (or trees). Each one of the trees is assigned a different VLAN identifier and the end-hosts tag their frames, hence they select the tree, in order to apply load balancing or to recover from a failure. Note that Viking spans its action into the end-hosts and hence it is not transparent to the user.

The difference with the previous proposals is that Viking centralizes all the tree calculations and the management of notifications into a central node. This control point uses SNMP to remotely manage the MSTP instances of each network node and configure the path changes for example after a failure. In addition, it also notifies the end-hosts when to change the VLAN tag applied to the data frames.

The construction of the trees carried out in the central control. Viking assumes that the administrator manually enters the physical topology in the central node, so no topology acquisition operation is described. The algorithm used first computes two point-to-point paths between each pair of nodes (one primary and one backup). The different trees are then selected based on these paths, starting from the longest and adding one at each iteration. Finally, SNMP is used to remotely set the MSTP configuration in the network nodes.

The central control periodically receives statistics from each end node about the traffic load between each pair of nodes. This information is used by the centralized algorithm to recompute paths, hence trees, if necessary. At the event of a failure, the central point is also notified and makes the corresponding recomputations of the trees and tells the end-hosts to switch VLAN. The use of a centralized computation allows for high flexibility in terms of paths configuration. The use of backup paths allows for a quick recovery once the failure is detected and notified to the central node. However, a significant amount of overhead is introduced which in turn delay the failure recoveries to hundreds of milliseconds. In addition, it is not transparent to end-hosts as they must be aware of the VLAN where they belong.

4.3 Routed solutions

STAR The proposal in STAR [47] aims at improving the performance of a network already running the spanning tree protocol. It uses some of the links that the spanning tree protocol blocks, but keeping backwards compatibility with the legacy standard protocol. The idea is to deploy some STAR bridges in a bridged network together with spanning tree bridges and create an overlay logical topology only composed by the STAR bridges. The communications that do not start and terminate in STAR bridges still use the tree paths of the normal STP, so the global active topology is the original tree with the addition of some optimal

paths between the STAR bridges. Note that STAR bridges are devices that both operate the common STP and the additional STAR functionality.

STAR bridges start computing the spanning tree with the rest of common bridges. The STAR bridges then exchange special STAR frames (encapsulated in Ethernet data frames to bypass the STP bridges) to construct the overlay paths. A distance-vector protocol constructs the optimal paths between the STAR bridges. Note that these distances cannot be accurately computed because the intermediate nodes do not participate. Consequently, STAR algorithm takes this into consideration and only considers additional paths that can be accurately estimated.

A STAR bridge maintains the common Forwarding table of the STP operation (FD), and the additional tables Bridge Forwarding (BF) and Host Location (HL). BF table stores the paths between the STAR bridges and the HL table keeps the mapping between end-hosts and such STAR bridges. When a data frame is received by a STAR bridge, this looks up the destination address in the HL table. With this it obtains the association between the host and the corresponding STAR bridge and uses the BF table to determine the path to use. If the address is not found, the regular FD table is used (and flooded if not found either).

An advantage of STAR compared to other proposals is that it is backwards compatible with the legacy spanning tree bridges. Moreover, the STAR operation allows for overall improvements in terms of path optimality because it allows the use of some of the paths that STP blocks. However, still not all communications use shortest-paths. Note that if all nodes were STAR bridges then the solution is the same than in the following OSR or Brouter. The evaluation however focuses on message complexity and storage necessities and does not provide information on resource utilization of convergence time. However, in this last aspect we can expect large times because the mechanism that determine the different phases of the operation are based on expiration of timers.

OSR The Optimal-Suboptimal Routing (OSR) in [48] proposes a transparent bridge architecture that aims at increasing the number of links used and providing better paths than those in the standard spanning tree.

OSR substitutes the spanning tree protocol instance by a distance-vector routing protocol that constructs point-to-point paths. As described in 2.2, the nodes exchange topological information to build the tables that indicate which is the outgoing port to reach each one of the nodes (*destinationBridge-nextBridge* mapping table). In addition, each node collects the addresses of the end-hosts connected to it and disseminates this information to the rest of nodes. This way, every builds a table that contains which end-hosts are connected to each node (*host-bridge* mapping table). Note that implementation of such database can be done on two different tables or in the same merging all the information.

The utilization of both tables is the base of the forwarding function. When a frame is destined to a particular end-host, this is looked up in the host-bridge table. Once the destination bridge is known, the path to it is looked up in the bridge-bridge table and the frame is forwarded to the neighbor indicated.

The OSR evaluation focuses on delay and throughput obtained once the active paths are configured. Since more links are used and individual point-to-point paths are configured, OSR outperforms the single tree solutions (20% less delay in average). They do not implement the protocol that constructs the active topology so metrics such recovery time or protocol overhead are not taken into account.

Finally note that this type of proposal is not backwards compatible with legacy spanning tree protocols because (1) the tables are configured by the distance-vector routing protocol implemented instead of using the automatic bridge learning, and (2) the broadcast condition is removed because a frame directed to an unknown destination is dropped.

Brouter The Brouter proposal in [49] is very similar to OSR. It also proposes a transparent bridge architecture that increases the number of links used and provides better paths.

The path selection in the Brouter is also managed by a distance-vector routing protocol. The difference is that the Brouters match the location of the end-hosts to the LAN where they are connected. This implies that the Brouters use a host-LAN mapping table together with a LAN-LAN table. The LANs are identified by the individual Brouter identifiers and the port numbers (in order to decide which Brouter "owns" the LAN, the lowest value is selected). As in OSR, the distance-vector algorithm is used to construct the point-to-point path between LANs, and the host-LAN table to locate the end hosts is disseminated by each Brouter.

The Brouter evaluation provides a qualitative analysis about the amount of overhead (memory, CPU, and BW) present in the Brouters. It does not implement the protocol either so a detailed evaluation of the protocol performance is not available.

The Brouter proposal can be seen as an implementation of the routing philosophy idea into Ethernet. First, point-to-point paths are configured. And second, forwarding to LANs corresponds to forwarding to subnets (although masks cannot be applied because the lack of hierarchy in the MAC address space). While it is a transparent solution to end nodes, it requires configuration of the tables before starting the communication. In addition, backwards compatibility is not maintained because automatic bridge learning is not used and the broadcast operation is not used.

LSOM The Link State Over MAC (LSOM) protocol described in [50] proposes a bridging architecture that aims at improving paths and number of links used by using a link-state algorithm to construct the paths. LSOM substitutes the spanning tree protocol instance by a link-state routing protocol that constructs point-to-point paths. As described in section 2.2 the nodes exchange topological information to construct the physical topology and then apply a centralized algorithm to compute the paths.

LSOM is a solution oriented to scenarios where bridges are used in a small backbone network and routers are located in the edges connecting to the access. This way, routers set the limit of the bridged network in order to reduce the amount of addresses to be learnt (routers are seen as the end-hosts of the bridged network).

The link-state protocol is used to share the entire physical topology formed by the bridges. Unlike IP routing protocols, the bridge MAC addresses are used as identifiers of each node. As a link-state, bridges disseminate their connections to neighbors and then use Dijkstra [35] to compute the active paths between bridges. This operation populates the table that indicates the next hop to reach a particular destination Bridge. In addition, each bridge disseminates to the rest of nodes which end-hosts (routers) are connected to each bridge. This information together with the paths computed in the link-state algorithm form the routing tables used to forward data frames. Resiliency in LSOM is obtained using keep-alive messages sent between bridges every 10ms. When 3 in a row are not received, the link is considered failed and the link-state procedure updates and disseminates the new state of the link.

The LSOM evaluation focuses on delay and throughput once the topology is constructed (using optimal paths ring or trees) but there is no estimation of the time it takes to construct the paths and hence estimation of the convergence time.

RBridges RBridge proposal in [51] is focuses on enhancing layer 2 Bridges with some layer 3 functionalities. As previously described in the LSOM proposal, RBridges also use a link-state algorithm to construct the point-to-point paths of the active topology. This allows to operate in shortest path communications. The link-state mechanism is also used to distribute the end-hosts that are connected to each RBridge.

The difference with other routed-based approaches, RBridges maintain the Ethernet broadcast requirement in order to provide backwards compatibility. This is done by constructing, using the link-state mechanism, a spanning tree that is only used to forward frames to unknown destinations. One of the problems of using this spanning tree for broadcast communication is that it might have temporary loops while the topology information is being distributed. To solve this, RBridges add a hop counter in the data frame headers to avoid potential transient loops.

The proposal does not evaluate the protocol hence we cannot extract performance conclusions. Also, the details of the RBridges approach are being defined in the TRILL workgroup in the IETF at time of writing.

SmartBridge The technique described in [52] is also based on creating shortest paths between each pair of nodes using tables that map the end-hosts with the bridge where they are connected and tables that contain the next bridge in order to reach a destination bridge (routing-alike)

The difference is that the topology acquisition is done by means of a mechanism based on diffusing computations [53]. In this type of procedure, an initiator node sends a topology request to all its connected peers. In turn, the peers forward the request to their own neighbors so the topology request is actually flooded. When a node receives such request, it sends a topology reply with the list of its directly connected neighbors to the parent node that sent the request. Note that the topology replies with the list of connected neighbors are transmitted upwards to the initiator that originally sent the topology request. Since the initiator node receives all the replies from all nodes, it actually holds all the lists of neighbors and hence it is aware of the entire topology. At his point the initiator disseminates the physical topology to all nodes so they can compute the paths. In summary, a diffusing computation can be easily seen as a 3-step procedure where (1) the initiator requests the connections to all nodes, (2) the initiator receives such information, and (3) the initiator disseminates the entire list of connections so every node becomes aware of the physical topology. This topology acquisition mechanism is triggered at start-up and when a node detects a failure.

The difference with the link-state protocols is that the diffusing computation technique allows the initiator to detect when the process has terminated so the nodes can start sending data using the computed active topology. This is an important issue to consider as detecting the termination of the topology construction mechanism is one of the key points in distributed systems. Actually, the original STP lacks of a reliable mechanism like this one and implements over-estimated timers to decide when the algorithm has converged and start transmitting data. Differently, RSTP introduces the proposal-agreement handshake to locally detect the termination of the algorithm. While the mechanism in RSTP does not indicate when the protocol has totally converged, it allows to start sending data as soon as the neighbors confirm the handshake (see section 6.2 for a more detailed explanation).

Evaluations show that the diffusing computations provides recovery times of $40msec$ for networks (in a 20 nodes network). In addition, the knowledge of the entire topology in each node allows SmartBridges to operate with shortest path communication. However, it is not backwards compatible with original Ethernet Bridging.

SEATLLE SEATTLE [54] implements the topology acquisition using a link-state protocol. The difference with the other link-state techniques is that the resolution of the host location (that is to which edge node a host is connected) is not done by means of link-state advertisements. Instead, the information that maps each end-host with the connected bridge is distributed among all network nodes (hence avoiding all nodes storing the entire information). When a packet needs to be forwarded, the receiving node A use a consistent hash function with the host destination as the key. This calculation provides the network bridge B that contains the edge node C where the host is connected. A then sends the packet to B, and this last forwards it to C. In order to avoid this additional overhead in further forwarding, A is notified by B that the host is connected to

C. In later forwarding A directly sends the frames to C.

The objective of SEATTLE is to increase Ethernet networks scalability by means of removing the flooding of unknown destination hosts. The use of the distributed hash tables improves the amount of overhead due to flooding of data packets and reduces the forwarding table sizes compared to the flat approach of original Ethernet. The study does not focus on the evaluation of the link-state algorithm to construct the active topology and it is assumed already available.

ARP-Path A different approach is proposed in [55] where the ARP messages are used to construct the paths substituting the bridge learning functionality. ARP-Path aims at configuring better paths than with a single tree while reducing complexity of other proposals. When an end-node wants to send a message to an unknown destination, it first floods an ARP request asking for that particular destination. When this request is successfully received by the destination, this replies back an ARP reply that is used to really select the path that will communicate source and destination introducing the corresponding entry in the forwarding tables. The different bridges that this ARP request traverses actually learn the port where the destination is located.

Note that this approach results in a connection-oriented solution because the path is created using the request and reply of the ARP messages and both end-nodes agree on the connectivity establishment. Also, the recovery is based on a notification to the source bridge so it can re-issue the ARP-based mechanism and re-construct the path. The bridge that detects the failure is in charge of notifying the rest of bridges in the path to the source so they remove the learnt ports. When this message reaches the source node, this floods a new ARP request and the path is reconstructed.

The evaluation of the proposal focuses on studying latency and allocated bandwidth measurements. Delay results with ARP-Path show an improvement of 60% compared to using a single tree. In terms of throughput, the single tree network saturates at 16% of the maximum load while in ARP-Path it saturates at 32%.

Although ARP-Path avoids the broadcast forwarding of classical Ethernet Bridging, the main shortcoming of this approach is that the constructed paths depend on how the first ARP message reaches each node. This results into non-deterministic routes that cannot be managed as they depend as instant aspects such as network congestion and on an algorithm not designed to establish paths.

4.4 Turn prohibition

Up/Down protocol The Up/Down algorithm was originally designed for the Autonet high-speed network [56] [57]. The objective is still to avoid the creation of loops, but instead of stopping the communication in some links (as a tree-based solution applies) this technique selects a set of turns that are prohibited. The algorithm that selects which turns are allowed uses a directed spanning tree that is constructed in a way that all links point to the Root of the tree. This results

in each link having an "up" direction (to the Root) and a "down" direction (to the leaves). In order to avoid loops, a message that is forwarded node after node is only allowed to change the direction once: first going up, and then going down.

The Autonet nodes need to know the entire physical topology in order to locally compute the routing tables based on the Up/Down rules. Autonet also provides a topology acquisition mechanism similar to SmartBridge based on diffusing computations [53]. A spanning tree using [22] is first constructed, and then each node uses it to disseminate their neighbors (send this information up to the Root). When the Root of the tree receives the adjacencies of each node, it combines all this information and disseminates it to all the nodes so they are aware of all the physical topology. At this point, nodes locally calculate the routes between all nodes taking into account the turn-prohibition rules. While this process is being operated the data communication is not allowed as inconsistent views of the topology might end up in forwarding loops. Once the tables are constructed, the message forwarding is based on the entries in the tables populated by the routing algorithm.

The main advantage of this type of protocols is that they use a higher number of links than deploying a single tree. However, since all messages are only allowed first to go up and then go down, there are still bottlenecks in the Root proximities and paths are not optimal. The drawbacks are that the data communication is cut while the topology construction is being done (the proposal does not provide numerical evaluations on this aspect, and that the Up/Down algorithm described does not have any upper bound on the amount of prohibited links.

These two disadvantages have triggered some proposals that aim at improving the Autonet. The protocol in [58] describes a way to keep the data communication flowing while the construction of the tables is being done. It still uses the Up/Down algorithm and the same topology acquisition mechanism, but the update of the routing tables is different. It is based on a sequenced partial update of the tables confirmed by the immediate neighbors (so the neighbors really confirm what to update and when to do it). However, the changes proposed imply that the constructed topology depends on the order of how bridges appear and hence it is not deterministic.

Another enhancement is the Hierarchical Up/Down Routing Protocol (HURP) described in [59]. This solution decreases the amount of prohibited turns and does not require that nodes have the entire physical topology. The hierarchical addresses are configured by an additional protocol that allows additional turns ensuring a loop-free delivery under some particular conditions (this reduces the amount of prohibited turns to 20%). In addition, the construction of routes is not centralized because a distributed distance-vector algorithm is used to exchange the allowed routes (similar to distance-vector solutions in section but taking into account the prohibited turns).

Tree-based Turn-Prohibition (TBTP) The TBTP [60] is based on the theoretical algorithm described in [61]. It reduces the amount of prohibited turns compared to Up/Down algorithms, and it bounds its maximum value to 50%.

Note that the more turns are allowed, a better resource utilization is achieved. TBTP is a centralized algorithm that is given the network topology and a spanning tree and it computes the set of prohibited turns (as in Autonet). In comparison to Up/Down, TBTP is a more complex calculation that aims at optimizing the number of prohibited turns while constructing optimal paths as much as possible. Since the algorithm is locally computed in all nodes, the architecture requires that each element knows the physical topology. The topology acquisition mechanism is not provided but assumed as known.

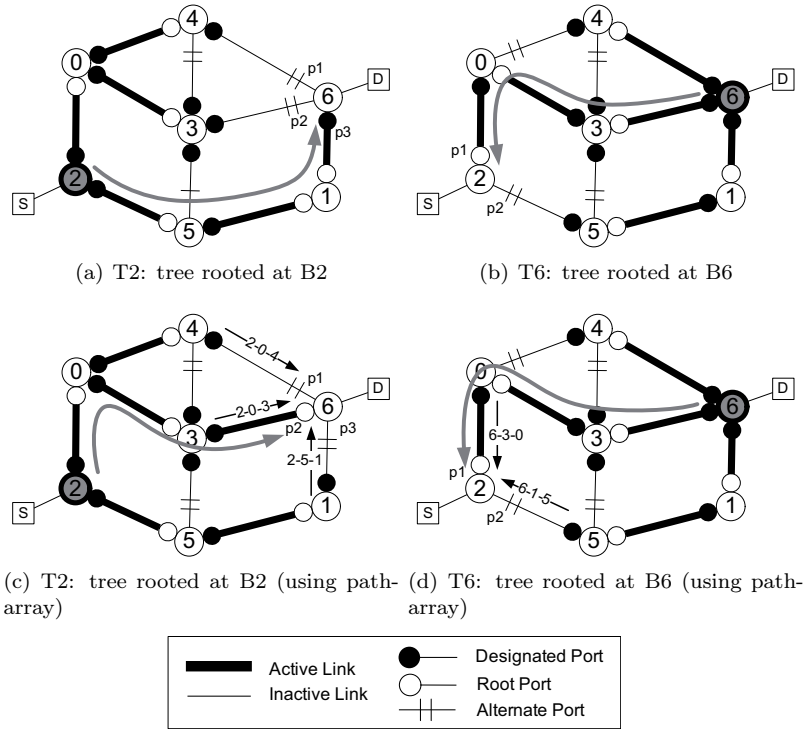
Using a single tree as the active topology is still inefficient in terms of resources utilization (not all links are used) and path optimality (only the Root observes shortest-paths to all nodes). For this reason the IEEE 802.1 standardization body is currently working on the update of the family of spanning tree protocols and is introducing Shortest Path Bridging (SPB) [17]. Note that this SPB description is presented in a standalone chapter, apart from the other proposals of the related work, because it represents the current standardization efforts that directly compares to our shortest path proposal RSTP-SP described in chapter 8.

The idea of SPB is to extend the active topology into one tree rooted at each node of the network. Note that if each node uses its own tree to introduce its own data traffic, shortest-path communications are achieved between each Root and the rest of nodes. Figures 5.1(a) and 5.1(b) show two trees rooted at $B2$ and $B6$ ($T2$ and $T6$ respectively). The traffic introduced by $B2$ is forwarded using $T2$. Note how all the communications follow a shortest-path between $B2$ and any destination bridge. Similarly, $B6$'s traffic is transmitted using $T6$ and therefore also using shortest-paths. If each node uses its own tree, all communications follow the optimal paths. In addition, this active topology provides an increase on network resources utilization because the links are active either in one tree or the others.

This chapter describes several aspects related to the design of SPB. First, deploying one tree per node introduces an additional concern to this new bridging because it is not trivial to configure symmetrical paths between nodes (section 5.1 elaborates on this topic). And second, sections 5.2 and 5.3 describes of the SPB protocol and provides operational examples.

5.1 The symmetry challenge

The active topology created by the spanning tree protocols determines the paths that are learnt by the learning function and implemented in the forwarding tables. In the single tree case, the unique active topology forces the bridge to learn bidirectional paths and, in turn, ensures symmetry. Differently, in SPB the communication paths are selected as if they were unidirectional because a different tree is used in each direction. The main challenge is that SPB needs a careful selection of trees due to the symmetry requirement of the fundamental bridge learning operation.



(a) T2: tree rooted at B2

(b) T6: tree rooted at B6

(c) T2: tree rooted at B2 (using path-array)

(d) T6: tree rooted at B6 (using path-array)

Figure 5.1: Trees rooted at B2 and B6 become symmetrical if the path-array tie-breaking is applied.

In the example of figure 5.1(a) the traffic generated at host S, injected by $B2$, is forwarded to destination D connected to $B6$ using the tree rooted at $B2$ ($T2$). And similarly in 5.1(b), traffic from D to S is injected by $B6$ and forwarded using the tree $T6$. Assuming that all forwarding tables are empty, $B2$ floods the frame from S to D to all ports where the tree $T2$ is active (both $p1$ and $p2$). This frame is flooded in each bridge following $T2$ and eventually reaches $p3$ of $B6$. Therefore, $B6$ learns that it can reach S through $p3$. If $B6$ now sends a frame from D to S, it does not flood it because it has already learnt that it can reach S through $p3$. Note that this frame is forwarded using $T6$, to $B1$ and to $B5$, but its transmission is stopped because of the blocked links in $B5$. Hence it will never reach S. Similarly, $B6$ uses $T6$ to flood a frame from D to S and $B2$ learns it in port $p1$. A frame to D injected by $B2$, hence using $T2$, will not reach $B1$ as it will be discarded after $B3$.

Since the symmetry condition needs to be met, the branch in $T2$ from $B2$ to $B6$ must be the same than the branch in $T6$ from $B6$ to $B2$. In this example the trees $T2$ and $T6$ are not symmetrical and the bridge learning and forwarding functions do not properly work as seen in the previous example.

Any shortest-algorithm can be used to compute the trees that compose the active topology because the objective is to select the shortest-paths from all nodes to each Root. However, a common implementation of the selected algorithm might easily lead to non-symmetrical trees. The reason is that when an algorithm needs to select among multiple shortest paths, the most common applied tie-breaking rule selects the path where the immediate next hop has the lower identifier. In the example of figure 5.1(a), $B6$ selects $B1$ instead of $B3$ or $B4$ because $1<3<4$. Similarly in 5.1(b), $B2$ selects $B0$ because $0<5$. The path between $B2$ and $B6$ is not symmetrical because the path selection depends on this local information (immediate next hop), which results in different decisions at different points of the network. Note that the selection of the best next hop that leads to each Root is actually the selection of the Root port for each tree.

In order to ensure global consistency, it is a matter of extending the decision elements and use the complete path from the Root: the *path-array*. Instead of looking at the parent identifier to select among multiple routes, this approach compares the entire array of each suitable path. Once the algorithm computes all the eligible shortest-paths, the tie-breaking rule using the path-array is applied. The path-array of each shortest-path is first sorted from lowest to highest, and then the elements are compared one by one. If the element is the same in both path-arrays, the next element is compared. Otherwise, the path-array with the lowest element is considered better and hence selected.

In the example of figure 5.1(c), $B6$ decides between the path-array in $p1$ (2-0-4), in $p2$ (2-0-3) and in $p3$ (2-5-1). Once sorted, the arrays become 0-2-4, 0-2-3 and 1-2-5, respectively. The path-array in $p3$ is discarded because of the highest first element ($0<1$) and among $p1$ and $p2$, the latter is considered better because $3<4$ at third position. Similarly in figure 5.1(d), $B2$ decides between 6-3-0 in p1 and 6-1-5 in $p2$. Once sorted, the arrays become 0-3-6 and 1-5-6. $B2$ considers $p3$ better because $0<1$ in the first element. Since both selected paths are the same but in opposite direction, the path between $B2$ and $B6$ are the same.

The calculation of the symmetrical trees using the path-array is straightforward in a framework where a centralized algorithm is used to compute the paths. Note that the central node that applies the algorithm knows all the topology and hence it can easily compute the eligible shortest-paths and then apply the tie-breaking rule. However, the SPB requires a distributed protocol that makes such calculations. This is one of the reasons why the SPB introduces the link-state approach moving away from the distance-vector algorithms of the current spanning tree protocols. As in the centralized protocols, the flexibility that the link-state approach introduces also allows for an easy solution to compute the path-array in a distributed framework. Following sections describe the link-state approach proposed by 802.1 to construct the SPB multiple tree active topology.

5.2 SPB protocol operation

This section provides a description of the SPB protocol proposed by the IEEE 802.1 workgroup. What SPB really does is to implement the link-state proto-

Table 5.1: Bridge Variables

Name	*Description*
BridgeID	Unique identifier of a bridge.
ADJ [n]	Array that contains the list of adjacencies received from each node (e.g. ADJ[3] contains the list of adjacencies of node 3). Note that each ADJ[i] is actually another array. Every node stores all adjacencies received because the complete list represents the entire physical topology.
seq [n]	The adjacencies received from each node have an associated sequence number so the protocol can easily distinguish between old and fresh information. A received list of adjacencies, with a higher sequence number than the last received, always updates the local version, otherwise it is discarded.

Table 5.2: Port Variables

Name	*Description*
Role	The port roles are the same as in the spanning tree protocols (Root, Designated, Alternate). The only difference is that SPB uses a link-state technique to elect them.
State	The state of the ports (Forwarding, Discarding) also determines which ports actually forward or block data frames. As in the spanning tree protocols, Root and Designated ports are Forwarding and Alternate ports are Discarding.

col IS-IS [36] as a base and introduce the SPB particularities as extensions. As already mentioned, one of the advantages of the link-state protocols is their flexibility and ease of extendibility, and IS-IS was originally designed aiming at this direction.

The objective of the link-state protocol used in SPB is to construct the trees that form the active topology. This actually means setting the port roles (Root, Designated, Alternate) and the port states (Forwarding, Discarding) of each one of the trees that are constructed. Note that, as in the spanning tree protocols, the link-state technique is not used to populate the forwarding tables as this is still left to the automatic bridge learning functionality. Therefore, SPB uses a link-state solution to only substitute the operation of the spanning tree protocols: constructing the active topology.

5.2.1 Bridge and port variables

The state that the SPB protocol implements can be divided in two areas: (1) those used by the link-state technique that disseminates the physical topology and locally compute the trees, and (2) those that relate to the construction of the trees from the bridging perspective.

In the first case, each node stores the necessary data structures to store the link-state information that is received from other nodes. The variables to manage the link-state operation are included in table 5.1. Note that these variables are all stored at bridge level as there is no real port information in the link-state

protocols. First, a unique node identifier is needed in each device: the *BridgeID*. Second, the topological information is stored in the form of adjacencies. An *adjacency* is a confirmed physical connection between two nodes. Each node stores a list of sequences of adjacencies ($ADJ[n]$). That is, each node keeps an array with as many entries as nodes in the network, and where each entry is a list of adjacencies that represent the neighbors of the node that corresponds to that particular array entry. Note that each one of the array entries contains the connectivity that each node disseminates. Therefore this adjacencies list can be seen as the compilation of received information from the rest of nodes. And third, each one of the entries also has an associated sequence number ($seq[n]$) to identify whether a received adjacency updates the stored information.

And in the second, the set of port variables include the port role and the port state as described in table 5.2. These two variables are what really determines the shape of the tree from the bridging perspective (whether a port is active or not). Note that each port stores one role and one state for each one of the trees. The difference with the spanning tree protocols is that these are elected with the computations done by the link-state protocol.

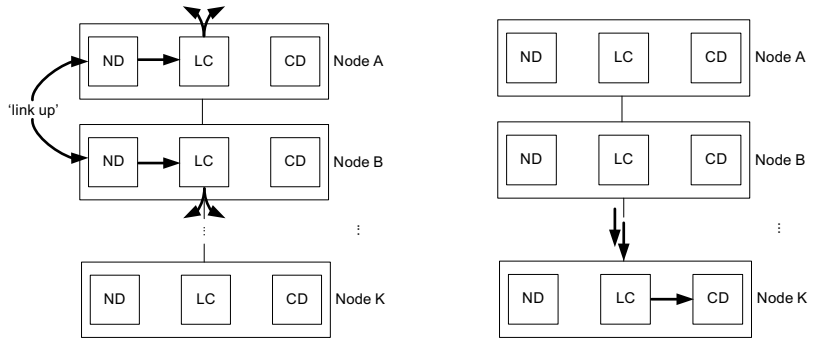
5.2.2 Construction of the multiple trees

The protocol functionality is composed by three different operations: the *neighbor discovery*, the *connectivity dissemination* and the *local computation* of paths. Figure 5.2 shows a diagram with the relationship of these three operations.

The neighbor discovery is triggered by a node when a link is added and aims at notifying to the opposite neighbor about the existence of the first (see how the signal "link up" calls the neighbor discovery operations in the neighbor A and B in figure 5.2(a)). Every time a node discovers a neighbor, it realizes about the new available connection and the connectivity dissemination is executed to announce the new 'link state' to other nodes. In the example, A and B execute their connectivity dissemination and flood the new available link. The disseminated 'link state' information is flooded and hence all other nodes eventually received it (figure 5.2(b)). Receiving fresh information about the state of a link implies that the database of adjacencies is updates and hence the physical topology changes. The local computation of trees is executed to revise the paths and update the port roles and port states accordingly.

At reception of the disseminated information including updated available connections, nodes construct the physical topology and run the local computation of paths, hence construct the trees. Node I in the diagram represents any other node that receives the link state information and recomputes the paths. Also observe that the trees are also recomputed by A and B when the connectivity dissemination is triggered because of the new link.

Note that although the three operations are independent, they are linked to each other and there is not any condition that determines the end of the operation as the nodes are always awaiting for a notification of a new link that triggers the process again. The details of these operations are summarized in the pseudo-code in figure 5.4 and described next.



(a) Neighbor Discovery triggers the Connectivity Dissemination

(b) Connectivity Dissemination triggers the Local Computation

Figure 5.2: Relationship between the operations of the SPB protocol

Neighbor Discovery At network start-up, the nodes are not aware of any connection hence all variables are initially cleared (see the event *Turn-on bridge* in block A of figure 5.4). Turning on a bridge implies that the ports of the links are already available, which triggers the neighbor discovery function in each bridge. All nodes send *Hello Messages* to all ports in order to notify to the neighbor that they are already available. Table 5.3 includes the description of the frame format of a Hello message. A part from many fields inherited from IS-IS that focus on managing the protocol and identifying the frame, the Hello Messages basically include the *BridgeID* of the transmitter, in the *SourceID* field, so the immediate neighbor realizes which bridge is at the other side of the link.

The reception of a Hello message is processed in the receiving node as the event *Reception of Hello Message* indicates in block B of the pseudo-code in figure 5.4. Since receiving a Hello message indicates the confirmation of the connectivity between the neighbors, the local node adds this adjacency into its own list (that is, the receiver B includes the $M - B$ connection into ADJ[B]). The diagram in figure 5.3 shows an example of the exchanged messages and processing events that occur in a cold-start scenario in a sub-set of nodes of the example topology. The thin arrows that are sent at t_0 are the Hello messages and the number in brackets indicates the BridgeID of the transmitter. For example, at t_1 $B0$ receives the Hello Messages from $B3$. At this instant $B0$ adds the connection $B0 - B3$ to its own ADJ[0].

After a Hello reception, the receiver B floods the new information about its own adjacencies, ADJ[B], to all neighbors. The adjacencies are distributed encoding them into *Link State PDU* (LSP) as shown in the frame format described in table 5.4. As in the Hello message case, this format is inherited from IS-IS and most of the field are used to control versioning and type of frames. The essential fields in the LSP frames are the *LSP ID* that includes the identifier of the node that originates such LSP, the *Sequence Number*, and the array of adjacencies including the *Neighbor ID* and the corresponding cost *Metric*. For example, observe in the message diagram of figure 5.3 how $B0$ and $B6$ transmit

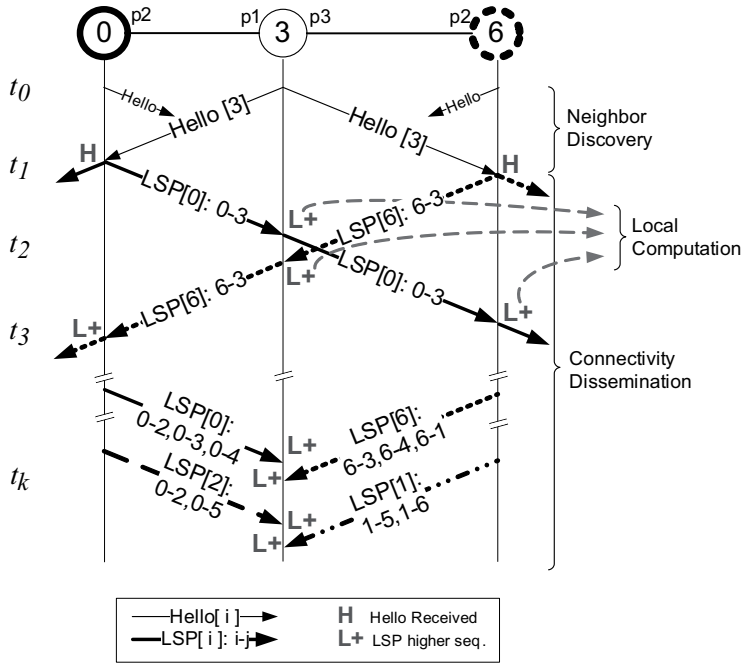


Figure 5.3: Example of a cold-start in SPB link-state protocol.

their LSPs at t_1 (LSP[0]:0-3 and LSP[6]:6-3 respectively).

Connectivity Dissemination This operation is triggered every time an LSP is received, as detailed in the event *Reception of LSP* in block C of figure 5.4, where node B receives and processes the adjacencies of node M. The receiver first compares the sequence number of the incoming LSP with the last sequence number received. If the received is higher (lines 1 to 5 of block C), the LSP[M] is considered to carry fresh information and the adjacencies of node M are stored in the local database of node B (ADJ[M]). This received LSP is then forwarded through all the ports except the incoming, so its flooding continues and ensures that the LSP reaches all nodes.

If the received sequence is equal to the stored (lines 6 to 7 of block C), nothing is done because it is considered repeated information and it does not update the topology database. If the received sequence is lower (lines 8 to 9), it refers to an old topology update and it is discarded. In this last case, the adjacencies of node M stored in node B (ADJ[M]) are sent back to the port where the old LSP was received so the neighbor corrects its database.

In the example of figure 5.3 the LSPs are represented by thick arrows with the owner's identifier between brackets and the adjacencies included written as $i - j$ (node i connected to node j). Note that only LSPs from nodes $B0$ and $B6$ are shown for simplicity reasons and different line styles are applied to LSP from different nodes. Observe how the dissemination of LSPs starts at t_1 after

the reception of the Hello messages. The propagation of LSPs advances node after node until reaching all network elements. For example, $B3$ receives LSP[0] at t_2 and forwards it to $B6$ that receives it at t_3, and so on. Also note that, for instance, the LSP[0] received by $B3$ at t_1 only includes the adjacency $0-3$ but a more updated version of LSP[0] is received at t_k with already all $B0$ neighbors (0-1, 0-3, 0-4).

Table 5.3: Hello Message Frame Format

Name			*Description*	*Bytes*
Ethernet encaps.	*SA*		Management MAC address of the bridge	6
	DA		Reserved address to indicate "all bridges" (01:80:C2:00:00:00). It is a broadcast protocol and messages are not sent to a particular destination and are consumed at the next bridge hop and only live in the link they are produced.	6
	Length/ Type		Indicates total length of the frame.	2
	LLC		Indicates the bridge client that processes the received frame and delivers it to the SPB client.	3
IS-IS PDU	*IS-IS header*	*Protocol identifier*	Fields defined by IS-IS used to manage the identification of the IS-IS frame.	1
		Header Length		1
		Protocol Version		1
		PDU type		1
		PDU version		1
		Reserved		1
		Maximum Area		1
	Hello Fields	SourceID	The unique identifier of the transmitter node is encoded in the SourceID field so the receiver realizes the identifier of its neighbor.	8
		Circuit Type	Additional fields defined by IS-IS used to manage the identification of the frame.	1
		Holding Time		2
		PDU length		2
		Local Circuit ID		1
		Extension TLVs		-
Frame Check Sequence				4

Table 5.4: LSP Frame Format

Name			*Description*	*Bytes*
Ethernet encaps.		*SA*	Management MAC address of the bridge	6
		DA	Reserved address to indicate "all bridges" (01:80:C2:00:00:00). It is a broadcast protocol and messages are not sent to a particular destination and are consumed at the next bridge hop and only live in the link they are produced.	6
		Length/ Type	Indicates total length of the frame.	2
		LLC	Indicates the bridge client that processes the received frame and delivers it to the SPB client.	3
IS-IS PDU	*IS-IS header*	*Protocol identifier*	Fields defined by IS-IS used to manage the identification of the IS-IS frame.	1
		Header Length		1
		Protocol Version		1
		PDU type		1
		PDU version		1
		Reserved		1
		Maximum Area		1
	LSP: header	*PDU length*	Fields defined by IS-IS used to manage the identification of the LSP information.	2
		Lifetime		2
		Checksum		4
		Flags		2
		TLV Type		1
		TLV Length		1
		LSP ID	Identifier of the bridge that originates this LSP. This node is actually announcing its adjacencies with this LSP.	8
		Sequence	Sequence number of the LSP included in the message.	4
	LSP: adjacencies	*Neighbor ID*	First adjacency of the LSP.	8
		Metric		3
		Sub-TLVs		1
		...	...	...
		Neighbor ID	Last adjacency of the LSP.	8
		Metric		3
		Sub-TLV		1
Frame Check Sequence				4

Local Computation of Paths When a node receives an LSP with new adjacencies (an LSP with a higher sequence number), it updates the physical topology and individually computes the active trees (line 4 in block C and subroutine *ComputeTrees()* in block E of figure 5.4). Since the computation is done locally, any technique that is able to compute shortest paths can be used as long as it is the same in all nodes. The common option is to apply Dijkstra algorithm [35] to compute the shortest-paths from a concrete node to the rest (note this creates a shortest-path tree rooted at the given node). Therefore, repeating the same calculation for each Root node results in all shortest-path trees. Note that the path-array tie-breaking rule described in section 5.1 is used to select among multiple equal shortest-paths.

This path information is then translated into port roles (Root, Alternate, Designated). Once the active trees are locally calculated, it is straightforward to determine the port roles. For each tree, the port in each node that provides the closest path to the Root is selected as Root port. Among the rest, the port becomes an Alternate if the neighbor is closer to the Root. Otherwise it is assigned a Designated role. Note that these distance comparisons are locally done by each node, so each one makes the same decisions.

The local computations are represented in the diagram of figure 5.3 with the sign "$L+$" (reception of an LSP with a higher sequence). Note that these are due to the receptions from the first LSP[0] received by $B3$, at t_2, to the last one received at t_k. This is because the distributed nodes do not know at which instant the adjacencies received are definitive or not. Therefore the nodes keep compute the trees every-time, which might result into transient configurations because different nodes might have inconsistent views of the topology. For example, $B3$ at t_2 has only received the LSPs from $B0$ and $B5$, and therefore the trees that $B3$ computes are those on a topology with these 3 nodes. On the contrary, the trees computations that $B3$ executes at t_k (assuming all LSPs already received) are done using the complete physical topology. Since the process is based on the propagation of the LSPs, the convergence time of SPB in a cold start scenario can easily be characterized as: $CT_{SPB} = (Diam + 1) \times t_{hop}$ (where $Diam$ is the diameter of the physical topology, the $+1$ is because of the initial Hello Messages, and t_{hop} is the propagation delay of a dine hop).

Different nodes might configure different trees which can result into the appearance of forwarding loops if the nodes start communicating data. In order to avoid this, there is the need for an additional port activation mechanism. The idea is to only transition Root and Designated ports to Forwarding if the local node and the neighbor have the same topology view. This small handshake protocol at most only requires the exchange of two messages between neighbors so as to confirm the transition of ports to Forwarding. The details of the additional protocol are still being defined in the standard body. For this reason in the present work we implement an ideal mechanism for the evaluation in the simulation platform. A Root or Designated port transitions to Forwarding if the neighbor in that link has the same topology view than the local node. This checking can easily be made in a simulation where information is available globally, but a practical solution requires the exchange of messages mentioned.

A. Turn-on Bridge	***/* It is triggered at each node at network start-up. All nodes are initially configured as the only device in the network */***
1. Clear all ADJ[n]	*/* At network start-up, each node is only aware of itself, therefore the link-state database containing adjacencies of other nodes is empty */*
2. Set all ports of the own tree to Designated role and Discarding state	*/* Each node is the Root of its own tree; hence all ports of this tree are configured as Designated. The state is initialized to Discarding waiting for a future confirmation that transitions them to Forwarding */*
3. Set all ports of the the other trees to an undefined role and Discarding state	*/* At the beginning, the nodes are not aware of their position in the other trees, so all ports are temporarily assigned to an undefined role and Discarding state */*
4. Send Hello messages to all ports	*/* Turning on a bridge results in starting the neighbor discovery mechanism so neighbors become aware of the existence of such bridge */*

B. Hello message from node M received in bridge B	***/* The reception of a Hello message from a neighbor confirms the connectivity between the local node and the neighbor */***
1. Add B-M adjacency to ADJ[B]	*/* In practice, this reception results in the addition of the connection between them into the adjacencies of the local node (ADJ[B]) */*
2. Increase seq[B]	*/* A change in the local adjacencies ADJ[B] implies an increase in the corresponding sequence number so it is considered fresh information when received by other nodes */*
3. Send ADJ[B] to all ports	*/* The updated local adjacencies list is disseminated so all other nodes realize about the new connection between B and M */*

C. LSP[M] message received in bridge B	***/* The reception, processing and dissemination of an LSP are the actions that compose the connectivity dissemination */***
1. if (sequence of LSP[M] > seq[M])	*/* A received LSP of node M (LSP[M]) with a higher sequence than the stored in seq[M] represents fresh information that updates the last received and hence it is stored in the local copy of the adjacency of M (ADJ[M]) */*
2. ADJ[M] = LSP[M]	
3. seq[M] = sequence of LSP[M]	
4. ComputeTrees()	*/* A change in the list of adjacencies means a change in the physical topology, so the trees must be reconstructed to match the paths to the new links */*
5. Send ADJ[M] to all ports except p	*/* The flooding of this LSP continues to other nodes */*
6. if (sequence of LSP[M] == seq[M])	*/* The reception of the same sequence number than the stored indicates that this LSP has already been received and processed */*
7. -	
8. if (sequence of LSP[M] < seq[M])	*/* The reception of a lower sequence number than the stored really means that the neighbor that forwarded has not updated information, so the local node sends back the own version of this LSP stored in ADJ[M] */*
9. Send ADJ[M] to port p	

D. Failure detection in port p of bridge B	***/* A port failure is interpreted as a loss of connectivity between the nodes that connect the link */***
1. Neigh = neighbor connected to B through port p	*/* Local variable that stores the identifier of the node connected at the other side of the failed port */*
2. Remove adjacency B-Neigh from ADJ[B]	*/* A port failure is interpreted as a loss of connectivity between nodes B and Neigh. This is why the local adjacency list is updated removing the B-Neigh connection and the sequence number is incremented */*
3. Increase seq[B]	
4. ComputeTrees()	*/* A change in the list of adjacencies means a change in the physical topology, so the trees must be reconstructed to match the paths to the topology without the failed link */*
5. Send ADJ[B] to all ports	*/* The new information in the local adjacency is disseminated so all other nodes realize about the lack of connectivity between B and Neigh */*

E. ComputeTrees()	***/* The computation of trees is triggered when there is an update in the database of adjacencies */***
1. Run Dijsktra for each Root node	*/* Since the objective is to create one tree rooted at each node, Dijkstra is executed as many times as nodes. Each one of the executions builds the shortest-path tree to one of the Roots */*
2. Apply path-array tie-breaking in equal-cost shortest-paths	*/* If the calculation of Dijkstra results in multiple shortest paths, the path-array tie-breaking is used to select a single path so the bridging symmetry requirement is met */*
3. Select port roles	*/* Once the unique shortest-paths to each Roots are identified, port roles are selected */*

Figure 5.4: Pseudo-code of the SPB operation

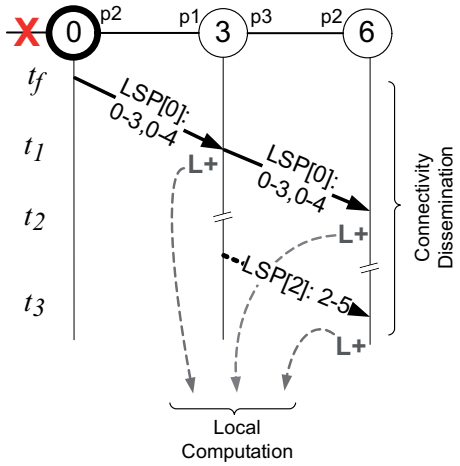


Figure 5.5: Example of a link failure recovery in SPB link-state protocol.

5.3 Failure recovery

Another event that triggers a change of topological database is the detection of a link failure. If a node detects a failure in one of the ports the connection to the neighbor becomes unavailable. The operation is shown in the event *Failure Detection* of block D of figure 5.4. The node removes the corresponding adjacency from its own ADJ because the link is not available anymore and hence its connections to the neighbors have changed. Since the adjacencies have been updated, which means a change in the physical topology, the node recomputes the trees so the paths match the new topology without the failed link. The new information of B's adjacencies needs to be distributed so other nodes also realize about this change. The bridge detecting the failure disseminates its new ADJ flooding an LSP.

Figure 5.5 shows an example where the link between $B0$ and $B2$ fails. $B0$ detects the failure at t_f and realizes that its connection to $B2$ is not available. $B0$ then removes the 0-2 connection from its own ADJ[0] and the LSP is sent to all the neighbors. Note this LSP only includes the connections [0-3,0-4] as $B0$ drops 0-2. The rest of nodes receive the flooded fresh LSP and consequently recompute the trees now considering a physical topology without the 0-2 link. When this LSP propagation reaches all network elements, every node already has the new topology.

The recovery of a node failure does not differ much from the single link case in terms of protocol operation. A node failure can be seen as a failure of each one of the links connected to the failed node. Therefore each one of the neighbors detects the failure and floods a new version of its LSP (as $B0$ and $B2$ do in previous example). Eventually, all nodes in the network receive the new LSPs of the affected nodes and configure the trees based on the new physical topology.

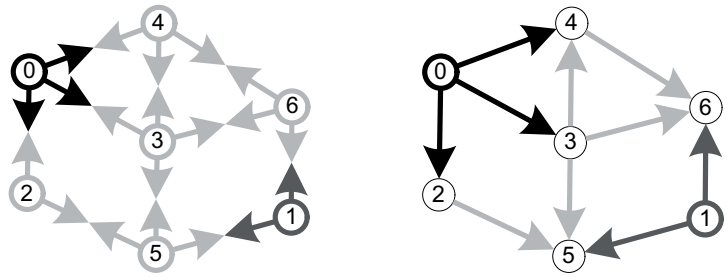
This chapter includes the study of the RSTP behavior when it constructs the initial tree at network start-up. The objective is to comprehend the evolution of the distributed operation in order to understand performance aspects such as the time that the protocol takes to construct the tree. Therefore, this study focuses more on analyzing the behavior of the protocol rather than the operation (which is described in chapter 3).

To the best of our knowledge, observing the protocol behavior from this aspect is a novel contribution. Most RSTP studies focus on performance evaluations that analyze the global protocol performance in different scenarios. References [27] to [29] present evaluations in small topologies. They use the OPNET [62] simulations and validate the results with small test-beds. An interesting analysis is included in [30], which focuses on the characterization of the processing delays of real equipment in small test-beds. Depending on the device, delays between $1,33ms$ and $12ms$ are observed.

The chapter is organized as follows. First, section 6.1 contains a comprehensive behavior analysis describing the propagation effect of the tree construction. Second, section 6.2 describes the derived theoretical lower bound for the convergence time and its implications in the design of the protocol operation. And finally, section 6.3 includes the performance analysis by means of simulation that confirms the propagation-based behavior and further evaluates the protocol performance.

6.1 Wave-fronts propagation effect

As described in section 3.3, the reception of a BPDU with a vector that updates the local information leads to the reconfiguration of the tree (reselection of port roles and port states) and the consequent dissemination of the updated information (send BPDUs to neighbors). This sequence of procedures is repeated every time a BPDU with a better vector is received at any bridge. Concretely at network start-up, this operation occurs several times because every bridge initially believes it is the Root and sends its own BPDUs. This results into many comparisons of received and stored vectors with different Root fields (like the case in the example in section 3.3 where $B4$ receives from $B0$). There are also comparisons between vectors of the same Root, but with different costs, or different Bridges, or different ports. Overall, the situation is quite complex to analyze in detail.



(a) t_0 (b) t_1

(c) t_2 (d) t_3

Figure 6.1: Propagation of wave-fronts during the tree construction

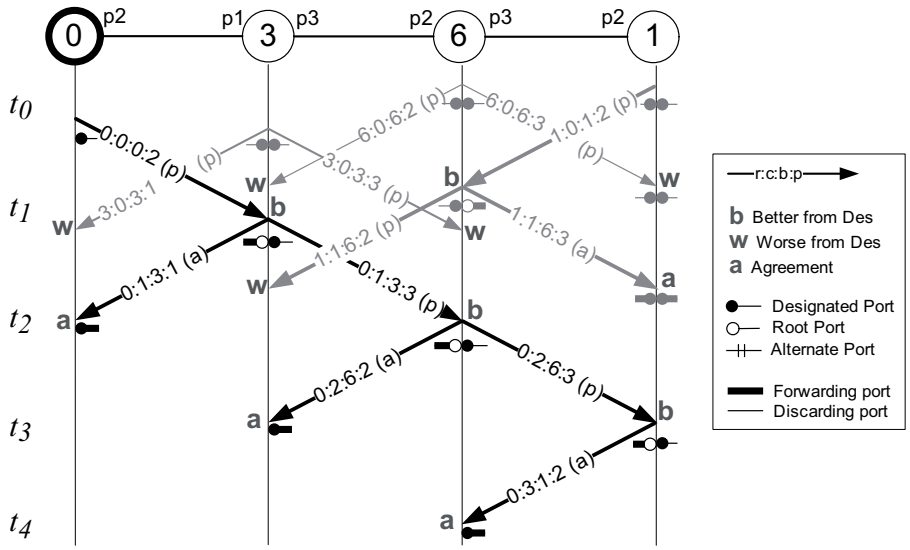


Figure 6.2: Diagram of exchanged BPDUs in a sub-set of nodes at network start-up.

However, what makes it simpler to observe and easier to understand is that a vector with the Root of the lowest BridgeID (0 in our case) always updates any other vector with another identifier as Root (0 is always lower than 2, 4, 27...). This means that the BPDUs of the Root $B0$ always update, always reconfigure the tree, and always get disseminated.

Furthermore, this local behavior can easily be seen in a graphical way if we imagine this sequence of events considering that each node starts propagating a wave announcing itself as Root. Figure 6.1 shows the wave-fronts started at each bridge where the grey levels of the arrows indicate the different BridgeIDs (darker is lower). Hence, the black arrows correspond to the Root $B0$ and the dark grey to the Root $B1$ (note the rest of wave-fronts are all depicted with light grey for simplicity reasons). The network snapshot at t_0 represents the start-up instant where all bridges are initialized as Root and send BPDUs. As the BPDUs of lower Roots are received and processed, these always win the comparisons, update the trees and make the wave-front advance. When two wave-fronts encounter, the one with the lowest Root is considered better, beats the other and continues its propagation while the defeated wave stops at that point. This comparison is done whenever two wave-fronts are faced and the winner always gets through. Therefore, the influence of a wave-front increases every hop as long as a coming wave-front disseminates a tree about a smaller Root than the own. But the wave-front diminishes when it contacts with a more 'powerful' wave-front, this is, with a smaller Root. Note that from a more technical perspective, a comparison of wave-fronts represents a comparison of priority vectors. This is the case at t_1 where $B0$ wins over $B2$-$B3$-$B4$, or $B1$ wins over $B5$-$B6$. Note that this is a transient situation as there are still several sub-trees configured in the network. As wave-fronts go forward (BPDUs get disseminated), observe how the black one advances more and more until it reaches the entire network at t_4. When the messages originated at the definitive Root span all the network, the other wave-fronts have already dissipated and the propagation of topological information is completed. At this point all nodes are aware of the final Root, their distance to it and therefore they store the right port roles.

The same wave-fronts evolution can be also observed with more detail in the diagram of exchanged messages in figure 6.2. The arrows represent the BPDUs transmitted (darker is lower) and note that only a sub-set of nodes is shown. See how the black arrows representing the wave-front of the Root $B0$ span the entire network while other wave-fronts are defeated as they encounter the black one ($B3$ and $B5$ at t_1, $B1$ at t_2). Also observe that the last messages processed are actually BPDUs with agreements terminating the handshake that are sent back to the parent node to activate the Designated port (for example $B5$ at t_4). Next section describes in more detail how handshake operation evolves.

6.2 Theoretical bound of the convergence time

The main conclusion of the wave-fronts observation is that the algorithm terminates when all messages starting at the Root reach the furthest bridge. Therefore,

the propagation through the longest tree branch determines the convergence time of the algorithm. It is straightforward to derive an analytical characterization of the RSTP convergence time in a cold start scenario:

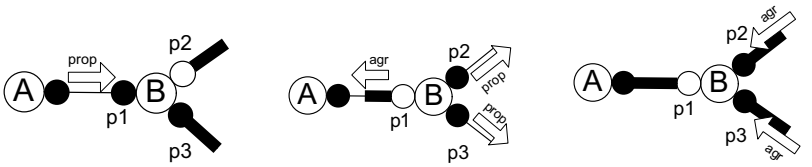
$$CT_{RSTP} = max(L_{br}) \times t_{hop} \tag{6.1}$$

where L_{br} is the branch length, t_{hop} is the hop delay.

This propagation time defines a theoretical bound of the convergence time. The feasibility of this theoretical bound depends on the ability to detect that the Root messages have fully propagated through the entire network. While this is easy to see in the figure it is very difficult to identify in the normal distributed operation. Nodes respond to received messages, and nothing is different in the messages after complete propagation. Therefore, it is not easy to detect the completion of the tree to stop the algorithm as all nodes operate distributed with minimal state and always waiting for the reception of a potential better message. This permanent transient situation must conclude at some point in order to consider the tree completely built and safely start forwarding data. Note that before the Root information has been completely propagated there are some nodes that have not yet received the Root wave-front and contain a tree configuration that will eventually be updated. Forwarding data based on this transient configuration can end up in forwarding loops resulting in broadcast storms as described in section 2.1.

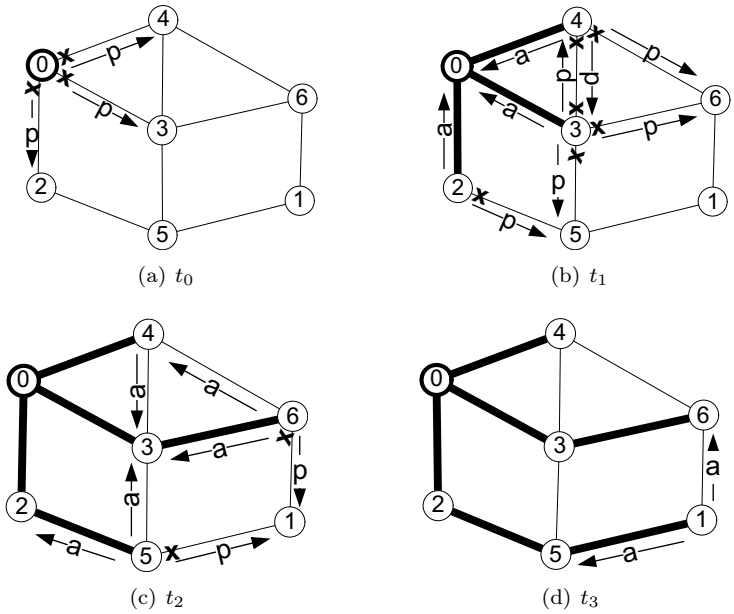
The original STP relies on timers that wait enough time until all messages have been propagated (the recommended value in the standard is 30 seconds). When this timer expires, the Root and Designated ports are set to Forwarding and start transmitting data traffic. The drawback of this approach is that the arbitrary value of the timers directly affects the recovery time. This value is calculated very conservatively considering the worst case scenario assuming a maximum propagation time to configure the values of the timers. This results into waiting for 30 seconds before the port states turn to Forwarding and hence data starts being transmitted.

On the contrary, RSTP uses the proposal-agreement handshake as a proactive technique no improve the termination detection. This mechanism is based on synchronization messages that are sent between a Designated port in a parent bridge and the Root port, or Alternate, in the corresponding child as the wave-front advances. The objective of the handshake is to allow for the transition to Forwarding of the Designated port in the parent. Figure 6.3 shows a general example where bridge A sends a BPDU marked as proposal in order to transition its Designated port to Forwarding (step 1). Bridge B receives this BPDU in the port $p1$ that also elects as new Root port (step 2). B blocks the Designated ports ($p2$, $p3$) so the Root port $p1$ can safely transition to Forwarding. Since the Designated ports are now Discarding, they start their own handshakes with the corresponding neighbor sending BPDUs marked as proposals. In turn, B sends back an agreement to A so it can transition its Designated port to Forwarding. Finally in step 3, B eventually receives the agreements from its child neighbors and also transitions the Designated ports to Forwarding.



(a) (step 1) (b) (step 2) (c) (step 3)

Figure 6.3: Local operation of the proposal-agreement handshake between direct neighbors



(a) t_0 (b) t_1

(c) t_2 (d) t_3

Figure 6.4: Evolution of proposal-agreement handshake in RSTP.

In order to understand how the local handshake affects the global network, figure 6.4 shows the overall perspective with the handshakes that occur within the propagation of the Root $B0$. At t_0, a port that is selected as Designated and Discarding state sends a BPDU that is seen as the handshake proposal. At t_1, these proposal BPDUs are received in the child nodes and they trigger the selection of the receiving port as Root port. In this case, the Designated ports of the receiving nodes are set to Discarding to safely transition the Root port to Discarding. This Root port in turn sends back an agreement BPDU so the Designated port in the parent can also transition to Forwarding. Since the Designated ports of the child nodes are Discarding, they start the corresponding handshakes with their neighbors sending proposal messages again. This handshake is repeated in a hop-by-hop basis until reaching the end of the tree, where nodes with

no Designated ports only send up agreements ($B5$ and $B6$ at t_2, and $B1$ in t_3). Relating to wave-fronts, the procedure can be easily understood if we consider that the blocked ports are pushed down the tree at every handshake until Alternate ports are found. This technique allows the nodes to start transmitting data as the tree is being built and hence the convergence time becomes proportional to the distance to propagate the wave-front of the Root node.

6.3 Performance evaluation of the initial tree construction

This section hence presents the performance evaluation of RSTP when creating the tree from scratch at network startup. We have implemented the protocols in the ns-3 network simulator [63][64] following the operations described in chapter 3.

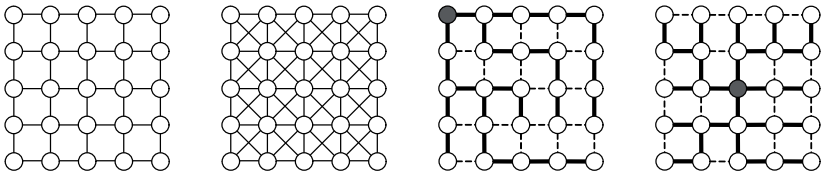
We run the protocols in different network topologies. In all the experiments, the Root is located in a given position and the rest of bridges are configured with random BridgeIDs. This allows us to configure random branches that grow from a known Root node. The time to transmit, propagate and process a BPDU depends on several aspects such as transmission rates, propagation delays, implementation of the BPDU processing unit, configuration of the bridge queues, etc. We take as a reference the study in [30] that assumes a delay of 1.33ms per message. The value of MaxAge is set to the number of nodes in the network, unless the experiment details otherwise, in order to ensure the propagation of a full wave-front. Only BPDU messages are simulated and no user traffic is modeled unless otherwise stated.

The performance evaluation focuses on the time to construct the tree (*Convergence Time*, CT) and the amount of information exchange required for such action (*Message Overhead*, MO). CT is defined as the time until the last port transitions to Forwarding state. We use the hop delay, t_{hop}, as the normalized unit for CT. That is, a CT of 5 hops means that the protocol takes 5 times the hop delay to converge. MO refers to the amount of messages that the nodes need to exchange in order to recover the tree. MO is measured in both number of messages and number of bytes.

6.3.1 Convergence time

In the first analysis we use the two-dimensional topologies shown in figure 6.5 (note that *grid4* and *grid8* refer to a two-dimensional mesh where the central nodes have a degree of 4 and 8, respectively). For each grid size, we run as many experiments as nodes and we locate the Root in a different location in every execution. Note that this results into forcing the BridgeID equal to 0 in one of the positions, while the rest of nodes are set with random identifiers. For example, figure 6.5(c) and 6.5(d) show the cases of locating the Root in the corner and the center of the grid, respectively.

Figure 6.8(a) shows the observed CT for different network sizes in the grid4 and grid8 topologies. The vertical axes indicate the CT normalized in hop counts



(a) grid4 (b) grid8 (c) Root in corner of grid4 (d) Root in center of grid4

Figure 6.5: Two-dimensional mesh topologies of degrees 4 (*grid*4) and 8 (*grid*8)

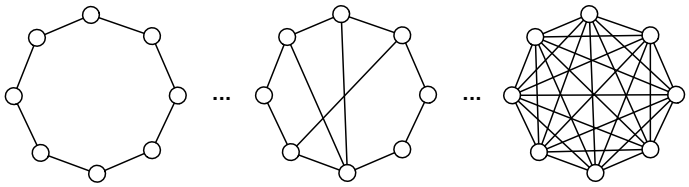
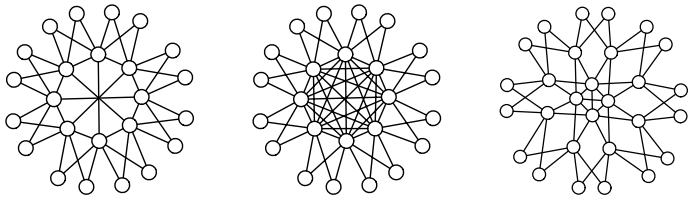


Figure 6.6: Ring-based topology of increasing connectivity (or average node degree).

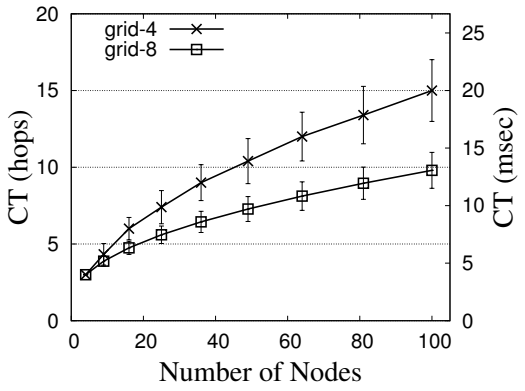


(a) mesh-heavy (b) mesh-full (c) two-tier

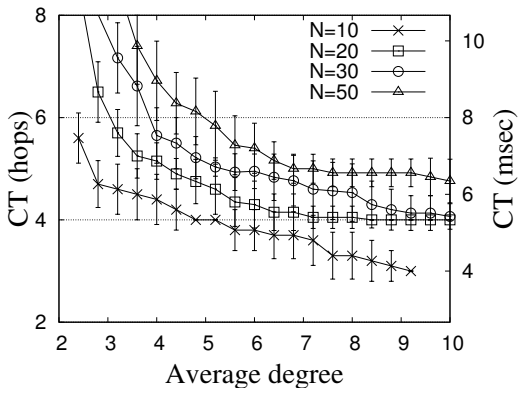
Figure 6.7: Realistic structured topologies

on the left and in absolute millisecond time on the right. The lines indicate the average CT values and the standard deviation is shown in the vertical lines at each point. The initial conclusion that we can extract from the observed CT is that RSTP is able to configure the tree after a few hop delay that translate into less than 25ms. This represents a big improvement over the performance achieved by the original STP. The CT of STP depends on the expiration of timers [65], which results into a constant convergence time of 30 seconds if the recommended timer values are used.

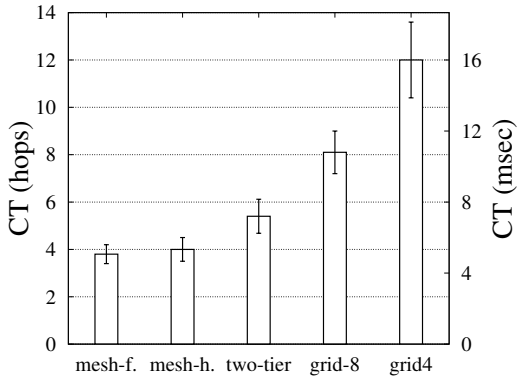
Another conclusion is that the convergence time depends on the size of the network as it grows with the number of nodes. More concretely, the CT depends on the distance between the Root and the furthest node. This is a direct consequence of the wave-fronts based propagation explained in section 6.1. Since RSTP terminates the tree configuration after the entire Root wave-front propagation and handshake agreements, the CT grows with the network diameter. Moreover, the wave-front of the Root determines the CT, hence locating this



(a) Two dimensional grid topologies of degree 4 and 8



(b) Ring-based topology of increasing the connectivity degree



(c) Grids, ring-based and structured topologies

Figure 6.8: CT of RSTP in a cold start scenario locating the Root in all possible locations of different topologies

node in a different location results in a different to construct the trees. If the Root is in one corner, the wave-front needs to traverse the entire network until the opposite corner. In the scenario with the Root in the center, the distance that the wave-front needs to advance is shorter (from center to corner).

In the experiments with the grid topologies, we fix the node degree and we vary the network size. In order to study the CT variability when varying the node degree, we use a ring-based topology where additional links are randomly introduced in order to increase the connectivity degree from an empty ring to a full-connected network (see figure 6.6). We also vary the size of the base ring and for each topology we run as many executions as nodes and locating the Root in a different place at each run. Figure 6.8(b) plots the CT measurements for different sizes of the ring-based topology (from 10 to 50 nodes) and varying the number of added links. The horizontal axis specifies the average node degree as a measure of connectivity and is computed as the number of links divided by the number of nodes (i.e. the empty ring has a node degree of 2; in the full network it increases to N-1, where N is the number of nodes). First, observe that the CT is higher for larger networks (as also seen in the experiments using the grid topology). Also note that for each size, an increase of the average node degree results into a decrease of the CT. This also relates to the diameter dependence because a more connected network has a smaller diameter. Actually, all sizes converge to a CT of 3 hops for a degree equal to the number of nodes (fully connected network). Note however that the plot only shows degrees lower than 10. Observing the values measured in milliseconds, the results confirm that in all cases the CT in a cold start scenario remains below the 50ms bound.

In order to generalize the previous two studies varying the size and the degree of the topologies, we evaluate the cold start performance in several more realistic network topologies. A part from the simple grid networks, we use structured topologies and shown in figure 6.7: the *mesh-heavy* that consists of a meshed core with dual-homed edges; the *mesh-full* that extends the previous one until fully connecting the core; and the *two-tier* that is composed by a smaller fully connected core and two levels of dual-homed tiers. Further details on these topologies can be found in [66]. Figure 6.8(c) shows the average CT and standard deviation for the different topologies (the grids have 64 nodes; the meshes have 50 nodes; and the two-tier has 56 nodes) and locating the Root in a different node at every execution. As seen in previous experiments, the diameter dependence also appears in the structured topologies (the columns are sorted based on each topology diameter). In addition, these topologies are physically designed to maintain the diameter regardless the network size, hence the CT does not grow with the number of nodes. The variability in the meshed topologies is smaller than in the grids because of the symmetry properties. Note that locating the Root in one of the nodes the core is actually the same experiment just varying the rest of BridgeIDs.

Generalizing all the results and assuming that the Root can be randomly located at any network node, a top bound of the convergence time for a given network topology is equal to the time required by the wave-front to traverse the diameter of the topology. As studied in [67], common provider networks have

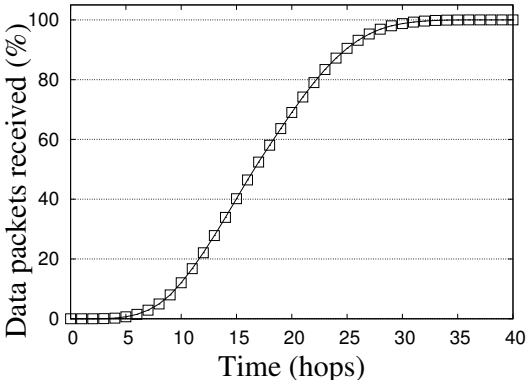


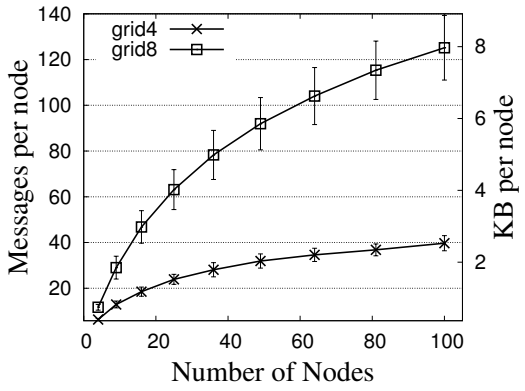
Figure 6.9: Traffic received by all nodes during cold-start

diameters between 4 and 6 hops, which result in sub-50ms convergence times as well.

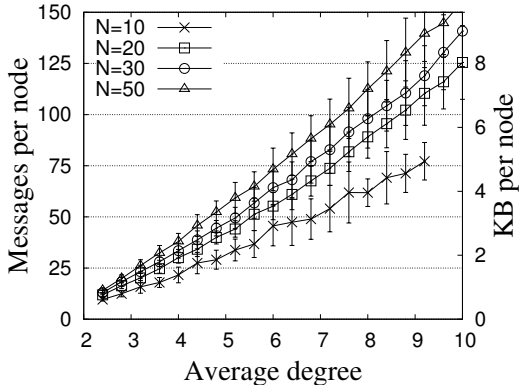
The convergence time can also be observed from the data traffic perspective. In the following test we measure the amount of data packets that nodes receive during a cold start. Note that this metric is related to the convergence time because the longer the protocol takes to configure the active topology, the longer is the period where data traffic is not received. In order to carry out these tests, each node sends broadcast data packets of 1000 bytes every 10us. Note that a node should receive one broadcast packet from each other device, hence the percentage of packets received represents the level of active connectivity at that instant. Plot in figure 6.9 shows the timeline of received packets in the grid4 topology of 64 nodes and setting the Root in one corner. The horizontal axis is measured in hops and the vertical axis in percentage of received packets by all nodes (100% represents 4032 messages). RSTP takes around 30 hops to reach the maximum connectivity because the Root wave-front first needs to reach the furthest nodes, and then this can start transmitting. This is why the 100% is achieved at hop 30 when the data messages of the node in the opposite reach the Root. The slope of the curve relates to the early stages where some nodes believe in transient configurations and are temporarily connected to a subset of nodes using a tree of a node that is not the final Root.

6.3.2 Message overhead

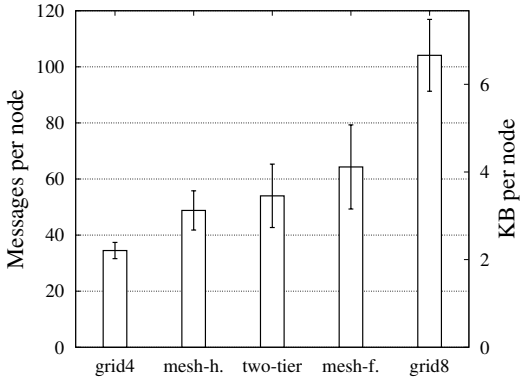
We now measure the message overhead (MO) due to the transmission of BPDUs in the cold-start case. We use the same scenarios that we have described in the convergence time study. Figure 6.10(a) shows the MO for the grid topologies of varying size. The vertical axes indicate the number of messages, on the left, and the transmitted Kilobytes, on the right. The overhead measurements are presented in per-node metrics (MO_{node}, messages per node), instead of the total amount, in order to easily compare networks of different sizes. The plot shows



(a) Two dimensional grid topologies of degree 4 and 8



(b) Ring-based topology of increasing the connectivity degree



(c) Grids, ring-based and structured topologies

Figure 6.10: MO_{node} of RSTP in a cold start scenario locating the Root in all possible locations of different topologies

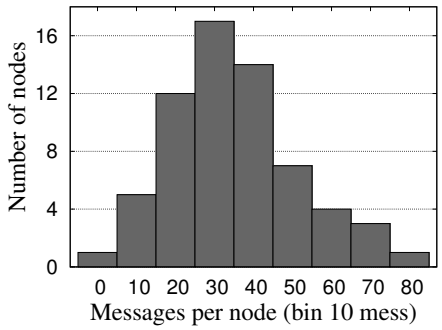


Figure 6.11: Histogram of MO_{node} in a grid of 64 nodes locating the Root in the corner

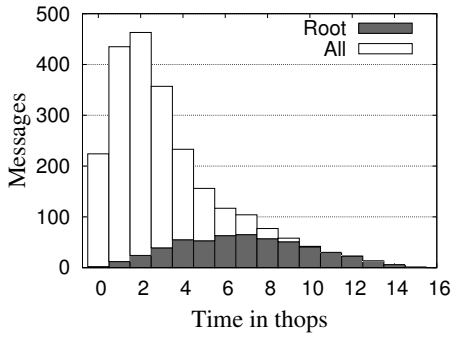


Figure 6.12: Timeline of BPDUs received by all nodes in a cold start scenario

that the MO_{node} grows with the number of nodes. In both grid4 and grid8 topologies, larger networks result in more messages per node transmitted, although the relationship between MO_{node} and network size is not proportional. The reason is that RSTP is based on the propagation of wave-fronts and hence it has a flooding nature. The amount of messages required by flooding protocols is proportional to the connectivity level of the topology, or the average node degree [32].

While the previous results represent the average values of MO_{node} accounting all the executions, the following analysis go into more detail and observe one of the runs with the grid4 of 64 nodes in order to understand how the message are really distributed. Figure 6.11 shows a histogram of the number of messages received per node in the execution locating the Root in one corner (i.e. the first bin indicates that 1 node has received 10 messages or less). While the average is 31 messages, we can see that the values have a high variability and are widely spread. The reason is the occurrence of transient configurations before the Root wave-front spans the entire grid. The nodes that receive more messages are actually those located in the center of the grid because they see more passing wave-fronts than those in the corners.

The same single execution with the Root in one corner of the grid4 of 64 nodes

can also be observed distinguishing the origin of the different BPDUs. Figure 6.12 shows a timeline of received messages by all nodes. The x-axis represents hop delays and the y-axis shows number of messages received during that period. The white boxes represent the total amount of messages and the superposed grey ones are the sub-set of messages that belong to the wave-front of the Root. First observe how at the initial steps there is a high number of non-Root messages because there are many wave-fronts still active. Also, note that as the time passes, the wave-front of the Root beats other wave-fronts that are discarded until, after 15 hop delays, the Root wave-front is entirely disseminated.

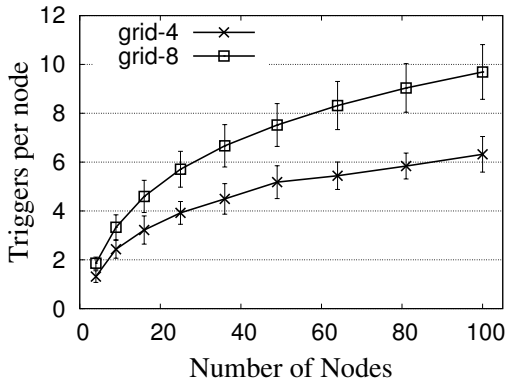
The proportionality of MO_{node} with the average degree can be easily observed using the set of ring-based topologies in figure 6.6. Plot in figure 6.10(b) shows the MO_{node} measurements for different ring sizes (from 10 to 50 nodes) and varying the average node degree. Observe that the overhead grows with the degree in all topology sizes. Although the lines are not completely straight, this results show that the node degree is the dominant factor in the characterization of the message overhead.

The relationship between the degree and the MO_{node} can also be observed in the tests with the different topologies of figure 6.10(c). Note that in this case the columns are sorted by degree. As previously mentioned, the degree might not be the only factor that determines the average per node, but results show it has a strong proportionality.

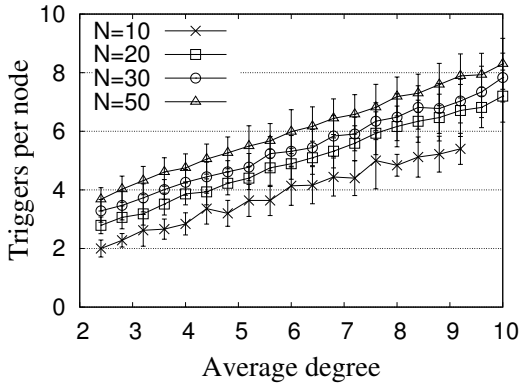
6.3.3 Triggers of tree calculations

The previous MO analysis has provided an evaluation of the overhead due to transmitted messages. These include all BPDUs that are received by nodes: those that update the state and trigger a tree reconfiguration, those that convey a worse priority-vector and are just discarded, BPDUs informing about the agreement of the RSTP handshake, etc. While MO is a measure of the practical overhead from link capacity perspective, it cannot be assumed as a measure of node processing overhead because not all received BPDUs result into a tree reconfiguration. The BPDUs that include a priority vector better than the locally stored are the only ones that really trigger a recalculation of ports roles and states. The metric that measures such messages is the *Triggers* (TR) and it corresponds to the number of messages that cause a recalculation. Note that this metric can be used to evaluate the real impact on the required processing at each node as only those triggering messages really require a reselection of port roles and port states.

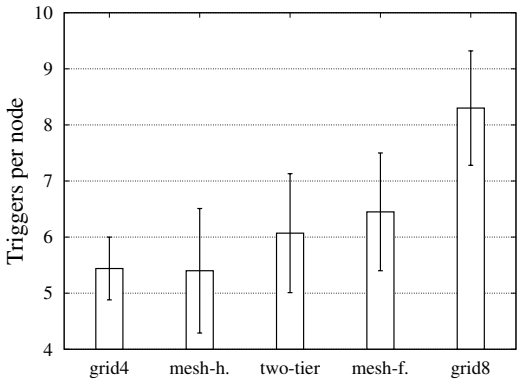
We measure TR in the same scenarios used so far. Figure 6.13(a) shows the TR in the grid4 and grid8 topologies. Note the vertical axis indicates number of triggers per node. The behavior of TR is similar to what we have described in MO: a higher connectivity degree results in more links and hence more messages transmitted. This relationship can also be seen in figure 6.13(b) where the experiments using ring topology of increasing node degree are shown. In this case the TR appears linearly proportional to the average node degree. And finally, the TR measured in the different topologies in the plot of figure 6.13(c) also shows a dependence on the average node degree.



(a) Two dimensional grid topologies of degree 4 and 8



(b) Ring-based topology of increasing the connectivity degree



(c) Grids, ring-based and structured topologies

Figure 6.13: TR_{node} of RSTP in a cold start scenario locating the Root in all possible locations of different topologies

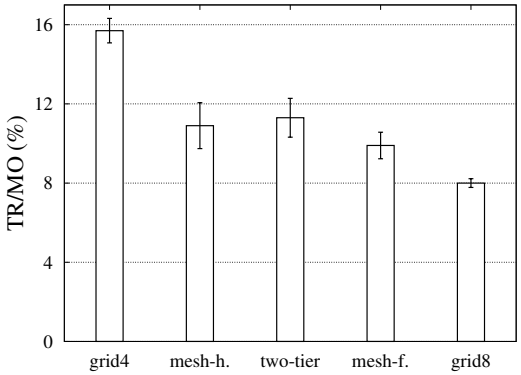


Figure 6.14: TR/MO ratio in a cold-start scenario for different types of topologies

Since the triggers are a subset of the total amount of messages, it is interesting to measure the variation of percentage of TR over MO for different topologies. Figure 6.14 shows the percentage of triggers calculated as TR/MO. Observe that for topologies with large degrees the percentage decreases. The proportion of TR over the entire MO can be seen as a measure of how efficient the protocol overhead is. Note that the triggers are the minimum messages required to configure the trees (as they are messages that update state). Therefore, a high percentage of TR/MO really means that MO is closer to the lower bound (TR). Contrarily, a low percentage of TR means that for that particular topology the total amount of messages required is further from the minimum.

§ 7. RSTP-Conf: Protocol Extension to Avoid Count-to-Infinity in RSTP

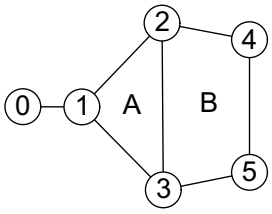
The main inconvenient of RSTP is the low performance when recovering from a Root failure because it suffers count-to-infinity. This effect results in BPDUs looping around the network avoiding the protocol to converge and hence delaying the recovery of the Root failure for tens of seconds. This chapter first provides a comprehensive description of the consequences of count-to-infinity in RSTP. We are not aware of any published study that provides a comprehensive explanation that justifies the large recovery times experienced. We have done a deep study of count-to-infinity scenarios by means of simulation and these have allowed us to identify the particular conditions that lead to unexpected consequences that delay the recovery for several seconds. Section 7.1 describes these hidden effects of count-to-infinity and section 7.2 introduces some possible approaches to follow in order to avoid such behavior.

Furthermore, the detailed study has allowed us to identify the reasons that trigger the count-to-infinity effect and, in turn, has driven the design of RSTP-Conf as the necessary extensions to RSTP in order to avoid it. RSTP-Conf introduces a simple yet effective confirmation mechanism that avoids the scenario where the looping BPDUs delay the Root failure recovery. Sections 7.3 and 7.4 describe the operational extensions that RSTP-Conf introduces. Finally, section 7.5 contains the performance analysis that evaluates the impact of count-to-infinity in RSTP and how effectively RSTP-Conf is able to avoid it.

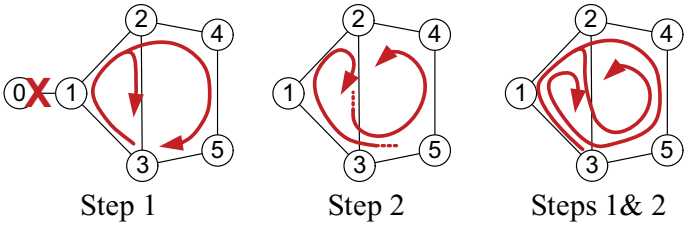
7.1 Hidden effects of count-to-infinity

In the count-to-infinity example shown in chapter 3 the BPDUs start looping around the only cycle that exists in the physical topology. As the messages are transmitted bridge after bridge, the MessAge is incremented hop after hop, and the counter reaches its maximum value of 20 hops after a few complete turns around the loop. At this point, the BPDUs about the old Root are completely removed and count-to-infinity terminates and the new Root finally constructs its new tree. Nevertheless, more complex topologies with more than one physical loop experience more complex behavior during the count-to-infinity that results into worse consequences in terms of recovery time.

An example is shown in figure 7.1. Note that in this physical topology the Root that fails is the outer node ($B0$) and that the remaining connections contain more than one loop (two of them are loop A and loop B indicated in figure 7.1(a)). The



(a) Network topology example with two loops



(b) Evolution of looping BPDUs

Figure 7.1: Looping BPDUs in a physical topology with several remaining loops after the Root failure

arrows in 7.1(b) represent the BPDUs about the failed Root ($B0$) that are being forwarded and cause the count-to-infinity effect. Right after the Root failure, the behavior is the same as in the example with a single physical loop in previous section. This is, BPDUs start looping around nodes $B1$-$B2$-$B3$. The difference is that nodes located between loop A and loop B ($B2$ and $B3$) have additional links where they also propagate the old information about $B0$. For example $B2$ at step 1 disseminates the BPDUs about $B0$ to $B3$ and to $B4$. Every time a BPDU about the failed Root enters a loop, the messages start being transmitted around the cycle. As shown in step 2, when the messages inside loop A reach $B2$ or $B3$, they are propagated into loop B again. Similarly, messages in loop B re-enter into loop A. As shown in the last diagram with the steps 1 and 2 together, the situation shows a complex and uncontrolled behavior as long as the looping BPDUs are propagated back and forth between loops. The more messages are propagated, the more loops are created and a more chaotic situation is reached.

7.1.1 Appearance of deadlocks

The scenario just described in figure 7.1 has a complex nature and it is evolution is difficult to characterize. However, one might expect that the BPDUs about the failed Root are removed after they have traversed 20 hops as it happens in then single loop topology. Therefore, the time required to remove the count-to-infinity should be equal to the propagation delay of 20 hops. Instead, exhaustive analyses by means of simulation show that the recovery time is much larger: order of several seconds when a single hop propagation delay is just a few milliseconds.

Analyzing the phenomenon in more detail we have observed that the BPDU transmission completely stops in the middle of the reconfiguration. The crossing of BPDUs with particular vector values results into the creation of a deadlock in a link where two Root ports are elected. Since Root ports wait for the opposite Designated to send refreshing BPDUs, a deadlock in this link arises. The silence in both sides prevents the existence of BPDUs in this link and hence any possibility to be reconfigured except after waiting long enough for BPDUs, when the timer 3xHelloTime expires and releases the deadlock. The timeout allows considering the port state outdated and is being reset losing the information of the "failed" Root. At this point the port is reconfigured to Designated role and the algorithm continues normally. The count-to-infinity problem has not disappeared as other ports in the network still have in their state the failed Root with a MessageAge smaller than MaxAge indicating that it is valid. Hence the count-to-infinity continues with the state information of these other ports at the current value of the MessageAge. Several deadlocks might appear at different links until the MessAge of the BPDUs reaches 20. When this happens, the BPDUs are finally removed and the new tree is finally constructed. This effect results into a non-continuous count-to-infinity due to the delay introduced by the deadlock.

An example of such situation is presented following. Figure 7.2 shows the case of the network topology in figure 7.1 where two crossing BPDUs between nodes $B2$ and $B3$ result into the deadlocked pair of ports. The diagram shows the exchanged BPDUs between both neighbors and the tables on the sides indicate the vectors stored in the bridge and in each port of both $B2$ and $B3$. Note that only Root, Cost and Bridge fields are shown. First of all note that during count-to-infinity the looping BPDUs disseminate a cost that is not corresponding to the real physical topology. This is the reason why at t_1, $B2$ and $B3$ receive BPDUs from their parents announcing a cost of 5 to the Root. From the overall perspective, it is clear that this cost is wrong. However, RSTP bridges operate distributed and assume that the received information always represents the real topology. Both $B2$ and $B3$ accept the received information, even if it is worse than the locally stored, because it comes from the parent. This results in $B2$ selecting $p1$ as Root port and disseminating the new cost through $p2$ (0:6:2). Similarly, $B3$ selects $p2$ as Root port and disseminates through $p1$ (0:6:3). These are the two BPDUs that eventually get crossed and result in the deadlocked configuration. Before this happens, at t_2, $B2$ and $B3$ receive another BPDU from their parents with a larger cost (0:6:4 and 0:6:3, respectively). These messages also update because they come from the parent. Both nodes select the same Root port they already had, but now the cost in the stored vectors has increased.

At t_3, $B2$ and $B3$ receive the aforementioned crossed BPDUs. The diagrams in figure 7.3 show the vector updates in $B2$ after processing this BPDU. As shown in 7.3(a), $B2$ receives the vector 0:6:3 in $p2$, and it updates the locally stored (0:7:2) because the received contains a lower cost. When $B2$ reconfigures the tree, it has to choose the Root port among $p1$, with 0:6:4, and $p2$, with 0:6:3 (as seen in figure 7.3(b)). The cost is the same in both port vectors, but $p2$ holds a lower Bridge field ($3<4$). Hence $B2$ selects $p2$ as Root port and $p1$ as Designated (as seen in figure 7.3(c)). Similarly, $B3$ selects $p1$ as Root port because the received vector

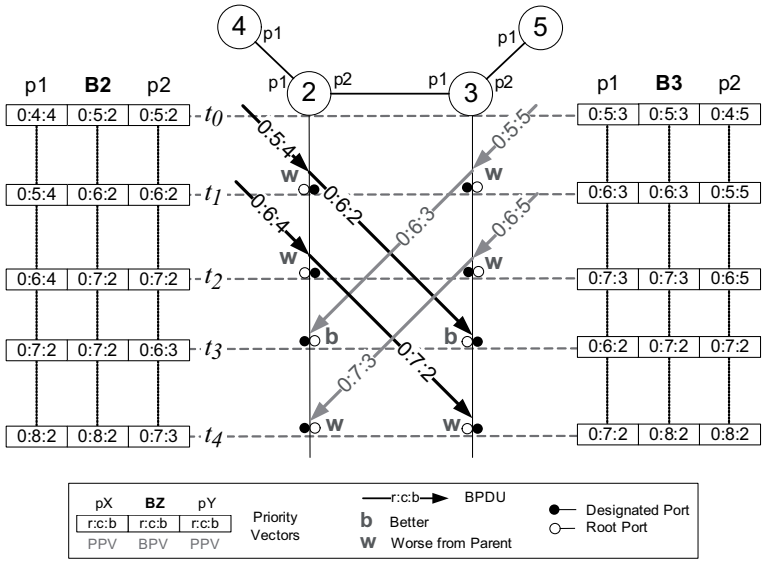
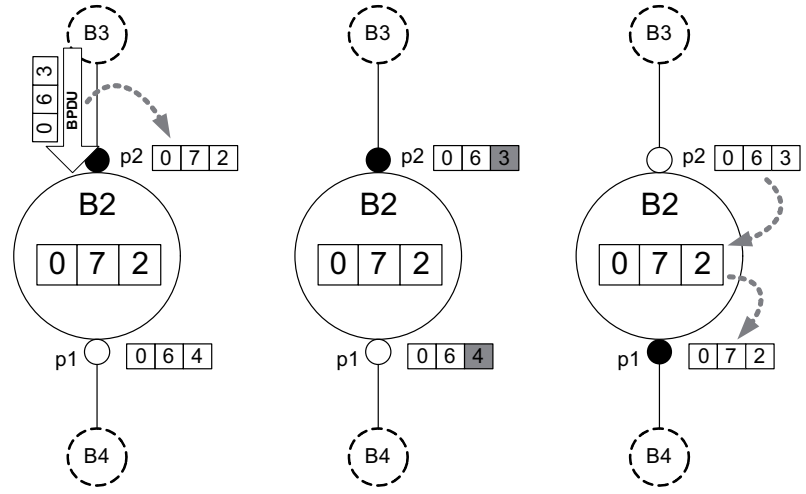


Figure 7.2: Diagram of exchanged BPDUs in a count-to-infinity scenario where a deadlock appears between bridges $B2$ and $B3$.



(a) Received BPDU has a lower cost

(b) p2 is the new Root port because of the lower Bridge field

(c) All vectors are reconfigured

Figure 7.3: Detail of $B2$ vectors update after processing one of the crossed BPDUs received at t_3

in $p1$, 0:6:2, is better than the vector in the $p2$, 0:6:5. Note that this results in the election of two Root ports in both sides of the link. This configuration is not possible because no tree can have such structure. The cause of this double Root port selection is actually the reception of the BPDUs that increase the cost at t_2. This cost increase results in interpreting the initially crossed BPDUs as better vectors in both sides of the link and the consequent double Root port selection.

Once the two Root ports have been selected, the rest of BPDUs received by $B2$ or $B3$ do not result into any port roles change. For example, the last two BPDUs received at t_4 update because they come from the parent, but they only result into an increase of the cost seen by the nodes. In addition, there are more BPDUs received in the Designated ports $p1$ of $B2$ and $p2$ of $B3$. These all convey a larger cost due to the count-to-infinity behavior and hence they are always considered worse, thus directly discarded, and do not trigger any tree configuration that changes the port roles. Since the BPDUs always flow from the Designated port of a parent to the Root or Alternate ports of a child, the configuration of this link with a double Root port stops the propagation of the looping BPDUs at this point. Each Root port actually assumes that there is a Designated port at the opposite side that is going to send a refreshing BPDU, but this will not happen. In addition, messages received in other ports of these two bridges do not break the deadlock because the increasing cost because of count-to-infinity makes that the deadlocked vectors are considered better. Therefore, a deadlock between the two Root ports arises.

The absence of received BPDUs in the two Root ports eventually expires their MessageAgeTimer timer that RSTP uses to detect a lack of received messages in a port (3xHelloTime). $B2$ and $B3$ timeout the information stored in their Root ports and both nodes reconfigure the tree. This results in the nodes continuing the looping of messages if they have an Alternate port with a vector about the failed Root. This new situation is like another count-to-infinity with a starting value of MessAge closer to MaxAge (at the value where it stopped because of the deadlock). This second phase might end up in another deadlock in some other link and the MessageAgeTimer would be needed again to continue the count-to-infinity until MessAge definitely reached MaxAge.

In conclusion, the occurrence of these deadlocks represents a temporary pause in the process of reaching MaxAge. The use of a timer to detect the lack of message reception results into a high increase of the recovery time and introduces a timer dependence that translates into recovery times proportional to the timer values (order of seconds).

7.1.2 Virtual Root creation

In the structure of the shortest-path tree constructed by RSTP, the Root ports point upward to the Root node, the Designated ports point downward to the leaves. In addition, BPDUs are sent from the Root of the tree toward the leaves, hence a Designated port disseminates BPDUs. Therefore, the real meaning of a node sending a BPDU through one of its Designated ports is that the Root node is located somewhere behind the transmitter. First diagram of figure 7.4 shows

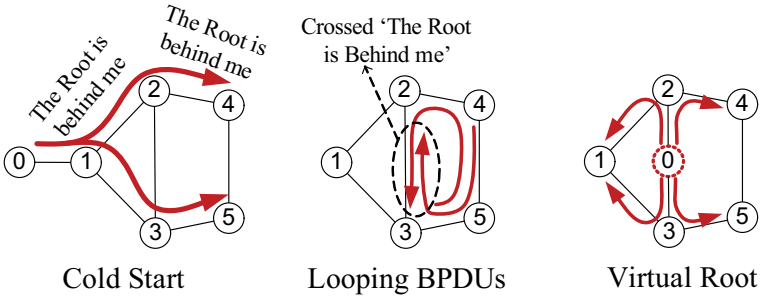


Figure 7.4: Diagrams describing the creation of a virtual Root when a deadlock appears in a count-to-infinity situation.

the initial network start-up case (cold start) where the BPDUs originated at the Root node are propagated node after node. For example, $B1$ is telling $B2$ and $B3$ that the Root is behind $B1$. $B2$ and $B3$ also tell $B4$ and $B5$, respectively, that the Root is behind the first two.

When the count-to-infinity occurs after the Root failure, each looping BPDU is still understood as a notification that the Root node is behind the transmitter. The crossing of BPDUs implies that the two nodes involved in the deadlock assume that the Root is behind the opposite neighbor (second diagram in figure 7.4). This is why both configure the Root ports pointing to each other. This actually results into the creation of a virtual Root in the middle of the affected link, as shown in last diagram in figure 7.4. This tree with a virtual Root remains as the temporary topology if no further action is taken. All nodes assume that the tree is stable, although it is rooted at the virtual Root, and operate normally sending the refreshing BPDUs every HelloTime. All Root and Alternate ports receive the periodical BPDUs except the two Root ports configured in the link where the virtual Root arises. When the MessageAgeTimer of these ports expires, nodes $B2$ and $B3$ clear the information about the virtual Root in their Root ports and reconfigure the tree using information in other ports. The count-to-infinity continues and a deadlock somewhere else might result in a different virtual Root location.

7.2 Approaches to avoid count-to-infinity

RSTP takes the design decision to keep simplicity in the protocol operation at the expenses of experiencing count-to-infinity after a Root failure. Note that, even if the recovery might be delayed for several deadlocks, the protocol eventually terminates the count-to-infinity effect and hence the recovery time is large but finite. However, an outage that expands to several seconds cannot be considered for current production networks. Therefore, one of the RSTP extensions required is the avoidance of the count-to-infinity effect.

The cause of the count-to-infinity is essentially the use of old Root information stored in the Alternate ports. This section describes different approaches in order

to limit the use of such information with the objective of avoiding the count-to-infinity behavior.

Avoid the use of Alternate ports A simple way to elude the problem is avoiding the use of the vectors stored in the Alternate ports when the tree is recomputed. This modification would solve the count-to-infinity issue but it would not allow a quick link failure recovery. As described in section 3.4, when an RSTP node detects a failure on its Root port it immediately selects an Alternate as new Root (using the Alternate vector as the new information to reach the Root). If we avoid the use of the Alternate ports we miss this automatic recovery capacity.

The critical issue is that the distributed nature of the spanning tree protocols does not allow distinguishing between a link failure and a node failure. When a node detects a failure on one of its ports, it does not even know if only the link or the entire neighbor has failed. In extension, when the Root has failed there is no other node that reliable knows the situation. Since nodes cannot distinguish between different types of failures, the protocol must operate equally in all cases. This leads into taking the design decision: (1) use the Alternate ports, allow a quick link recovery, but experience count-to-infinity when the Root fails; or (2) do not use Alternate ports, delay all failures recoveries, but avoid count-to-infinity.

Avoid the propagation of potential false information Another approach to control the use of the Alternate ports is to limit the propagation of information to avoid a possible count-to-infinity. This is the strategy adopted by the original STP and it is a very conservative solution.

An STP bridge that detects a failure on its Root port initially operates as RSTP and does select an Alternate as new Root port (so far, this would create count-to-infinity). However, child nodes do not accept the disseminated and misleading information because these messages are considered worse. The difference with RSTP is that STP nodes discard all worse messages even if they come from the parent. This operational detail avoids the propagation of the information about the old Root stored in the Alternate port of the parent.

Since the BPDUs from the parent do not update, the child node eventually detects a lack of received messages in that port, through expiration of MessageAgeTimer, and realizes that the vector stored on this particular port cannot be used because it has not been refreshed. The child then recomputes the tree excluding the expired vector and selects new port roles. If the Root has really failed, the Alternate ports in the child node have not been refreshed and the information about the old Root has also been removed. If, however, the Root is alive the refreshing messages are received in the Alternate ports and the child node can use the vectors on them when it reconfigures the tree.

This approach successfully avoids the count-to-infinity behavior but it introduces an unnecessary delay on the single link failure recoveries. Since the worse information from the parents is not accepted, the recovery is not effective until the nodes expire their vectors. The MessageAgeTimer used in STP for this ex-

piration has a default value of 20 seconds. Hence this is the minimum time that an STP network takes to recover from any failure. Note that setting the timer values needs the consideration of the worst-case scenario so these are usually over-dimensioned.

Delay the use of information in Alternate ports Our proposed approach to avoid count-to-infinity is to extend RSTP in order to make a safer utilization of the information in the Alternate ports. Instead of just avoiding the use of Alternate vectors as in the first approach, we implement a confirmation mechanism that allows for the utilization of such vectors only if the Root node is still alive. We refer to this extension of RSTP as RSTP-Conf.

RSTP-Conf is a reactive technique based on an exchange of messages. No timers are involved so the delay occurred in failure recovery is only due to the processing and propagation of such messages and not to possibly over-dimensioned timers (as in the STP case in the previous approach). Next sections describe in more detail the fundamentals and operation of the extended protocol.

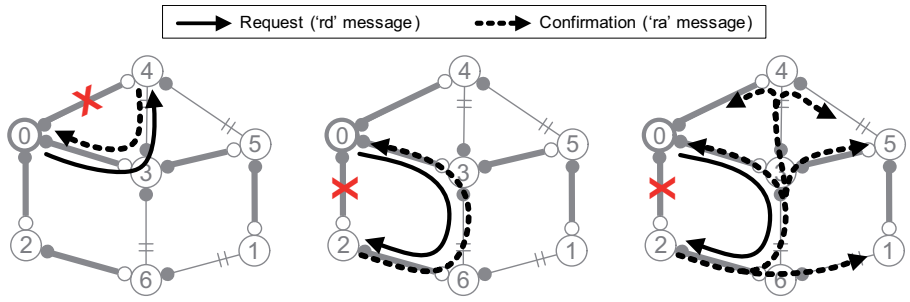
7.3 Fundamentals of RSTP-Conf

The complexity of the problem is related to the distributed nature of the protocol. Bridges just detect port failures and hence they are not able to distinguish between a failure of the link connecting that port or a failure of the entire neighbor. This is why the protocol performs the same operation in front of any type of failure and hence why count-to-infinity arises when the Root fails. In order to solve the problem, RSTP-Conf introduces a mechanism to identify the failure of the Root bridge.

The main challenge is that within the distributed nature of RSTP, nodes only know who the Root is and in which direction it is located. A smart way to use such information is to focus on the Root neighbors: the nodes that know for sure that they are directly connected to the Root. The idea is to exploit this fact and design a solution that distinguishes when the Root has failed and hence avoid the count-to-infinity. The proposed approach accomplishes this objective by dividing the problem in two sub-problems: (1) how to make a safe use of an Alternate port; and (2) how to reliably detect that the Root has failed.

7.3.1 Safe utilization of Alternate ports

When a Root neighbor detects a failure on its direct connection to the Root bridge, it realizes that this port failure might indicate the entire Root failure. Figure 7.5(a) shows an example where the link between the Root $B0$ and the neighbor $B4$ fails. $B4$, instead of reconfiguring the tree and selecting the Alternate port as new Root port, what would trigger a count-to-infinity, sends a message asking whether the Root is still alive. This request eventually reaches the Root $B0$ and it replies affirmatively back to the neighbor $B4$ that initiated the query. Note that the message asking for the Root's availability cannot be answered by any



(a) $B4$ sends request and Root $B0$ replies with confirmation

(b) $B2$ sends request and Root $B0$ replies with confirmation

(c) The requests messages are flooded to ensure they reach the Root

Figure 7.5: The Neighbor detecting the failure sends a message requesting for Root availability

node because this might still believe in a Root that has possibly failed, and again this would result into a count-to-infinity. Once this reply is received in $B4$, this now knows for sure that the Root is still alive and hence it can use the information in the Alternate port and make it the new Root port. With this procedure the neighbor requests and receives a confirmation to use the information stored in the Alternate port.

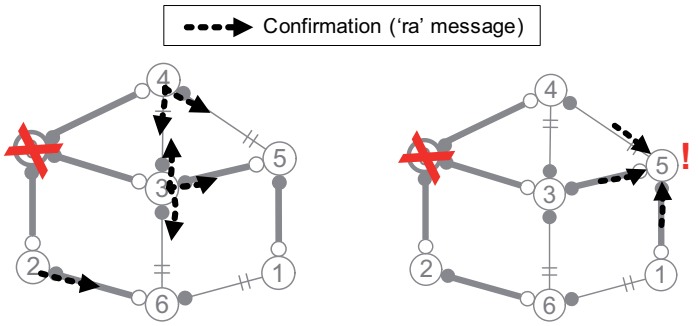
Failure detection by a Root neighbor without Alternate ports (as $B2$ in figure 7.5(b)) does not change from an operational perspective. The neighbor still sends the request asking if the Root is available. In this case, the neighbor $B2$ is asking for a confirmation to reconfigure the tree and arise as Root. It needs the permission because such situation would lead to a count-to-infinity triggered by the first child node with an Alternate port.

Observe that the neighbor sending the message asking for the Root availability does not really know the location of the Root and hence it does not know in which direction (outgoing port) it should send the message. The simplest way to solve this issue is to flood such message and ensure that it arrives to the Root as long as it is still available (see figure 7.5(c)).

7.3.2 Reliable detection of the Root failure

The previous example describes the situation where the Root is still alive and replies to the neighbor that originally asked for confirmation. The case when the Root has really failed needs to be treated differently because no one will actually reply the request.

Figure 7.6 shows an example where the Root fails. In this case, as shown in 7.6(a), all the neighbors detect a failure on their connections to the Root and they all flood an alarm message telling that they believe that the Root is dead. As mentioned before, each neighbor is only aware of the failure of its connection to the Root failure. In consequence, the reception of all the alarms in a network node can be seen as a signal to notice that the Root has failed (see figure 7.6(b)).



(a) All neighbors flood alarms

(b) All nodes receive all the alarms

Figure 7.6: The Root failure is detected across the network

In this technique, a neighbor really acts as a witness of the Root availability: the Root can be reliably considered unavailable when all witnesses, or neighbors, individually confirm that the Root has failed. When a node has received the alarms from all neighbors, and realizes about the Root failure, it arises as potential new Root (as in a cold-start scenario). A new Root is elected and the tree is recovered. Note that with this mechanism, any network node can detect when the Root has failed and hence it allows for a global tree reboot triggered within a distributed framework.

7.3.3 Same solution for the two sub-problems

A particularity of the two previous mechanisms is that they can be implemented using the same procedure. The message that a Root neighbor sends to the Root bridge asking about its availability can be the same message that it uses as an alarm to notify that it believes that the Root is dead. Therefore a single type of message allows to (1) obtain a confirmation about Root's availability and (2) globally detect that the Root has failed and hence "reboot" the tree.

Note that from the neighbor's perspective there is not any difference between the two situations in figures 7.5 ant 7.6. The neighbor just floods a message asking whether the Root is available or not. The difference is the consequence of such message: it either receives (a) a confirmation from the Root in 7.5, or (b) the alarm messages from the other neighbors in 7.6. From now on, the multi-purpose messages issued by the Root neighbor are referred as 'rd' messages ('Root dead') and the Root reply as 'ra' messages ('Root alive').

7.4 RSTP-Conf operation

In this section we detail the description of RSTP-Conf operation that extends RSTP by introducing the necessary elements to implement the confirmation mechanism. The base of the RSTP-Conf operation is the standardized RSTP,

hence most of the functionalities described in chapter 3 still apply. The following description focuses on the differences that RSTP-Conf introduces. This is why we re-use and extend the pseudo-code previously described for RSTP.

7.4.1 Confirmation variables

In order for the confirmation mechanism to operate properly, bridges need to be aware of several aspects related to Root neighbors. All nodes must know how many neighbors the Root has. The bridge variable *numNeigh* stores this value (table 7.1 includes all additional variables). Note the Root is the only one that really knows how many neighbors it has, hence it initially sets its own variable to its number of ports (line 1 of the updated *BecomeRootBridge* procedure in block E of the pseudo-code in figure 7.13). In the example topology, observe in figure 7.7(a) how the Root $B0$ initializes numNeigh to 3 because it has 3 ports that connect to 3 neighbors.

A simple way to distribute the previous information is to include it in the BPDU as the additional field *numNeigh* (table 7.2 shows the updated format of the BPDU). This new field requires a small update in the event *BPDU received* and the pseudo-code in block C.1 describes the operational changes introduced by RSTP-Conf (the updates are included in lines 3 to 6). Observe in line 3 that the numNeigh field in the received BPDU is stored in the local numNeigh variable. In the example of 7.7(a), the Root $B0$ distributed a numNeigh equal to 3 through the disseminated BPDUs. When $B4$ receives one of these BPDUs, figure 7.7(b), it updates the local numNeigh variables to 3 (number of ports of the Root bridge announced in the received BPDU).

Those nodes directly connected to the Root need to know they really are Root neighbors. The bridge variable *IamNeigh* is a flag that stores this property (as shown in table 7.1). As shown in line 2 of the *BecomeRootBridge* procedure of block E and in the example of figure 7.13, this variable is initialized to false because a Root node does not have any Root neighbor. Whether a bridge is neighbor or not is checked during the processing of a received BPDU. As shown in lines 4-5 of the *BPDU Received* event of block C.1, a node realizes that it is a Root neighbor when it detects that its parent is the Root node. This is true if the Bridge field stored in the vector of the Root port is the same as the Root field of the bridge vector. In such case, IamNeigh is set to true. Otherwise, to false. In the example of figure 7.7(b), $B4$ decides it is a Root neighbor (IamNeigh to true) because the believed Root bridge in BPV (0) is the same as the Root field in the PPV of the Root port (0).

Another variable stored by each node is an array that stores whether the 'rd' messages of each neighbor has been received. This array is called *rcvdFromNeigh* (see table 7.1) and stores the portID of the local node where the 'rd' from a particular neighbor has been received (for example, node A storing $rcvdFromNeigh[B] = p$ means that the A has received the 'rd' from B in port p). This array is cleared in the initialization of a bridge when it becomes Root (line 3 in the procedure *BecomeRootBridge* in block E) and every time a better BPDU is received (line 6 in the event *BPDU Received* of block C.1). In the first case, it is cleared be-

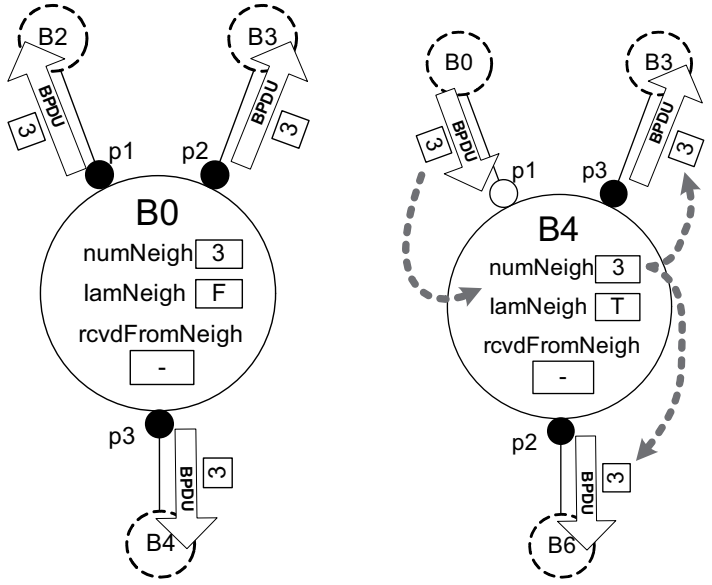
Table 7.1: Additional Bridge Variables

Name	*Description*
numNeigh	Stores the number of neighbors of the Root bridge. This information is disseminated with the common BPDUs and it is used to detect the reception of the alarms from all neighbors and hence reboot the tree.
IamNeigh	Boolean that indicates whether the local bridge is a direct neighbor of the Root. A bridge needs to know whether it is a Root neighbor or not because it performs differently in the event of failure detection
rcvdFromNeigh []	Array that stores the 'rd' messages received from neighbors. It actually stores the portID of the port where the 'rd' message has been received so the 'ra' message can be replied back

Table 7.2: Extended BPDU Frame Format

Name			*Description*	*Bytes*
Eth. encaps.	*SA*		The BPDU encapsulation is the same than in RSTP, hence the Ethernet MAC layer does not need to be updated.	6
	DA			6
	Length/ Type			2
	LLC			3
BPDU Fields	*Protocol id.*		Protocol identification fields indicating that the BPDU received corresponds to an RSTP-Conf instance and it is a common BPDU	2
	Version			1
	Message Type			1
	PV	*Root*	The core operation of RSTP (calculation of the tree shape by means of the priority vectors, proposal-agreement handshake, or timers configuration) is not changed in RSTP-Conf.	8
		Cost		4
		Bridge		8
		Port		2
	MessAge			2
	Flags	*Role*		1
		Prop.		
		Agreem.		
	MaxAge			2
	HelloTime			2
	ForwardDelay			2
	numNeigh		It contains the number of neighbors of the Root. The Root bridge encodes its number of ports and the dissemination of BPDUs distributes this value to all the other nodes	2
Frame Check Sequence				4

cause the bridge is Root and hence it knows it is alive. In the second case, the reception of updated information from a Root node means that any confirmation mechanism started is stopped because the node has received news from the Root, hence it is still alive.



(a) Root $B0$ initialized as non-Neighbor

(b) $B4$ set to Root neighbor

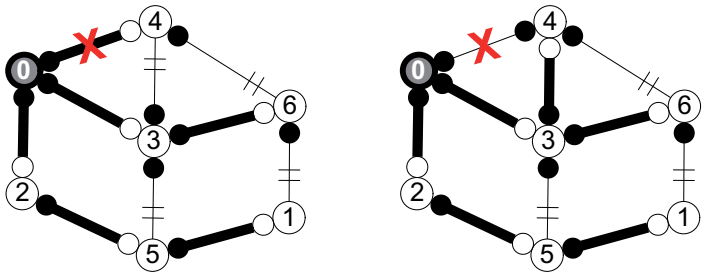
Figure 7.7: Initialization and distribution of confirmation variables

7.4.2 Trigger of the confirmation mechanism

The confirmation mechanism is triggered when one of the neighbors detects a failure on its Root port. This is the case of the neighbor $B4$ detecting the failure of its Root port as seen in figure 7.8(a) (observe that this example covers the case of a single link failure). Lines 2-5 in *MessageAgeTimer Expiration* event in block D of figure 7.13 describe the operation performed by $B4$. As the common RSTP, this neighbor removes the information in the failed port because it is not valid (as shown in the node diagram of figure 7.9(a)).

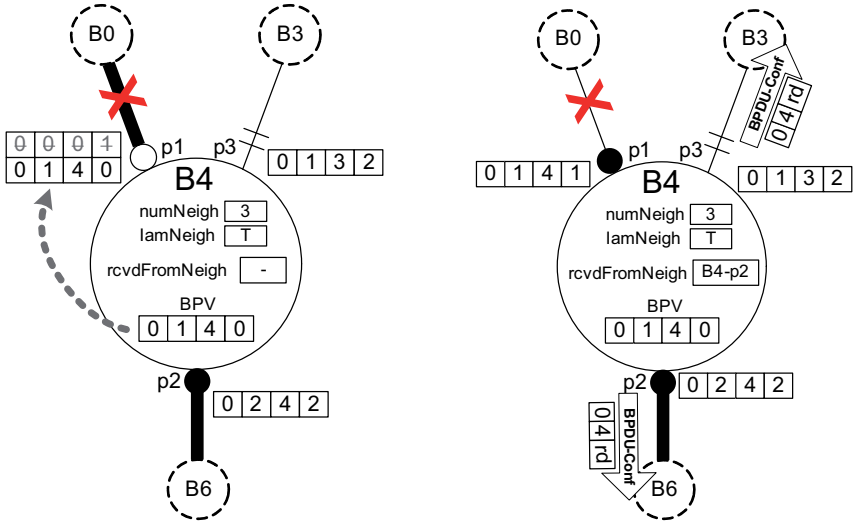
The difference with RSTP is that before reconfiguring the tree, this neighbor starts a flooding of a confirmation message (BPDU-Conf) asking whether the Root is alive (figure 7.9(a)). Note that selecting the port roles now would result in the Alternate becoming the new Root port. If the entire Root had failed, this would trigger a count-to-infinity. The confirmation is triggered right before this configuration in order to avoid it. As shown in the events and procedures diagram in figure 7.12, the added confirmation functionality only lies in the operation triggered by the Message Age expiration event right before the tree is reconfigured. This is the point in the operation where the confirmation procedure is activated. Also observe that if the failure is detected in a Designated or Alternate port, or in a non-neighbor node, the protocol operates as in the original RSTP and the confirmation mechanism is not triggered (lines 6-7 in block D).

Table 7.3 contains the fields included in the BPDU-Conf messages: the identifier of the Root (*Root*), the identifier of the neighbor that initiates the mechanism



(a) Tree before recovery (b) Tree after Recovery

Figure 7.8: Example of a single link failure recovery



(a) Port failure detection and port vector removal (as common RSTP)

(b) Trigger of the confirmation mechanism

Figure 7.9: Confirmation mechanism triggered by the neighbor $B4$ when it detects the failure in the Root port

(*Neigh*) and a flag that indicates the type of confirmation (*Type*, whether it is an 'rd' or an 'ra'). Note that from the bridge perspective, these BPDU-Conf messages are just a BPDU that the LLC layer delivers to the STP instance of the node. The differences between these and the common RSTP BPDUs are all treated within the STP instance.

The propagation of the BPDU-Conf messages is shown in the diagram of figures 7.10 and 7.11. In this example, $B4$ detects the failure at t_f and floods the confirmation messages including the Root, its own BridgeID as neighbor identifier, and the type 'rd' (0-4-rd). Note that the procedure of $B4$ at the instant t_f is the

Table 7.3: BPDU-Conf Frame Format

	Name	*Description*	*Bytes*
Ethernet encapsulation	*SA*	Management MAC address of the bridge.	6
	DA	Reserved address to indicate 'all bridges' (01:80:C2:00:00:00). RSTP-Conf it is still broadcast protocol and messages are not sent to a particular destination. BPDU-Confs are consumed at the next bridge hop and only live in the link they are produced. These frames never bypass a bridge but instead the bridge consumes and reacts to it producing a corresponding BPDU-Conf for the following neighbors when needed.	6
	Length/ Type	Indicates total length of the frame	2
	LLC	Indicates the bridge client that processes the received frame and delivers it to the STP client (DSAP:42; SSAP:42; Cntrl:03).	3
BPDU-Conf fields	*Protocol Identifier*	Protocol identification fields indicating that the BPDU received corresponds to an RSTP-Conf instance and it is a BPDU-Conf.	2
	Version		1
	Message Type		1
	Root	The identifier of the Root. This BPDU-Conf message might ask for the Root availability (this one) or might contain a reply from the Root (this one).	8
	Neighbor	The identifier of the neighbor that has initiated the request for Root availability.	8
	Type	It indicates whether the BPDU-Conf is an 'rd' message (type 0) or an 'ra' message (type 1).	1

one shown in figure 7.9.

All network peers eventually receive the 'rd' messages issued by $B4$. Lines 3-12 of the event *BPDU-Conf Received* in block C.2 of figure 7.13 describe the operation to process these messages. A node that receives one for the first time (see the node diagram of $B3$ at t_1 in figure 7.11(a)) updates the variable rcvdFromNeigh to store that the neighbor, included in the BPDU-Conf Neigh field, believes that the Root, in the Root field, has failed. Note that the actual value stored in rcvdFromNeigh is the port identifier where the 'rd' is received because the later 'ra' reply will be forwarded to this port so as to reach the neighbor that triggers the confirmation. Since $B3$ is not the Root, it continues the flooding of the 'rd' messages on its way to the Root.

When the Root receives such 'rd' (line 9 in block C.2), it replies with a BPDU-Conf of type 'ra' indicating that the Root in the Root field confirms its availability to the neighbor in the Neigh field. In the example, $B0$ receives the 'rd' from $B4$ at 2 and it replies with 0-4-ra. This 'ra' message is sent back to the original neighbor that issued the 'rd'. Every intermediate node that receives this BPDU-Conf forwards it to the port leading to the given neighbor (stored in rcvdFromNeighbor) and clears the corresponding confirmation variables (lines 14-15 of block C.2). Observe in figure 7.11(b) how $B3$ at t_3 forwards the 'ra' received in p1 only to p2 as the variable rcvdFromNeigh indicates.

The neighbor eventually receives the 'ra' carrying the confirmation that the

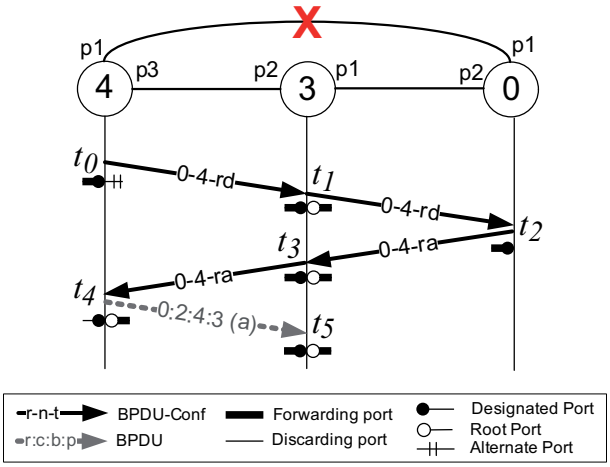
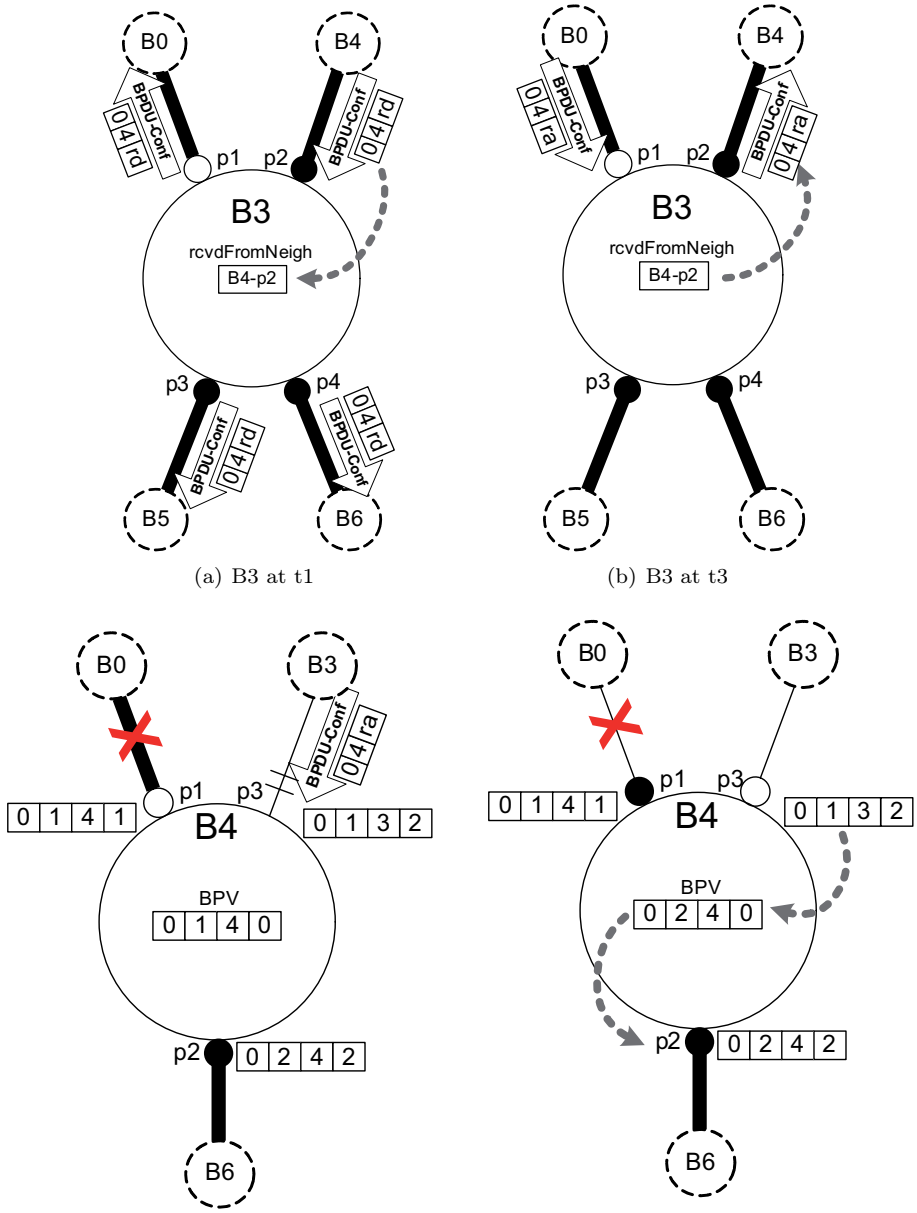


Figure 7.10: Exchanged Messages in the recovery of a single link failure

Root is alive ($B4$ at t_4 in 7.10 and detailed in 7.11(c)). The neighbor interprets the reception of this message as the final confirmation that the Root is alive and hence it can proceed with the tree reconfiguration that was hold at t_f. Since the Root availability is confirmed, it is safe to select the Alternate port in $B4$ as the new Root port. The confirmation mechanism is terminated at this point and the tree recovery proceeds as the common RSTP with the dissemination of BPDUs including the new information. In the example, $B4$ just sends an agreement back to $B3$ and a BPDU with the new cost to $B5$.

Note that the delay introduced by the confirmation mechanism in the event of a single link failure is one round-trip time between the Root and the Neighbor through the new path. Although this value can vary depending on the physical topology, highly connected networks with enough redundancy (specially around the Root) only experience an additional delay of 4 (2+2) hop delays.



(a) B3 at t1

(b) B3 at t3

(c) B4 at t4

Figure 7.11: Detailed node diagrams in the recovery of a single link failure

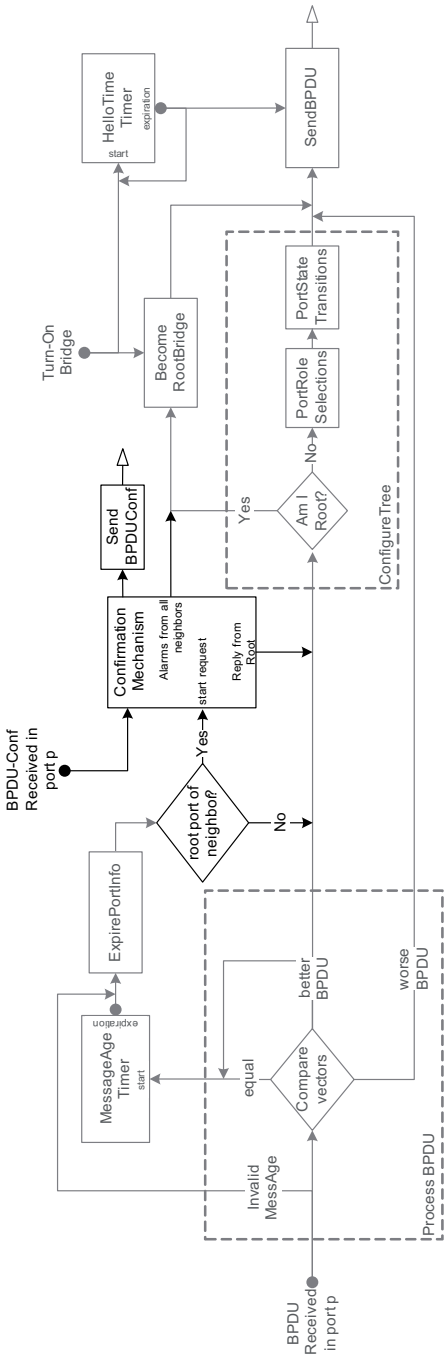


Figure 7.12: General diagram of the RSTP-Conf operation (RSTP-Conf updates in black; original RSTP operation in grey)

C.1 BPDU Received in port p		***/* The only change in the common BPDU processing is to update the neighbor variables */***
8.	... ConfigureTree()	*/* Previous instructions are the same as in RSTP */*
9.	numNeigh = rcvdNumNeigh	*/* The number of neighbors is distributed by the BPDUs from the proper Root of the tree */*
10.	if (PPV[b] of Root port==BPV(r))	*/* Checking whether a local node is a neighbor of the tree is straightforward. The bridge is neighbor if the bridge field in the PPV of the Root port is the same than the Root field of the BPV. In other words, if the parent bridge through the Root node is the believed Root, then the bridge is a neighbor */*
11.	iAmNeigh = true	
12.	else	
13.	iAmNeigh = false	
14.	Clear rcvdFromNeigh[]	*/* The entries in the array that manage the received 'rd' is cleared because the reception of a better BPDU indirectly confirms that the Root is alive, hence any 'confirmation' mechanism triggered is now terminated */*
15.	if(BPDU is proposal) ...	*/* Following instructions are the same as in RSTP */*

C.2 BPDU-Conf Received in port p	***/* It is triggered when a BPDU-Conf is delivered to the RSTP-Conf instance by the LLC inferior */***
1. neigh = Neigh field in the received BPDU-Conf 2. type = Type field in the received BPDU-Conf	*/* These are temporary local variables that store the Neigh and Type fields of the received message */*
3. if (type is 'rd')	*/* An 'rd' BPDU-Conf is a request from the neighbor to the Root and hence must be flooded */*
4. if (rcvdFromNeigh[neigh] is empty)	*/* An 'rd' message from a particular neighbor is only processed the first time it is received */*
5. if (local bridge is not Root) 6. rcvdFromNeigh[neigh] = p 7. for all ports p' except p 8. SendBPDUConf (p', neigh, 'rd')	*/* Non-Root bridges that receive an 'rd': mark the port where it is received is stored because a later 'ra' must be forwarded to this port. And flood the message so it finally reaches the Root, if this is still alive, or it reaches the rest of nodes so they can reboot the tree, if the Root has failed */*
9. else 10. SendBPDUConf (p, neigh, 'ra')	*/* When the alive Root bridge receives the 'rd' message, it replies with an 'ra'. This message is forwarded bridge by bridge back to the neighbor that initiated the confirmation mechanism */*
11. if ('rd' received from all neighbors) 12. BecomeRootBridge()	*/* If the Root fails hence no one issues the 'ra', all nodes eventually receive the 'rd' from all neigbors as a signal to reboot the tree. In this situation each node arises as Root as in a cold start scenario */*
13. if (type is 'ra')	*/* An 'ra' BPDU-Conf is a reply from the Root to the neighbor and hence must be forwarded to it */*
14. if (neigh is different than BPV[b]) 15. SendBPDUConf (rcvdFromNeigh[neigh], neigh, 'ra')	*/* The 'ra' message received in a non-Root bridge results into a Forwarding to the single port that indicates the array rcvdFromNeigh (because it is the port where the 'rd' was received). */*
16. else 17. ConfigureTree()	*/* The 'ra' message that reaches the neighbor that originated it represents the confirmation from the Root that it is alive and hence it can use the information in the Alternate ports. Therefore it can safely reconfigure the tree and re-elect Root roles based on the information stored in any port */*
18. Clear all entries in rcvdFromNeigh[]	*/* The confirmation mechanism is terminated in those nodes that received the 'ra' message */*

D. MessageAgeTimer expiration (or physical failure detection) in port p	***/* The change introduced by RSTP-Conf is only to distinguish between a neighbor loosing its connectivity to the Root, and the rest of cases */***
1. ExpirePortInformation(p)	*/* The information stored in the port is not valid anymore and hence it cannot be trusted. RSTP-Conf introduces the changes in the actions that follow this port expiration */*
2. if (port p is Root port && IamNeigh is true) 3. rcvdFromNeigh[BPV[b]] = p 4. for all ports p' except 5. SendBPDUConf(p', BPV(b), 'rd')	*/* A neighbor that looses its connectivity to the Root triggers the confirmation mechanism: marks itself in the array that accounts for the neighbors that started the procedure,and floods the 'rd' messages asking for Root availability. This might result in (a) a response from the still alive Root, or (b) a tree reboot once all 'rd' from all neighbors have been received */*
6. else 7. ConfigureTree()	*/* If the bridge detecting the failure is not a neighbor, it proceeds as in RSTP and directly reconfigures the tree because of the change in the port information. Note that RSTP-Conf actually delays this reconfiguration until it has received the confirmation from the Root */*

E. BecomeRootBridge()	***/* The changes on this routine result in the initialization of the confirmation variables */***
1. numNeigh = local number of ports 2. IamNeigh = false 3. Clear all entries in rcvdFromNeigh[]	*/* In RSTP, the Root is the only node that really knows how many ports it has so it sets this value in numNeigh (which is distributed to other nodes within common BPDUs). Also, the Root is not the neighbor of any Root (hence IamNeigh is reset). Finally, a Root clears all the entries of the array rcvdFromNeigh because the Root, itself, is alive */*
4. SetPV(BPV, BridgeID, 0, BridgeID, 0) ...	*/* Following instructions are the same as in RSTP */*

I.2. SendBPDUConf (port, neigh, type)	***/* The information encoded in the BPDUConf represents a request/reply from/to a neighbor that triggers the confirmation mechanism */***
1. Set Root field of the BPDUConf equal to BPV[r] 2. Set Neigh field of the BPDUConf equal to argument 'neigh' 3. Set type field of the BPDUConf equal to argument 'type'	*/* The Root field of the BPDU-Conf message is set to the Root believed by the bridge (BPV[r]), the encoded Neigh and Type fields encoded in the message are the value passed as arguments because these depend on the situation where this procedure is called from */*
4. Send the BPDU-Conf to port	*/* Once the fields are filled, the BPDUConf is sent to the outgoing port indicating the the ethernet encapsulation details to the inferior sub-layer (LLC) */*

Figure 7.13: Pseudo-codes of the RSTP-Conf operation

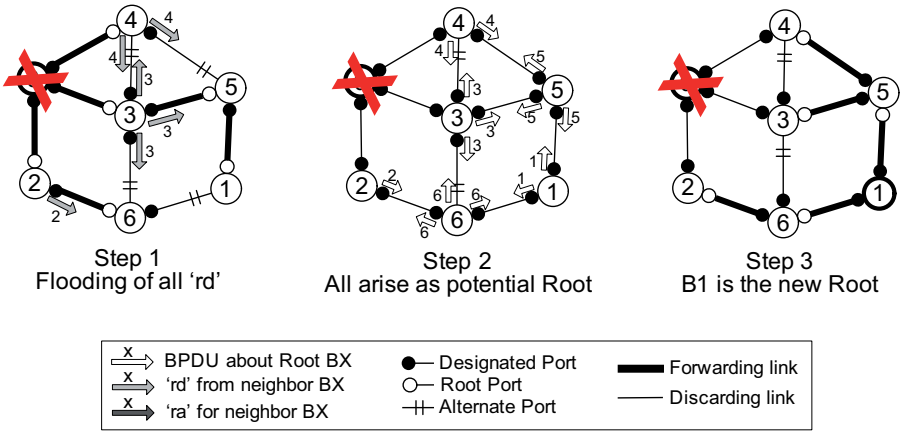


Figure 7.14: Sequence of steps of the confirmation mechanism to recover from a Root failure

7.4.3 Tree reboot after the Root failure

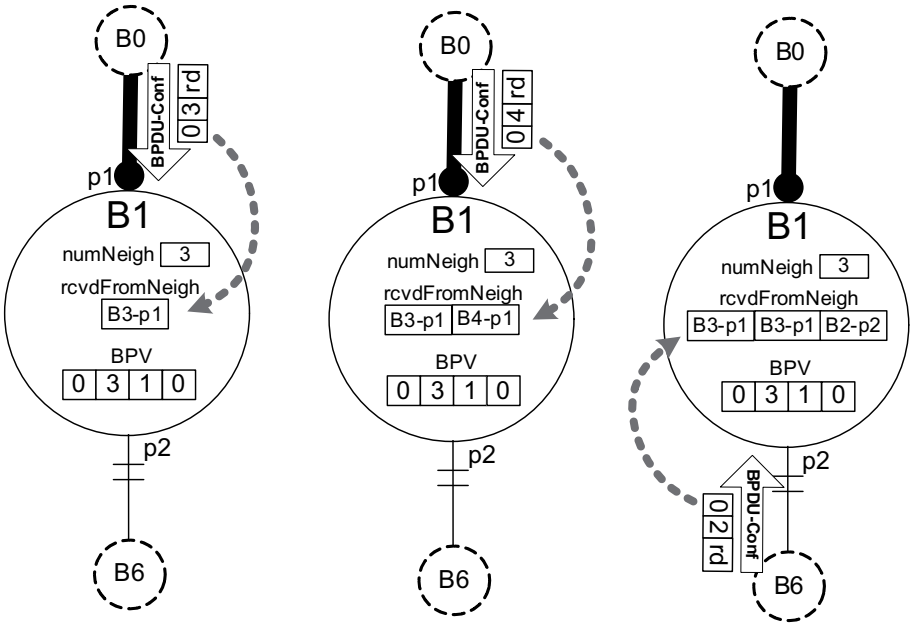
The scenario where the Root bridge fails is different from the previous single link example because (1) the neighbor will never get the 'ra' message because there is actually no Root that sends it, and (2) all neighbors flood their own 'rd' each one notifying that they believe that the Root is unavailable (step 1 in figure 7.14).

Figure 7.15 includes the node diagrams showing the evolution of $B1$ evolving towards the reboot of the tree ($B1$ will actually be the new Root). First observe that $B1$ does not directly notice that the Root fails at t_f. $B1$ receives the 'rd' messages of each one of the neighbors (figure 7.15(a)) and keeps track of the alarms adding the information to the array rcvdFromNeigh.

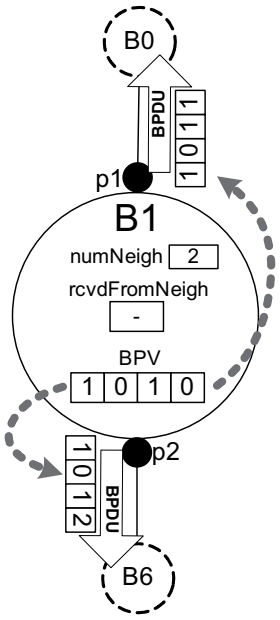
All the network nodes eventually receive the 'rd' messages issued by the neighbors. Note that any node receives as many 'rd' messages as the number of neighbors of the Root. Since all nodes know the exact number of neighbors (disseminated with the common BPDUs and set in line 3 of block C.1), it is straightforward to implement a condition that is met when all 'rd' have been received (line 11). Receiving the 'rd' messages from all neighbors implies that the Root has really failed hence a node can propose itself as new Root (line 12). Accordingly, $B1$ realizes there is no Root and arises itself as Root (figure 7.15(b)).

At this point, all network nodes consider that the old active topology is not updated any more and hence they all arise as potential new Roots (step 2 in figure 7.14). This results into a global reboot leading to a situation similar to a cold-start. After all the exchange of BPDUs, a new node is elected as Root ($B1$ in step 3) and the tree is configured again.

Note that a Root failure recovery requires (1) the time to flood the 'rd' messages plus (2) the time to disseminate the BPDUs of the new Root. In the worst case where $B1$ is the furthest node from the old Root $B0$, the recovery time is one round-trip delay.

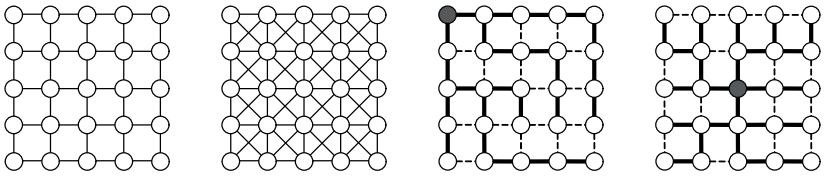


(a) $B1$ receives alarm from $B3$, $B4$ and $B2$



(b) $B1$ arises as Root because all 'rd' alarms have been received

Figure 7.15: Root failure recovery from $B1$ perspective



(a) grid4 (b) grid8 (c) Root in corner of grid4 (d) Root in center of grid4

Figure 7.16: Two-dimensional mesh topologies of degrees 4 (*grid*4) and 8 (*grid*8)

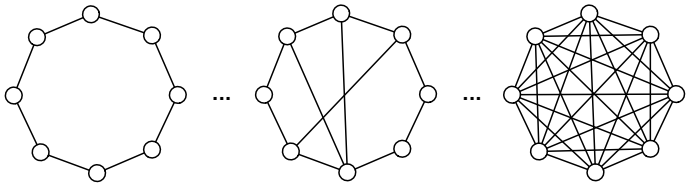
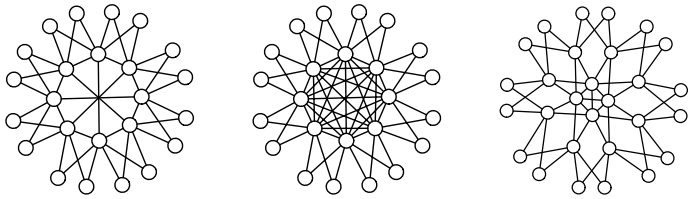


Figure 7.17: Ring-based topology of increasing connectivity (or average node degree).



(a) mesh-heavy (b) mesh-full (c) two-tier

Figure 7.18: Realistic structured topologies

7.5 Performance evaluation

This section includes the performance evaluation of RSTP and RSTP-Conf under different failure situations. As in previous evaluations, we run the protocols in the different network topologies of figures 7.16, 7.17 and 7.18.

The Root is located in a given position and the rest of bridges are configured with random BridgeIDs. We take as a reference for the time to transmit, propagate and process a BPDU the study in [30] that assumes a delay of 1.33ms per message. The value of MaxAge is set to the number of nodes in the network, unless the experiment details otherwise, in order to ensure the propagation of a full wave-front. Only BPDU messages are simulated and no user traffic is modeled unless otherwise stated. The modeled failure detection mechanism is the immediate physical failure detection where the affected ports directly realize that the link has failed and notify the protocol instance.

The performance evaluation focuses on the time to construct the tree and the amount of information exchange required for such action. Since the following

evaluation focuses on studying the protocol recovery in the event of failures, we measure the *Recovery Time* (RT) as the time that the protocol takes to reconstruct the tree. Note that we use a different notation for the convergence time in a recovery scenario (RT) than for the convergence time in a cold start (CT). Although the concept of convergence also applies to the recovery phase, we find RT more adequate and it allows easily distinguishing them. We still refer to the message overhead as MO.

Each one of the following sections focuses on a different scenario in order to evaluate different particularities of the protocols performance. Sections 7.5.1 and 7.5.2 provide the response of both protocols in front of Root failures so as to evaluate (1) the consequences of count-to-infinity in RSTP and (2) how RSTP-Conf avoids it. Finally, section 7.5.3 includes a broader perspective of the analysis and shows the performance in the event of non-Root failures such as single links or other nodes.

7.5.1 Characterization of count-to-infinity consequences in RSTP

This analysis focuses on the evaluation of RSTP recovering from a Root failure. Since it suffers count-to-infinity, this section actually evaluates the consequences of this effect under different scenarios and different topologies. In each execution we locate the Root in a different node, and we fail it.

7.5.1.1 Recovery time

The first tests use the ring-based topology of increasing connectivity degree. Plot in figure 7.19 shows the measured RT. The left vertical axis measures the time in seconds. Note that in this scenario for all topology sizes, RSTP takes the order of tens of seconds to recover. The reason of this low performance is the appearance of the count-to-infinity problem, as described in section 7.1, which vanishes after the appearance of several deadlocks that stop the message transmission during 6 seconds (3xHelloTime). Observe the right y-axis of figure 7.19 is measured in number of deadlocks that occur during the count-to-infinity (up to 19 in a 50 nodes network of degree 10).

The stopping of message transmission due to deadlocks can also be observed if we further analyze a single execution. Figure 7.20(a) shows a timeline with the transmitted BPDUs in a network of 20 nodes and degree 5. Note that each bin is divided into the messages about the failed $B0$ (dark grey), the messages about the new Root $B1$ (light grey) and the messages about the rest of nodes (white). First observe that the appearance of deadlocks causes the not-continuous transmission of messages. Each one of the high peaks in the timeline represents the different rounds of the count-to-infinity until BPDUs are discarded because they reach the MaxAge limit. The first bin, or round, represents the messages that are sent right after the failure is detected and the count-to-infinity is triggered (figure 7.20(b) shows the detail). A total of 1843 BPDUs are transmitted until the first deadlock temporarily stops the transmission. Most of these messages (1760) are

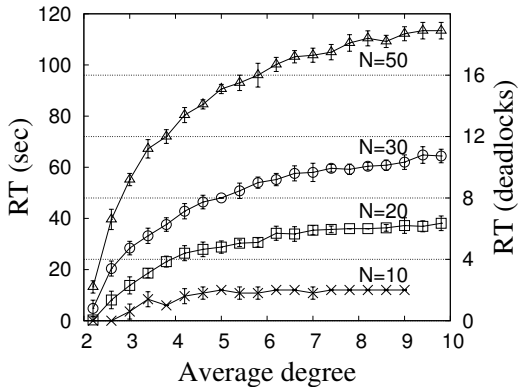
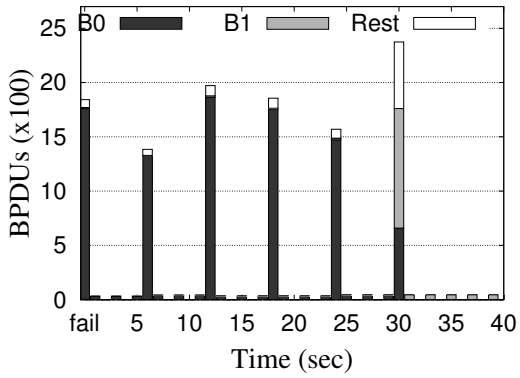


Figure 7.19: RT of RSTP in a Root failure scenario in the ring-based topologies

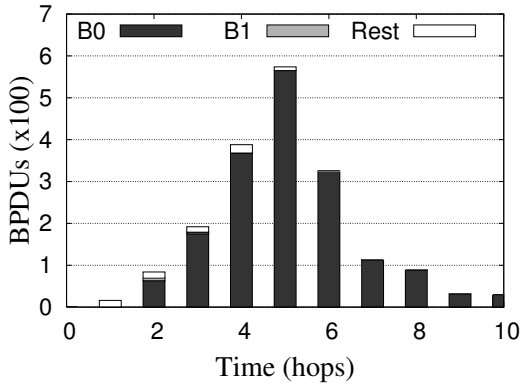
BPDUs about the failed Root and only a small amount (86) are BPDUs about other nodes, including $B1$, that try to arise as Roots. These non-$B0$ messages are transmitted during the early stages of this first round but they are quickly "eaten" by the stale $B0$ BPDUs. This is why in all rounds the percentage of $B0$ messages over the total is very high.

The looping of all these BPDUs during each round leads to the deadlock in a link where both sides assume that are Root ports and hence do not forward BPDUs. When this deadlock occurs, nodes remain in a transient but stable situation with a virtual Root (see section 7.1). This results into periodical transmission of BPDUs in those nodes that believe in the virtual Root (see the small bins between the high message peaks in figure 7.20(a)).

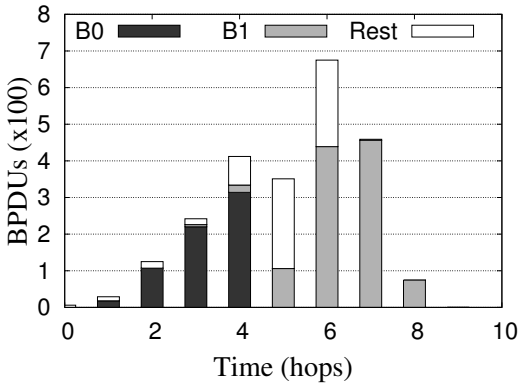
The silence in both sides of the deadlocked link with the deadlock prevents the existence of BPDUs in this link and hence any possibility to be reconfigured except after waiting long enough for BPDUs, when the timer 3xHelloTime expires. The timeout allows considering outdated the port state and is being reset losing the information of the failed Root. At this point the port is reconfigured to Designated and the algorithm continues normally but still under the count-to-infinity effect because other ports in the network still have in their state the failed Root with a MessAge smaller than MaxAge indicating that it is valid. Figure 7.21 shows a similar timeline but indicating the value of the MessAge field in the transmitted BPDUs. Observe that the BPDUs about $B0$ (circles) reach a MessAge of 7 in the first round. When the deadlock is released the count-to-infinity continues with the state information of ports at their current value of the MessAge. For example, the MessAge re-starts at 7 and reaches 11 in the second round before a new deadlock occurs. This situation is repeated at every round until MessAge of the different BPDUs reach MaxAge (20 in this execution). This happens in the fifth round of messages at second 30 after the failure. Observe in figure 7.21 how the MessAge of $B0$ BPDUs reach 20, and are definitely removed hence eradicating the count-to-infinity effect. At this point, the BPDUs of several nodes arising as potential Roots are exchanged creating a similar situation as in



(a) Timelines of BPDUs about B0, B1, and the rest



(b) Detail of first deadlock



(c) Detail of last deadlock

Figure 7.20: BPDU timelines of a Root failure recovery in a ring-based topology with 20 nodes and degree 5

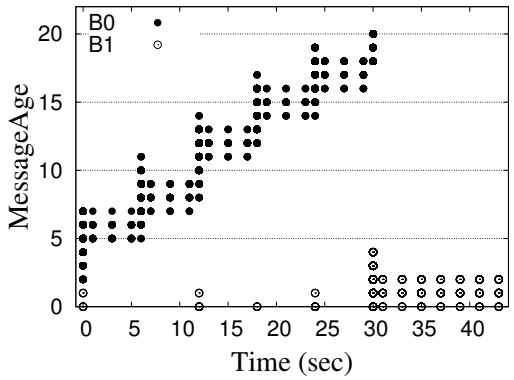


Figure 7.21: MessAge field evolution in a Root failure recovery in a ring-based topology with 20 nodes and degree 5.

Table 7.4: RT in seconds after a Root failure recovery in ring-based topologies with an average degree of 10 and different MaxAge values

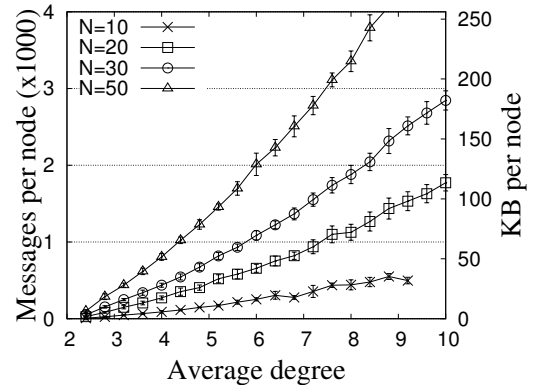
	MaxAge			
Size	*10*	*20*	*30*	*50*
N=10	12	31	52	93
N=20	14	38	61	108
N=30	-	39	64	111
N=50	-	-	67	113

a cold start. This is shown in the last round (figure 7.20(c)) where the wave-front of the new Root $B1$ ends up spanning the entire network winning the encounters with wave-fronts started at other nodes.

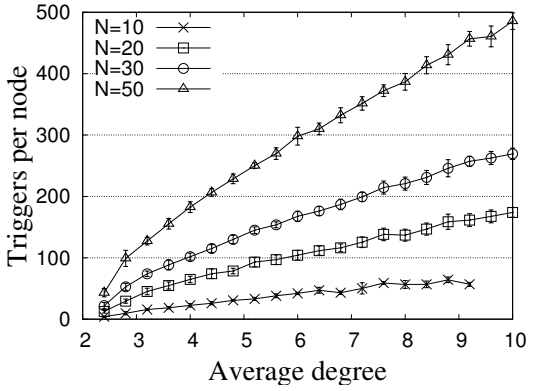
The value of MaxAge drives the convergence time in the Root-failed scenario due to the counting to infinity. This can be clearly observed in the plot of figure 7.19 as large networks, with larger MaxAge, recover the failure after more deadlocks. This dependence is confirmed fixing the network size and varying the value of MaxAge. We have done the same experiments for all the previous topology sizes and for each MaxAge. Table 7.4 shows the average RT for the topologies with an average degree of 10 (note it corresponds to the last point on the lines of figure 11). If MaxAge is really high it becomes the dominant factor for the RT as it oscillates from 93-113 compared to N that varies from 10 to 50.

7.5.1.2 Message overhead

The count-to-infinity effect clearly affects the recovery time of the protocol after a Root failure, but it also causes a significant increase of the message overhead. The looping BPDUs are removed only after they reach a MessAge equal to MaxAge



(a) MO_{node}

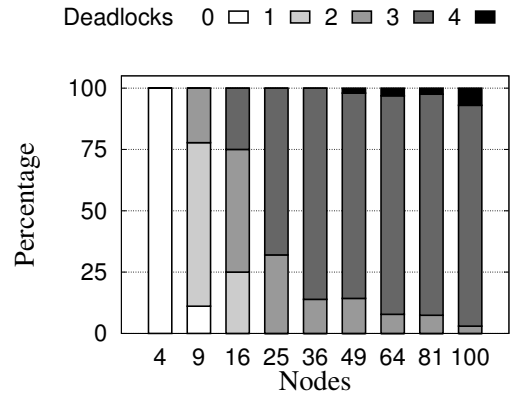


(b) TR_{node}

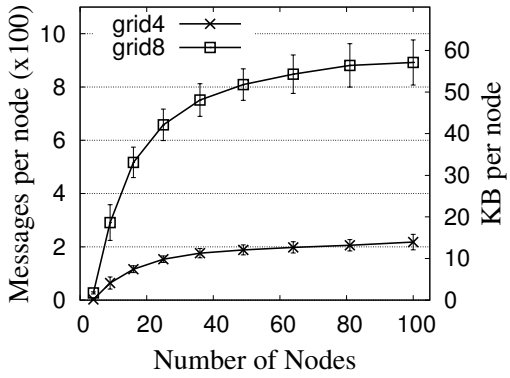
Figure 7.22: Message overhead of RSTP in a Root failure scenario in the ring-based topologies

and this results into a large quantity of messages that are forwarded during all rounds and deadlocks. The plot in figure 7.22(a) shows the MO_{node} measurements for the ring-based topologies of varying connectivity degree. The amount of messages transmitted clearly grows with the size of the network and with the average node degree. The results clearly show that a count-to-infinity situation results into a very large overhead (i.e. 2500% of the MO_{node} required for a cold-start). The only alleviation is that these messages are really spread in time in different rounds between deadlocks and hence the instant overhead on the nodes is reduced.

The increase of the message overhead due to count-to-infinity not only affects the capacity of the links. If we study the amount of triggers, shown in the plot of figure 7.22(b), we can observe that each node receives a high quantity of



(a) RT



(b) MO_{node}

Figure 7.23: Performance of RSTP in a Root failure scenario in the two-dimensional grid topologies

messages that trigger a tree recalculation. This implies that during a count-to-infinity the processing power of the nodes is affected by the need to recalculate the tree every time a looping BPDU that updates is received. Also note that these recomputations are useless because the messages are looping and the protocol is just waiting for the MessAge to reach MaxAge and terminate the count-to-infinity behavior.

7.5.1.3 Variability due to topological aspects

In order to evaluate the effect of the topology size we run experiments in the grid4 topologies of different sizes and we keep the MaxAge constant to 20 (the default value). Note that the maximum degree in the grid4 topology is less hence

20 hence the MaxAge does not limit the wave-front construction.

Figure 7.23(a) includes the RT evaluation of RSTP for different sizes of the grid and locating, and failing, the Root in all possible locations. The bars indicate the percentage of experiments experiencing different number of deadlocks for each grid size. For example, in the grid of 9 nodes, 1 run did not experience any deadlock (white), 1 deadlock occurred in 6 of the runs (lightest grey), and 2 deadlocks in 2 runs (grey). Observe that the amount of deadlocks required grows with the size of the grid. However the main factor is the connectivity level: the larger the network, the higher the average node degree (increasing from 2 to 3.6). Note that in this case we are keeping MaxAge constant in all executions hence in these experiments we can see the effect of other parameters such network size or node degree. Observing the MO_{node} per node in figure 7.23(b) we can also confirm that larger networks, that are slightly more connected, require a higher amount of messages due to count-to-infinity.

The experiments involving the different topologies are shown in figure 7.24. In this case we have also configured a constant MaxAge of 20 as it is the recommended value by the standard and it is large enough for the tested topologies to construct the entire wave-fronts in the cold start. The CT performance in figure 7.24(a) shows that the grid4 topology recovers in 3 deadlocks while all the rest do it after 6 deadlocks. The reason is the low average degree of the grid4 topology of 64 nodes (3.5) compared to the rest (from 4.4 to 6.5). However, observing the MO for the different topologies in figure 7.24(b), we see that there is a correlation between the amount of messages and the average degree. This confirms the growing MO observed in the ring-based topologies of figure 7.22(a).

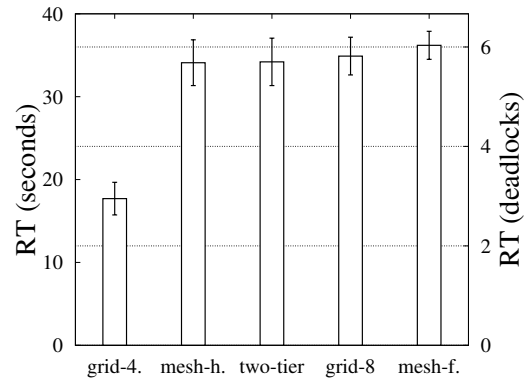
7.5.2 Avoiding count-to-infinity with RSTP-Conf

This section includes the simulation results of RSTP-Conf in the scenarios where the Root fails. The objective is (1) to show that count-to-infinity does not appear and (2) to characterize the performance of the protocol in terms of recovery time and message overhead. Note that in this section we do not include the evaluation of RSTP-Conf in a cold-start scenario, or network start-up, because the protocol performs equally as RSTP. In not failure scenarios, the confirmation is not triggered and hence there are no performance differences with the original RSTP.

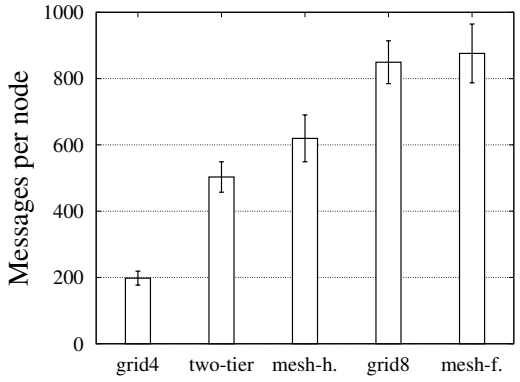
7.5.2.1 Node degree variation

In this analysis we focus on the evaluation of the protocol in a scenario where RSTP suffers count-to-infinity so as to show that RSTP-Conf avoids the effect. As we have already seen in the previous section, the node degree is one of the important factors that determine the influence of count-to-infinity. Therefore we first use the ring-based topologies of increasing connectivity for an easier comparison.

As shown in figure 7.25(a), RSTP-Conf outperforms RSTP in terms of RT because it does not suffer the delay due to count-to-infinity. Observe that in



(a) RT

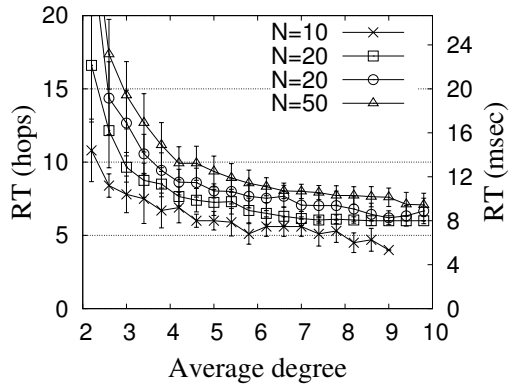


(b) $\mathrm{MO}_{\mathrm{node}}$

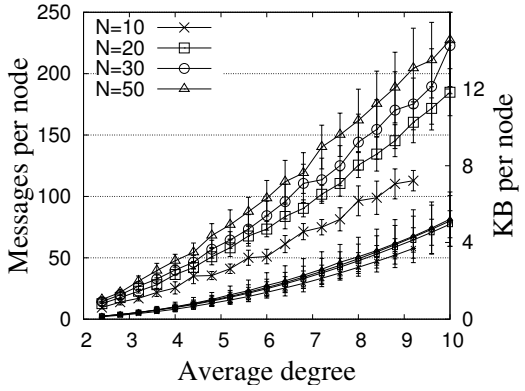
Figure 7.24: Performance of RSTP in a Root failure scenario in various topologies

this case the RT is measured in hop delays in the left axis and in milliseconds in the right axis. While RSTP requires several deadlock delays to recover (tens of seconds), RSTP-Conf is able to reduce the RT time to a few hop delays (order of milliseconds). In addition, the RT for RSTP-Conf decreases with the redundancy level because the diameter of the topology is smaller for higher connectivity. Since the proposed protocol is based on a flooding technique, its recovery time actually depends on the time to span the network and therefore on the topology diameter.

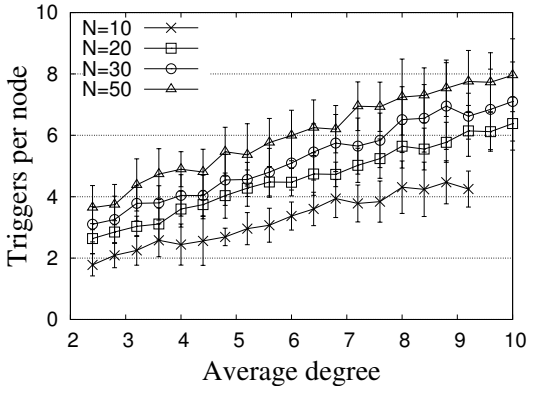
Figure 7.25(b) shows the message overhead ($\mathrm{MO}_{\mathrm{node}}$) over the connectivity degree for the different topology sizes. Observe how the amount of messages grows with the connectivity degree because in flood-based techniques the number of messages is proportional to the number of links. However, the values obtained in RSTP-Conf stay one order of magnitude below those in RSTP. The reason for such difference is the existence of the count-to-infinity behavior in RSTP.



(a) RT



(b) MO_{node}



(c) TR_{node}

Figure 7.25: Performance of RSTP-Conf in the ring-based topologies

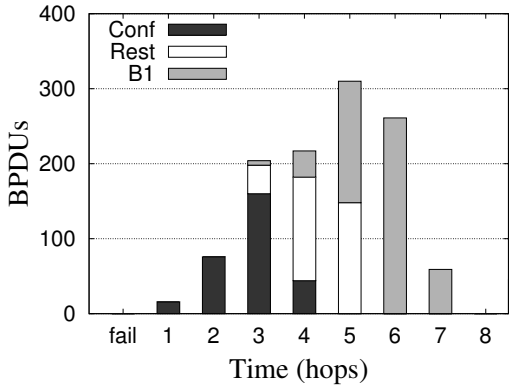


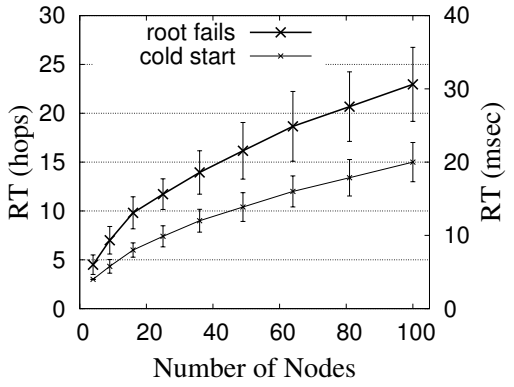
Figure 7.26: Timeline of received BPDUs during a Root failure recovery with RSTP-Conf

Nevertheless, the amount of messages transmitted by RSTP-Conf is not that low because it forces a global reboot where the Root needs to be elected from scratch. Since the count-to-infinity behavior does not occur, and hence there are no BPDUs to be removed after the maximum number of hops, changing the value of MaxAge does not effect in the performance of RSTP-Conf.

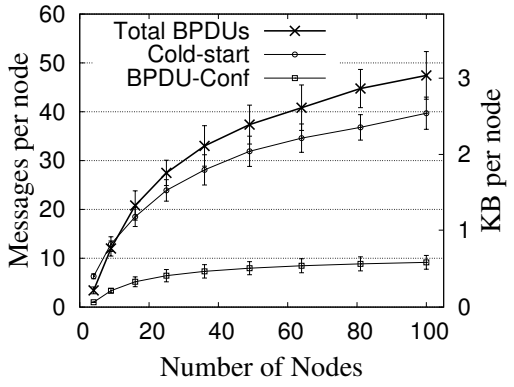
An additional observation is the amount of BPDU-Conf messages ('rd' floodings) required. The thin lines in the plot represent the number of these messages for each degree. The ratio between the confirmation messages over the total amount of BPDUs decreases for larger networks. For example in the topology of 10 nodes the confirmation messages represent up to 51% of the total amount of messages in some cases. Differently, in the network of 50 nodes this ratio decreases to 33%.

The plot in figure 7.25(c) shows the average amount of triggers per node that occur during the recovery. Although the Root recovery triggers an entire tree reboot, the amount of tree recalculations per node remains low. As in the cold-start case, the triggers slowly grow with the connectivity degree because of the flooding nature of the protocol that makes it proportional to the number of links in the topology.

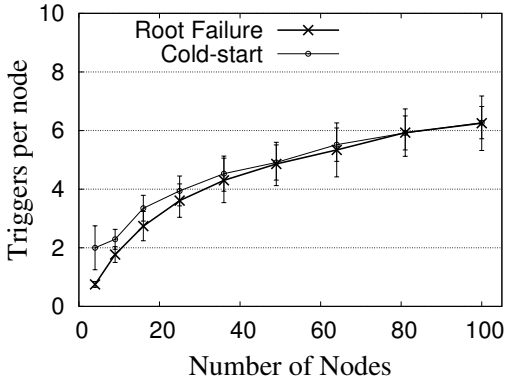
A further analysis looks into a single experiment and observes the particularities of the transmitted BPDUs. Figure 7.26 shows the time-line of the total number of BPDUs in the topology with an average node degree of 4 (20 added links). Note that the different colors represent different types of BPDUs: black for BPDU-conf, grey for common BPDUs announcing $B1$ as Root, and white for common BPDUs announcing other nodes as Root. Observe that right after the failure (until hop 4) the BPDU-Conf messages originated in the neighbors are flooded until these reach all nodes and trigger the tree reboot. At this point, a behavior similar to a cold start occurs because each node starts their own wave-front, but only the one from the new Root $B1$ remains and beats all the others (last 5 hops). Observe the similarities with the timeline of BPDUs during a cold



(a) RT



(b) MO_{node}



(c) TR_{node}

Figure 7.27: Performance of RSTP-Conf in the grid topologies

start in figure 6.12 in section 6.3 once the confirmation mechanism has triggered the tree reboot.

7.5.2.2 Topology size variation

This second analysis presents the performance of RSTP-Conf in the two-dimensional grid topologies in order to evaluate the effect of topology size while keeping the node degree.

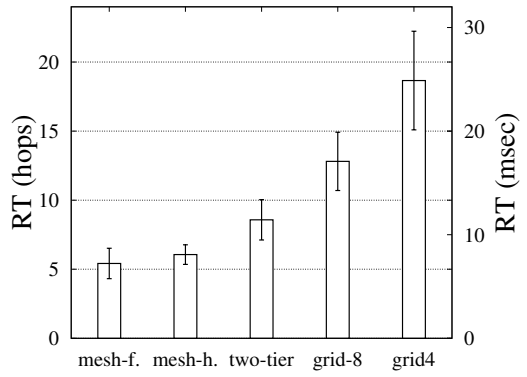
Plot in figure 7.27(a) shows the RT observed in the grid4 topologies (each execution fails the Root in a different location). For all topology sizes, the RT in RSTP-Conf remains in the order of milliseconds as it takes a several hop delays to reconstruct the new tree. It grows with the network size because it depends on the network diameter. This is due to the flooding nature of the mechanism: a first flooding of the confirmation messages and a second one of each node that arises as potential Root. Therefore, the total CT really depends on the location of the new Root $B1$. For example, in the case of 100 nodes the diameter is 18 hops. In the experiment where $B0$ and $B1$ are located in opposite corners the total required CT is 36 (twice the diameter) because of the two floodings happening one after the other. If the two nodes are located next to each other, $B1$'s flooding to become Root happens together with the flooding of confirmation messages and the total CT is reduced to 18 (one diameter). The plot also includes the cold-start CT as a reference line to compare the emulated global reboot after the Root failure and the initial construction of the tree. The two lines evolve similarly and the only difference is the delay due to the confirmation mechanism that neighbors initiate.

The MO_{node} performance of RSTP-Conf in the grid topologies is shown in the plot of figure 7.27(b). Root recoveries in RSTP-Conf result in a similar scenario to a cold-start, hence the performance is similar to it. The amount of transmitted messages per node grows with the size because the topology is larger, which means there are more potential Roots after the global reboot that result in more wave-fronts that advance. The difference with the cold-start is that in the Root failure recovery the Root neighbors first trigger the confirmation mechanism and flood the 'rd' messages. Nevertheless, these represent a small percentage of the total BPDUs. For example, in the grid of 100 nodes the Root failure recovery procedure transmits 10 BPDU-Conf messages per node of the total amount of BPDUs (48).

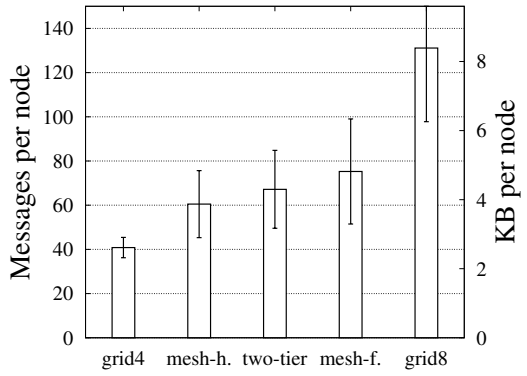
In this experiments the messages that trigger a tree recalculation almost match the triggers in a cold-start (as shown in plot of figure 7.27(c)). The reason is that the metric TR does not account for the confirmation messages. Therefore a cold-start and the tree reboot that occurs in a Root failure recovery in RSTP-Conf result in a similar amount of tree reconfigurations.

7.5.2.3 Other topologies

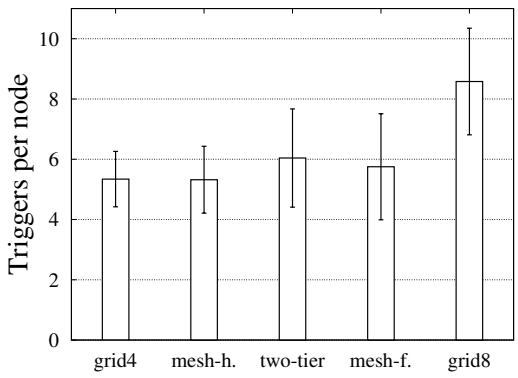
In this last analysis of RSTP-Conf in the event of Root failures we present the performance of the protocol in the different topologies included in figure 7.28. The



(a) RT



(b) $\mathrm{MO}_{\mathrm{node}}$



(c) $\mathrm{TR}_{\mathrm{node}}$

Figure 7.28: Performance of RSTP-Conf in the various topologies

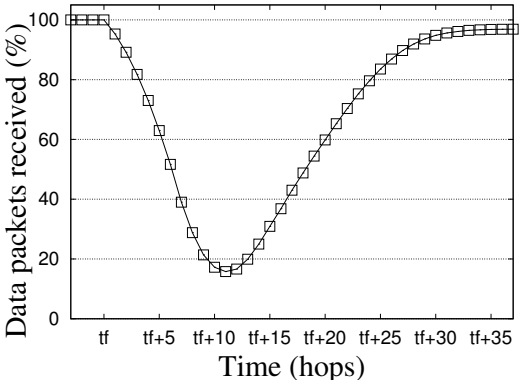


Figure 7.29: Traffic received by all nodes during the Root failure recovery

plot in figure 7.28(a) shows the RT measured in hop delays and in milliseconds. The different columns have been sorted from the smallest to the largest diameter. This results confirm that Root recovery time in RSTP-Conf depends on the network diameter because of the two floods that occur (originated at neighbors; originated at the new Root) and that the difference with the initial cold-start is the delay due to the confirmation mechanism. The message overhead is shown in the plot of figure 7.28(b). In this case the columns have been sorted by the average node degree, with the lowest value in the first column. These results also confirm that the cold-start triggered by the alarms sent by neighbors is the main factor. The amount of the initial BPDU-Confs also remains small in comparison to the common BPDUs transmitted while the new tree is constructed. Finally, figure 7.28(c) shows the messages that trigger a tree recalculation for the different topologies (the columns are also sorted by degree). As in the previous scenarios, the amount of triggers is similar to those in a cold-start and remains between 6 and 8% of the total amount of BPDUs transmitted.

7.5.2.4 Traffic outage

The last evaluation of the Root failure recovery with RSTP-Conf studies the outage that data communications experience during the recovery process. The failure of the Root results into the disconnection of some branches, hence the loss of connectivity between some nodes. As in previous similar tests, we test the connectivity level by configuring each node to send broadcast data packets of 1000 bytes every 10us. The percentage of packets received represents the level of active connectivity at that instant.

Plot in figure 7.29 shows the time-line of received packets in the grid4 topology of 64 nodes where the failing Root is located in one corner. The horizontal axis is measured in hops (t_f indicates time of failure) and the vertical axis in percentage of received packets by all nodes (100% represents 4032 messages). RSTP-Conf takes around 30 hops to recover the maximum connectivity level. This delay is because (1) the propagation of the neighbor alarms, and (2) the dissemination

of the wave-front of the new Root. Note that after the recovery the maximum connectivity does not reach 100% because the failed Root is neither sending nor receiving data.

7.5.3 Performance in the event of non-Root failures

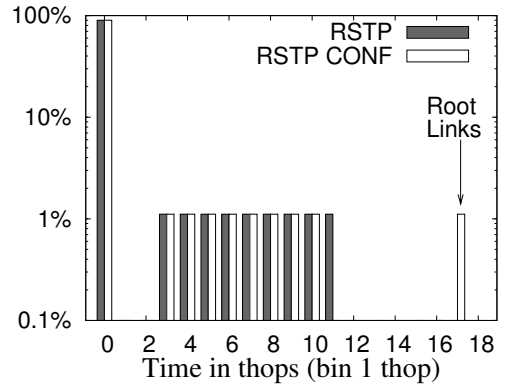
The analysis presented following aims at studying the performance of RSTP and RSTP-Conf in non-Root failure situations. Although the Root failure is one of the critical scenarios, specially in RSTP as it leads to a count-to-infinity, failures of other devices such as non-Root nodes or single links are actually the most common in realistic networks.

7.5.3.1 Link failures

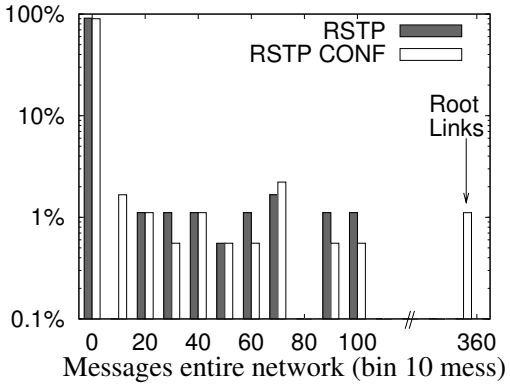
For these tests we use the grid4 of size 100 and we locate the Root in one corner. We first observe the protocols recovery in the event of all possible single link failures. That is, we run as many experiments as links in the topology, and we fail a different connection in each execution. In the grid of 100 nodes this results into 180 different scenarios. Plots in figures 7.30(a) and 7.30(b) contain the histograms for RT and MO obtained from the runs of all possible link failures (note that we now measure the message overhead in the entire network). The white bins refer to RSTP and the grey ones to RSTP-Conf. The most remarkable observation is that 90% of the link failures result into an immediate recovery (RT of 0 and MO below 10) in both protocols. This is either because the link did not belong to the tree, 40% of the cases, or because an Alternate port becomes Root port, the remaining 60%.

A few link failure cases require the protocols to exchange some more messages and hence delay the recovery (10% of the link failures in RSTP and 9% in RSTP-Conf). In these cases the protocols transmit from 10 to 100 BPDUs and the convergence time spans from 3 to 10 hop delays. The closer to the Root the failed link is located, the longer the recovery because the new topology needs to be disseminated to the entire network.

In the scenarios described so far, both protocols provide the same performance because they really operate equally. The only difference is observed when a link connected to the Root fails and a confirmation procedure is triggered in RSTP-Conf. This is why RSTP-Conf experiences a CT of 17 hop delays with 355 exchanged messages in approximately 1% of the link failures (actually, the two links connected to the Root in the corner). Note that the additional 6 hop delays really represent the delay introduced by the 'rd' and 'ra' propagations. Also note that even if the confirmation is triggered, this situation does not result into a global reboot because only one neighbor issues the 'rd' messages. This is a situation where RSTP outperforms RSTP-Conf because the confirmation mechanism is triggered but it results in a false alarm that just delays the recovery.



(a) RT

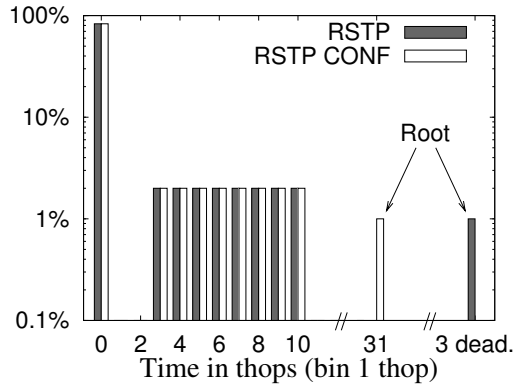


(b) MO (entire network)

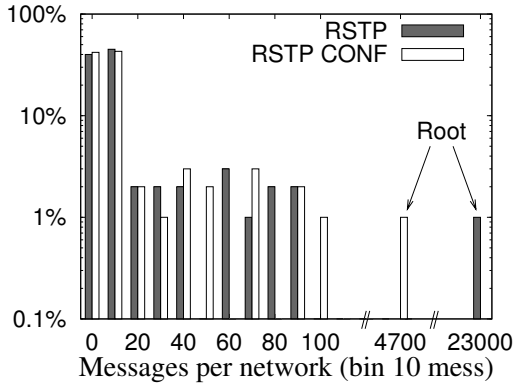
Figure 7.30: Histograms of RT and MO recovering from all links possible failures in the grid of 100 nodes and the Root in a corner

7.5.3.2 Node failures

The histograms in figures 7.31(a) and 7.31(b) show the protocols performance when recovering from of all possible node failures. We use the same grid topology as in the previous case but we fail a different node in each run (resulting in 100 different scenarios). Similarly to the link failures, both protocols immediately recover from most of the failures (83% observe a RT of 0). However, the number of messages is slightly higher and in general the immediate recoveries exchange 20 messages or less. There is also a set of failures, 16% in both protocols, that result into a longer recovery: RT from 3 to 10 hop delays, and MO from 20 to 100 messages. Note that the recovery from non-Root node failure is quite similar to the non-Root link failure, with more messages because a node failure really represents a failure up to 4 links at the same time.



(a) RT



(b) MO (entire network)

Figure 7.31: Histograms of RT and MO, entire network, recovering from all node failures in the grid of 100 nodes and the Root in a corner

Nevertheless, the difference in performance arises when the Root fails. RSTP takes up to 3 deadlocks while RSTP with confirmation stays in the order of hop delays (31). In terms of MO there is a difference of two orders of magnitude: 230000 for the RSTP and the count-to-infinity; and 4700 for the RSTP-Conf triggering a global reboot. The Root failure represents the worst-case scenario in both protocols. But the effect is much harmful in RSTP because there is no way to stop the count-to-infinity situation.

§ 8. RSTP-SP: Shortest Path Bridging Keeping the Distance-Vector Approach

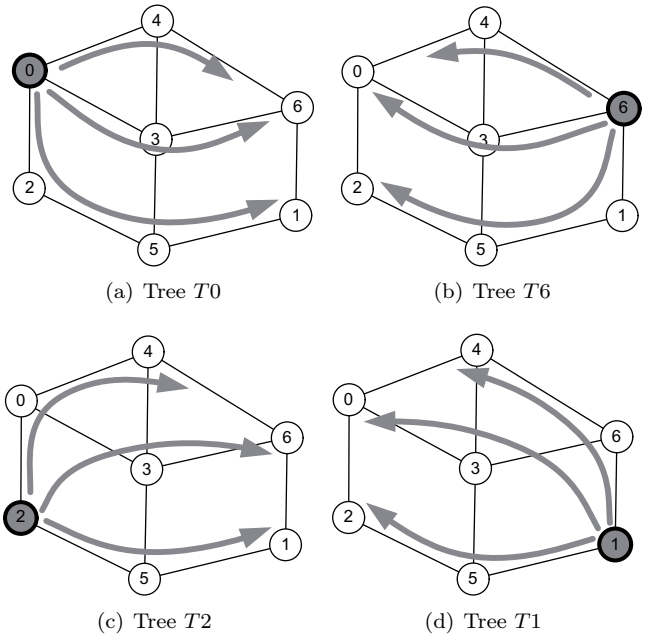
The IEEE 802.1aq Shortest Path Bridging is the current step in the evolution of the control protocols for Ethernet Bridging. The link-state protocol proposed by IEEE represents a big change in the core of the bridge architecture as it substitutes the, successful so far, distance-vector family of spanning tree protocols.

We however think that there is an alternative to provide optimal paths keeping the original distance-vector protocols. From now on we refer to such extension as RSTP-SP. The idea is to reuse the RSTP protocol introducing the necessary changes to construct the multiple trees that deploy optimal paths. Since RSTP is a distance-vector protocol that focuses on building a single tree, it makes sense to extend its applicability in order to build several trees. However, it is not as simple as executing the standardized RSTP for each tree. The framework with several trees implies a revision of some aspects such as the symmetry challenge also faced by the link-state SPB protocol. This chapter describes the implications of extending RSTP to be used in a Shortest Path Bridging environment.

The chapter first describes the different aspects where protocol updates are required: section 8.1 first discusses the changes due to the use of several tree instances; the distance-vector solution to the symmetry challenge is addressed in section 8.2; the changes required in the event of failure detection are described in section 8.3; and section 8.4 elaborates on the Root failure consequences (count-to-infinity) and details the implementation of the confirmation mechanism of RSTP-Conf. Finally, section 8.5 evaluates the RSTP-SP protocol comparing it to the IEEE proposal SPB.

8.1 Deployment of parallel instances

RSTP-SP is based on executing the extended RSTP as many times as trees to construct. This is, in a network with N nodes, N trees are configured by N instances of the single tree protocol. These instances are completely independent and run at different levels. Each one of these tree instances has the Root in a different node, hence each one of the instances of the single tree protocol starts if a different node (the Root of that tree). In terms of message propagation, this means that the BPDU wave-fronts generated at each node independently advance in different planes and hence never encounter any other better wave-front that removes them.



(a) Tree $T0$ (b) Tree $T6$

(c) Tree $T2$ (d) Tree $T1$

Figure 8.1: Propagation of wave-fronts started at $B0$-$B2$-$B6$-$B1$

Figure 8.1 shows some of these wave-fronts (actually those originated at $B0$, $B2$, $B1$ and $B6$). Note that each one of the diagrams occurs at the same time but at a different level. This allows all the wave-fronts to propagate the entire network; hence each instance eventually configures in all nodes the tree that is rooted at the bridge that originates such wave-front. The execution of different protocol instances implies that bridges need to distinguish the management of the different trees. This is why (1) the nodes keep one set of variables for each tree (as described in section 8.1.1), and (2) the processing of the different events also distinguish between different trees (section 8.1.2).

8.1.1 Per-tree variables

The use of parallel tree instances requires each tree to manage its own messages and, in consequence, each node needs to store separated information per tree. Tables 8.1 and 8.2 include the bridge and port variables used in RSTP-SP. Note that these are the same that are described for RSTP in section 3.1. And the only difference is that most of them are kept for each tree. In other words, and from an implementation point of view, RSTP-SP stores arrays of the RSTP variables where the array index is the tree identifier. Note that the the BridgeID and the HelloTimeTimer are the only two variables that are not replicated per tree. The former is shared among all trees because it is a unique identifier of the node and hence does not change between different trees. The latter is a timer at bridge

Table 8.1: Bridge Variables

Name	*Description*
BridgeID	The unique bridge identifier is shared by all the tree instances.
Bridge Priority Vector (BPV[r:c:pa:p])	Each tree has its own BPV. The only difference with the BPV in RSTP is that the Path-array substitutes the Bridge field.
HelloTimeTimer	The transmission of the refreshing BPDUs is controlled by a central timer that is shared among all the trees. At timeout, the Designated ports of each tree send their corresponding BPDUs.

Table 8.2: Port Variables

Name	*Description*
State	Each tree stores en entire set of port variables. Note that all these variables are used to manage the condition of the port within the tree; hence a different group is stored and managed separately by each tree. Within each corresponding tree instance, each variable has the same role and operation as in RSTP.
Role	
Port Priority Vector (PPV[r:c:pa:p])	
MessAge	
MessageAgeTimer	
Proposal/Agreement	

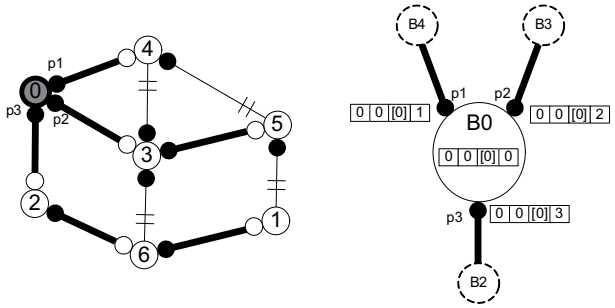
level that triggers the distribution of refreshing BPDUs in each one of the trees at the same time.

Figure 8.2 shows a more detailed version of the previous example of the propagating wave-fronts. It shows the already constructed tree instances $T0$, $T6$, $T2$ and $T1$. First observe that the same port has a different role and state in different trees. For example, $B3$ has $p1$ as Root port in the tree instance $T1$, but it selects $p3$ as Root port in $T6$.

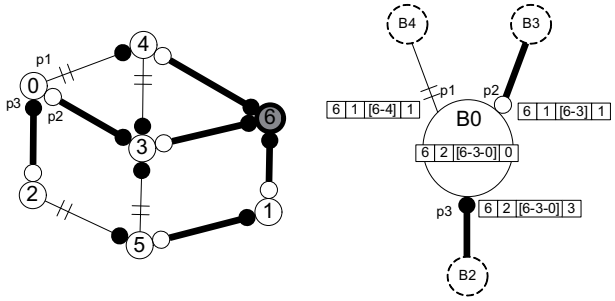
The priority vectors, both BPV and PPVs, are also stored independently for each tree. Figure 8.2 also includes the details of the vectors that node B0 stores for each one of the tree instances. Since $B0$ is the Root of $T0$, it has only Designated ports and all the vectors contain information about itself (e.g. [0:0:0:0] in the BPV). Differently, observe in the other trees that $B0$ has Root and Alternate ports that store information about the neighbor (for example $p1$ in $T6$ or $p3$ in $T2$). As in RSTP, the vectors in RSTP-SP are used to select the port roles and hence to decide the shape of the tree branches. The only difference is the third element of the priority vector where the BridgeID field has been substituted by the Path-array. Section 8.2 provides a more detailed description of this change and how it is used.

8.1.2 Per-tree event processing

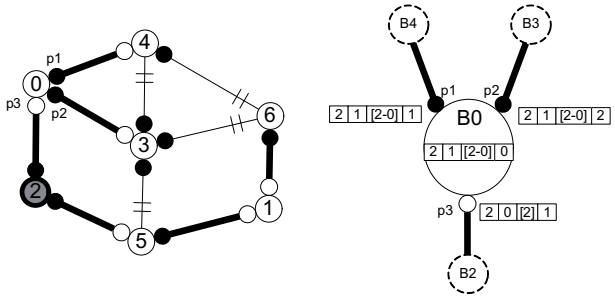
Since the trees are independent, the protocol operations interact with one tree instance at a time. Each event that triggers a protocol operation (for example a BPDU reception) applies to one of the tree instances and the operation executed uses the variables of that particular tree.



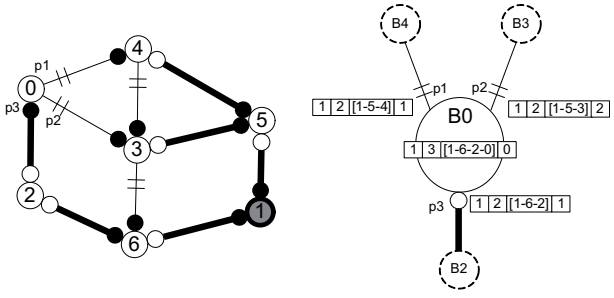
(a) Tree $T0$



(b) Tree $T6$



(c) Tree $T2$



(d) Tree $T1$

Figure 8.2: The nodes store an independent set of variables for each tree instance

Most of the original RSTP procedures described in section 3.2 are kept the same with the only difference that they are run applying to a particular tree instance (as shown in table 8.3). First, the process of the BPDU received event is only updated to distinguish between different tree instances. For example, when a BPDU belonging to tree T is received, the priority vectors taken into account are those belonging to instance T. Second, the MessageAgeTimer Expiration event only applies to the single that has detected the lack of refreshing BPDU. Third, the initialization of the trees in the procedure BecomeRootBridge stays the same as in RSTP with the only difference that the operation is applied to the tree instance where the node is Root (its own tree). At the beginning, the rest of trees in that node are inactive so there is no initialization required assuming all variables are cleared. Fourth, the recalculation of port roles (PortRoleSelection) and port states (PortStateTransitions) when the one of the trees is being reconfigured applies to only this tree. Fifth, the SendBPDU procedure also applies to one of the tree instances at a time. This implies that the BPDU sent includes the priority vector belonging to that particular tree. And finally, the auxiliary subroutines that edit port variables also apply to one of the tree instances.

Table 8.4 shows those events that also do not change in operation but are applied to all tree instances. First, the dissemination of refreshing messages triggered by HelloTimeTimer expiration applies to all active trees. Therefore, all Designated ports of all trees send periodical messages every HelloTime. And second, the direct physical detection of a port failure results into a removal of port information and the consequent tree reconfiguration in all trees. This means that a port failure detection results into as many tree reconfigurations as trees.

8.2 Selection of symmetrical trees

The link-state protocol of SPB addresses the symmetry challenge applying the path selection policy based on the path-array. The link-state approach can easily compare multiple shortest paths and select the right one because each node is aware of the entire physical topology and locally executes the path calculation. The challenge for a distance-vector protocol like RSTP-SP is how to derive the path-array in order to apply the same rule.

8.2.1 The path-array in the distance-vector environment

In distance-vector protocols, the topology distribution and the path calculations are merged together in the same operation. That is, the paths are selected step by step as the information is distributed. Therefore, in our case, the distribution of BPDUs must be used to obtain the path-array. The solution is straightforward because the BPDUs flow from the Root to the leaves following the tree branches. Hence the BPDUs can keep track of the visited nodes and construct the path-array step by step as they are propagated. By adding the path-array in the BPDUs, nodes become aware of the entire paths to reach the Root and hence they can make the right selection adding the same rule as SPB (sorting the path-array and comparing item by item). Observe in figure 8.3 how in the BPDUs

Table 8.3: Events and procedures, with the same operation as in RSTP, that only apply to one of the trees (passed as argument)

Event	*Description*
BPDU about tree t received in port p	Each BPDU only applies to the tree indicated in the Root field of the priority vector. The processing of this message hence only considers the information stored in the local priority vectors, and reconfigures variables, of such tree.
MessageAgeTimer of tree t expiration in port p	Each port keeps a different timer for each tree. Therefore, the expiration of a timer only applies to that tree.

Procedure	*Description*
BecomeRootBridge(tree)	A bridge can only become Root of one of the trees. Actually, this procedure is only in the event Turn on Bridge when at start-up each node is configured as the Root of its tree.
PortRoleSelection(tree)	The update of the port variables state only applies to one tree at a time. This is, when the tree T is reconfigured, the port variables of tree T are reselected.
PortStateTransition(tree)	
ExpirePortInformation (port, tree)	
SendBPDU(port, tree)	The transmission of BPDUs is also managed independently for each tree because the BPDU contains the corresponding priority vector.
SetInactiveDesignatedPort (port, mAge, tree)	The auxiliary subroutines that edit the port variables also apply to the particular tree that is passed as argument.
SetDesignatedPort (port, mAge, tree)	
SetRootPort (port, tree)	
SetAlternatePort (port, tree)	
PortActivationHandshake (port, type, action, tree)	

Table 8.4: Events with the same operation as in RSTP that apply to all trees

Event	*Description*
HelloTimeTimer expiration	The refreshing of BPDUs is triggered from the bridge perspective and it results in the periodical transmission of the BPDUs through the ports that are Designated in the different trees.
Physical Failure Detection in port p	The direct physical detection of a port failure triggers the removal of the port information and a reconfiguration in all trees.

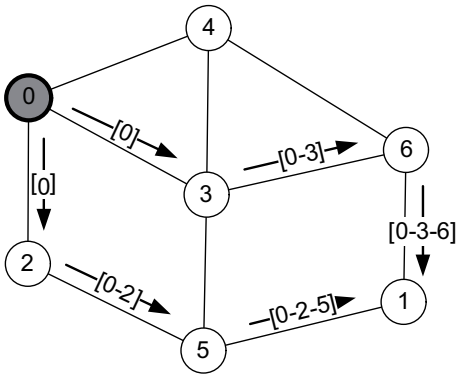


Figure 8.3: The BridgeID of the traversed bridge is appended t the path-array of the BPDUs

originated by $B0$, hence to construct $T0$, the path-array filled is updated with the traversed nodes hop after hop. This is how $B1$ receives the path-arrays 0-3-5 and 0-2-6, and finally selects the branch through $B6$ as the connection to the Root.

Since the path-array is used to decide among different equal-cost paths, it is stored in the priority vectors instead of the BridgeID field. Note that it is not really a substitution as the BridgeID is actually the last element of the path-array. The four fields of a priority vector hence become:

- The *Root* (r) indicates the BridgeID of the Root node and it is interpreted as the identifier of the tree.
- The *Cost* (c) is the distance to this Root.
- The *Path-array* (pa) is the sequence of BridgeIDs in the path starting at the Root (being the last element the BridgeID of node that owns the vector).
- And *Port* (p) stores the PortID of the port that owns this vector.

For an easier reference to the vector fields, from now on we refer to them with the notation *r:c:[pa]:p*.

8.2.2 Changes in the protocol operation

The BPDUs record the path-array as they are propagated; therefore the frame format needs to be extended in order to encode the additional path-array field. Table 8.5 shows the contents of an RSTP-SP BPDU. In order to add the path-array we require 8 bytes for each BridgeID in the path-array of the priority vector and 2 bytes for a field including the path-array length (*LengthPathArray*). Adding a variable size field in the BPDU message might result into an implementation limitation because of too large frames. However, the path-vector is always a shortest-path and the upper bound of its length is the network diameter. With

Table 8.5: RSTP-SP BPDU Frame Format

Name			*Description*	*Bytes*
Eth. encaps.	*SA*		The BPDU encapsulation is the same than in RSTP, hence the Ethernet MAC layer does not need to be updated.	6
	DA			6
	Length/Type			2
	LLC			3
BPDU Fields	*Protocol id.*		Protocol identification fields indicating that the BPDU received corresponds to an RSTP-SP instance and it is a common BPDU.	2
	Version			1
	Message Type			1
	LengthPathArray		The number of elements included in the path-array is encoded in the message so the frame parsers correctly interpret the field values.	2
	Priority Vector	*Root*	The Root field is interpreted as the identifier of the tree.	8
		Cost	The Cost field still contains the distance to the Root of this tree.	4
		Path-array	Different BridgeIDs (each one 8 bytes) are encoded in the third field of the priority vector.	8+8 +...
		Port	The Port field still contains the portID of the transmitter port.	2
	MessAge		The rest of parameters remain unchanged and keep the same description and usage as in the original RSTP.	2
	Flags	*Role*		1
		Prop.		
		Agreem.		
	MaxAge			2
	HelloTime			2
	ForwardDelay			2
Frame Check Sequence				4

a maximum MTU of 1500 bytes for Ethernet frames, the maximum length for the path-vector (and for the network diameter) is 178 elements; long enough for current provider networks [67].

Since the structure of the priority vector has changed, the tie-breaking mechanism described for RSTP needs to be updated as well. The updated *CompareVectors()* procedure shown in block K in figure 8.6 shows the new tie-breaking rule used in RSTP-SP. First, the Root field is not used as the initial tie-breaking checking. Instead, it represents the tree identification (i.e. the tree is identified by the BridgeID of its Root). Note that within a tree instance there are only messages about that particular tree (there is only one wave-front). This means that the vector comparisons are always between vectors of the same Root, and hence comparing the Root fields would always result into jumping to the comparison of the Cost as the second element. Second, if the costs are equal the path-array is compared. The rules to decide if a path-array is better than another are the same than those described fro SPB in section 5.1 (sort the identifiers and select the array with lower values in the elements). The last tie-breaking step remains the same comparing the port numbers.

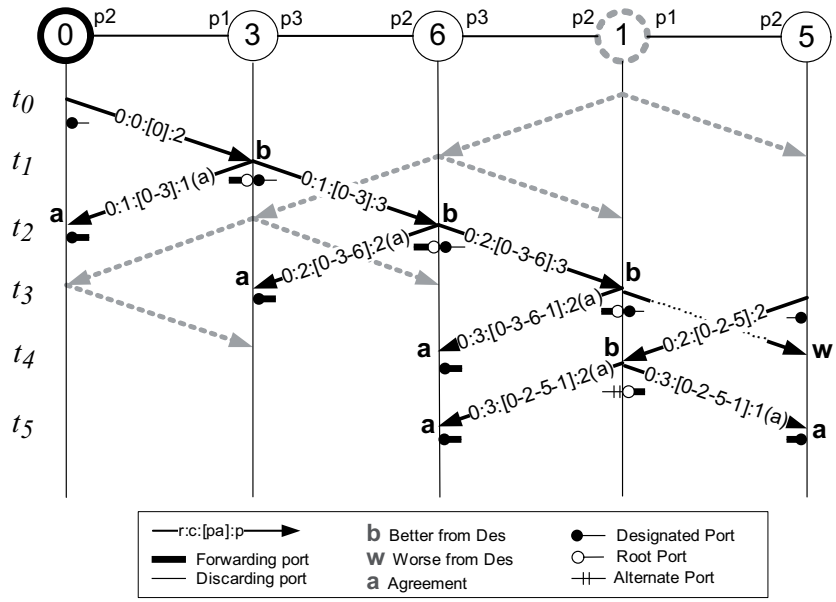
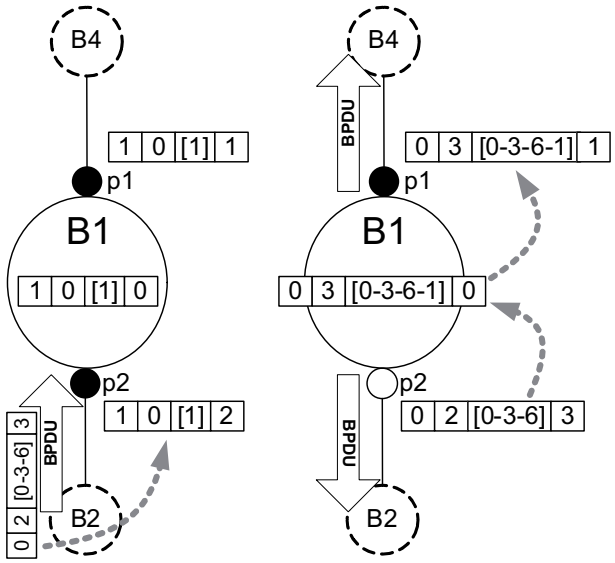


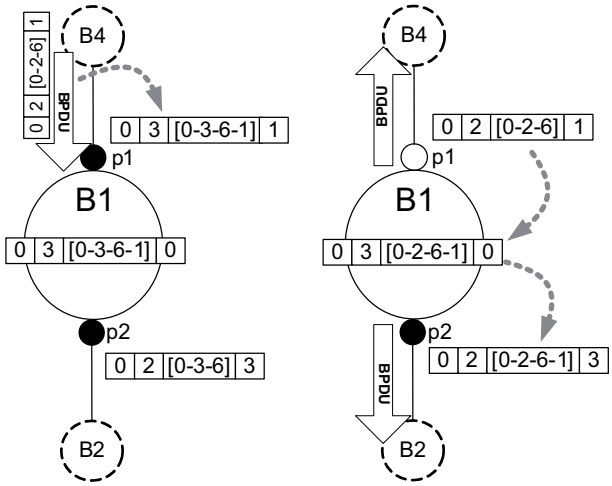
Figure 8.4: Exchanged messages during the initial trees configuration with RSTP-SP.

The diagrams already shown in figure 8.2 show the detailed configuration of bridge and port variables of $B0$ in the four tree instances shown. First observe how the priority vectors stored by $B0$ for the tree $T0$ (its own tree) contain information about itself telling that $B0$ is the Root at cost 0. The path-array in this case just contains $B0$ in the BPV and in the PPV of all Designated ports. As in RSTP, the comparison of priority vectors still decides the port roles and hence the tree structure. For example in $T2$, $B0$ selects $p3$ as Root port because it has the lowest cost (0 received from $B0$). In this case the path-array is not used in the selection of the port roles. Instead, in the tree $T6$, $B0$ receives a cost of 1 both from $B3$ and $B4$. In this case the path-array needs to be used to decide among the two possibilities. The path-array received by $B3$ (6-3) is considered better than the one received from $B4$ (6-4) because , once sorted, $3<4$. Similarly in $T1$, $B0$ selects $p3$ as Root port because $2<3<4$.

The effect of the new tie-breaking rule can also be seen in the diagram of message exchanged shown in figure 8.4. This example relates to the wave-front propagation and the creation of the different tree branches if the path-array is used in the tie-breaking. Note that only two wave-fronts are shown in the diagram for simplicity reasons ($T0$ and $T1$). Unlike RSTP, observe how $T1$ is not removed by $T0$ when these encounter and both wave-fronts continue their propagation until spanning the entire network. A particular difference with RSTP is the use of the path-array in selecting the roles for example in $B1$ (see detailed node diagrams in figure 8.5). At t_3, $B1$ receives the BPDU about $B0$ 0:2:[0-3-6]:3 and sets its $p2$ as Root port and $p1$ as Designated. At t_4, it receives on $p1$ the BPDU 0:2:[0-2-5]:3



(a) Time t_3



(b) Time t_4

Figure 8.5: $B1$ vectors configuration of tree $T0$ at t_3 and t_4 during the initial trees configuration with RSTP-SP

from $B5$. The last received vector is first compared to the port vector 0:3:[0-3-6-1]:2. The received cost is lower, hence the received is considered better. When selecting the port roles, the best priority vector is the stored in p1 because its path-array, 0-2-5, is better than $p2$'s path-array, 0-3-6. Therefore $p1$ is elected as Root port. Note this step is different than in the RSTP because the path-array is now being considered and it is concretely this Root port selection that allows for symmetrical trees to be configured. While RSTP selects $p2$ of $B1$ as Root port (because of the lower BridgeID), RSTP-SP chooses $p1$ (because of the better path-array).

The convergence time of RSTP-SP in a cold start scenario is the same as in RSTP because the wave-fronts of BPDUs are all propagated from each Root to the furthest node. While in RSTP there is only one wave-front that fully propagates, in RSTP-SP all of them reach the entire network. However, since this propagation is done in parallel at the same time, the analytical characterization states that $CT_{RSTP-SP} = CT_{RSTP}$ (from section 6.2).

F. ConfigureTree(tree)	***/* The update in the tree configuration relates to the deactivation of a tree when the node would have decided to arise as Root of another tree */***
1. Select the Root Bridge	*/* Selecting the Root Bridge is still done looking for the lowest BridgeID the local node is aware of. */*
2. if (Im the Root Bridge)	*/* Arising as Root in another tree is not possible and hence this instance is deactivated and the cost in the port vectors are set to infinity to represent a lack of connectivity. */*
3. Deactivate tree	
4. Set cost to infinity in all port vectors	
5. PortRolesSelection ()	*/* The configuration of the tree (port roles and states) is done as in RSTP. If infinity costs are set to the ports, these are elected as Designated and Discarding and the disseminated BPDUs distribute the 'no-connectivity' signal, with the infinity costs, so other nodes also deactivate the tree. */*
6. PortStateTransition ()	
7. for each port p	
8. SendBPDU(p, tree)	

K. CompareVectors (A, B)	***/* It returns whether vector A is considered BETTER, EQUAL or WORSE than B according to the tiebreak rules */***
1. if (A.cost < B.cost) ||	*/* A is better than B if it has a lower cost, */*
2. (A.cost == B.cost && A.path-array < B.path-array) ||	*/* or same cost and a lower bridge, */*
3. (A.cost == B.cost && A.path-array == B.path-array && A.port < B.port)	*/* or same cost, same bridge and a lower port */*
4. return 'BETTER'	
5. else if (A.cost == B.cost && A.bridge == B.bridge && A.port == B.port)	*/* A and B are EQUAL if all fields are the identical */*
6. return 'EQUAL'	
7. else	*/* A is WORSE than B if none of the previous conditions are met */*
8. return 'WORSE'	

Figure 8.6: Pseudo-code of the updated operation in RSTP-SP

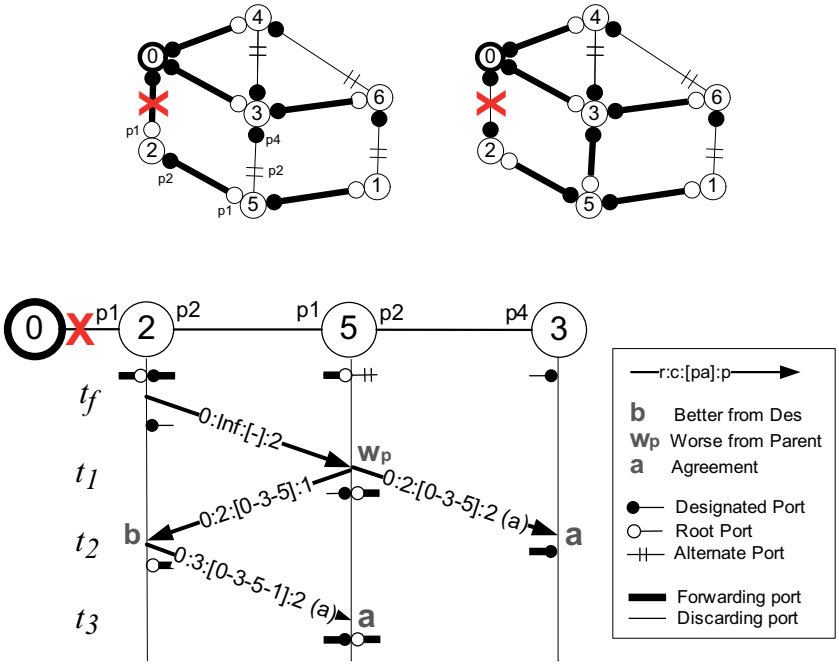


Figure 8.7: Single Link Failure recovery in RSTP-SP

8.3 Failure recovery

When a node in RSTP-SP detects a failure in a port, it removes the port vector and reconfigures all the trees to match the changed information in the vectors of the failed port. The fact of using one tree routed at each node introduces a new aspect to consider. In RSTP, the node with the lowest BridgeID is elected as the unique Root of the tree. If this node fails, the protocol recovers choosing the node with the second lowest BridgeID as new Root (note this would happen after the count-to-infinity). However, in RSTP-SP each node is the Root of its own tree and no other node can arise as Root of another's tree. This relates to the situation when a node detects a Root port failure in one of the trees and cannot arise as new Root of that tree. In RSTP-SP, this node then considers that the tree is inactive and notifies a no-connectivity situation to the rest (the updated ConfigureTree procedure introduces this functionality in lines 2-4 in block F of figure 8.6). This is simply done by sending BPDUs with an infinity cost. These BPDUs are normally processed by the receiving nodes and are seen as common BPDUs with a very large cost. Note that issuing the messages with infinity (or very large) cost is an effective solution to disseminate the co-connectivity situation without changing the process of BPDU processing.

Also observe that the concept of an infinity cost needs to be translated into a practical number. The easiest way is to understand as infinity the maximum cost value allowed by the Cost field in the BPDU frames. This field id 4-bytes long,

so the maximum value it can convey, and hence the practical representation of infinity, is $2^{32} - 1$ (4294967295).

Figure 8.7 shows an example where the link between $B0$ and $B2$ fails. It is important to note that the reconfiguration is executed for each tree instance. This example relates to the recovery within the instance of the tree rooted at $B0$, but similar actions occur in other trees. $B2$ detects the failure in port $p1$ and since it has no Alternate ports, it deactivates the tree and sends a BPDU with the infinity cost. This BPDU is received by $B5$, and considered better because it comes from the parent (note the infinity cost is stored in $p1$ of $B5$). When $B5$ configures the port roles, it selects $p2$ as the new Root port, an old Alternate, and $p1$ as Designated. $B5$ sends a BPDU with the new bridge cost back to $B2$ and an agreement to $B3$. Note that the infinite cost BPDU is not propagated any more because the information in the Alternate port of $B5$ is used to recover the tree. When $B2$ receives the BPDU from $B5$, it configures again activating the tree of $B0$ through the Root port at $p2$. As in RSTP, the failure of a single link is recovered almost immediately as soon as a node with an Alternate port is found.

8.4 Node failures and count-to-infinity

A recovery from a node failure does not differ from RSTP in terms of protocol operation and behavior. If a non-Root node fails, the recovery is as quick as in the single link failure case. If instead the Root of a tree fails, a count-to-infinity behavior is experienced within the tree instance of the failed Root. Unfortunately, a node failing in this multiple tree framework always results into a Root failure in one of the trees. This means that any node failure leads to (1) a count-to-infinity situation in one of the trees and (2) a quick recovery in the rest.

Nevertheless, the particularity is that after a Root failure, this node does not inject any more traffic into its own tree. If there is no communication, there is no urgency to reconfigure the tree because communications from this node cannot be established until the failed Root is repaired. Therefore the network recovery is not really affected by the long convergence time because of to the count-to-infinity in the dead tree. However, the network is really affected by this behavior in terms of message overhead. The count-to-infinity within the dead tree generates BPDUs that loop around and get stooped from time to time due to the deadlocks occurred. These messages are sent from bridge to bridge and therefore they reduce the processing power of the nodes and the available capacity of the links.

An alternative is to implement the confirmation mechanism in RSTP-Conf in order to completely avoid the count-to-infinity behavior. Since RSTP-SP executes a protocol instance for each tree, the required changes to implement the confirmation functionality are basically those already described in section 7.4. These mainly affect the management of the confirmation variables and the processing of the confirmation BPDUs that are sent by the Root neighbor ('rd') and those replied by the Root itself ('ra'). From now on, we will refer to this

Table 8.6: Operational updates as described in RSTP-Conf to introduce the confirmation mechanism in RSTP-SP

Additional Bridge Variables	*Description*
numNeigh	One set of these confirmation variables are stored for each tree because the information that relates to the Root neighbors is different for each instance.
IamNeigh	
rcvdFromNeigh	

Frame Formats	*Description*
BPDU	The field numNeighbors (2 bytes) is added to the RSTP-SP BPDU.
BPDU-Conf	The fields of the confirmation messages is the same as those described for RSTP-Conf. In the multiple tree framework, the field Root is used as the tree identifier to distinguish between different Root confirmations.

Events	*Description*
BPDU about tree t received in port p	The updates of the confirmation variables are the same as in RSTP-Conf. The confirmation variables updates are applied to the tree instance indicated by the Root field in the priority vector of the received BPDU.
BPDU-conf about tree t received in port p	This process of this new event in RSTP-SP is the same as described for RSTP-Conf with the only difference that it is applied to the concrete tree instance indicated in the Root field.
MessageAgeTimer of tree t expiration in port p	The confirmation mechanism is triggered when a Root neighbor detects a failure on its Root port. The changes are the same as those described in RSTP-SP but considering that a different tree instance is considered.

Procedures	*Description*
BecomeRootBridge	The initialization of the confirmation variables is added as described for RSTP-Conf.
SendBPDU	The procedures to encode the fields in the messages are updated according to the corresponding frame formats.
SendBPDUConf	

extended version of the protocol as RSTP-SP-Conf. Note that the execution of the confirmation mechanism is independent from the main RSTP-SP operation and hence its activation is optional.

Table 8.6 lists the operational changes to introduce the confirmation mechanism in RSTP-SP. First, observe that the bridge variables to manage the confirmation procedure are the same as those in the single tree instance with the only observation that each tree stores its own set (note that these are initialized in the BecomeRootBridge). Second, the message formats are updated in order to introduce the exchange of the necessary confirmation data: the numNeighbors field is included in the common BPDU of table 8.5; and the BPDU-Conf messages have the same structure as in RSTP-Conf. Also note that the corresponding SendBPDU and SendBPDUConf procedures are updated to encode the corresponding new fields. Third, the updates on the message processing are the

same than those described for RSTP-Conf: the reception of a common BPDU introduces the updates of the confirmation variables, and the processing of received BPDU-Conf deals with the processing of the 'rd' and 'ra' messages. Note that each message reception applies to one of the tree instances. And fourth, as in RSTP-Conf the confirmation mechanism is triggered when a neighbor bridge detects a failure on it Root port. The changes are the same as those described in RSTP-SP but considering that a different tree instance is considered depending on the MessageAgeTimer that expires.

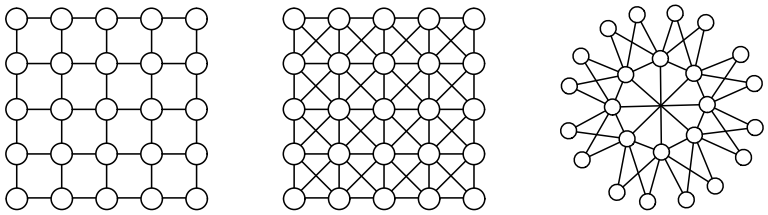
The implementation of the confirmation mechanism certainly avoids the count-to-infinity but it results into an increase of control messages exchanged in the event of link failures. Note that in the scenario with one tree per node, the failure of a single link always represents the failure of a Root link at least in two of the trees (those rooted in the nodes connected to the failed link). This means that any link failure triggers the confirmation mechanism and hence the control messages are flooded within these two trees. In the case of a node failure, the confirmation mechanism is really effective because it is only triggered within the dead tree and it totally avoids the count-to-infinity behavior. Therefore, there is the need to understand which situation is globally less harmful to the network: either (1) using the confirmation mechanism and trigger the process for every link failure, or (2) not using it and survive the count-to-infinity effect within the dead tree with no data traffic being forwarded. The particularities of this tradeoff are studied in the evaluation provided in section 8.5.

8.5 Performance evaluation

This section includes the performance evaluation of the RSTP-SP protocol, as well as a comparison with IEEE SPB, operating in different scenarios: *cold-start*, *link failure* and *node failure*. We have implemented SPB, RSTP-SP and RSTP-SP-Conf in the ns-3 network simulator following the protocol rules described in chapter 5 and in previous sections of this chapter. As in previous analysis, the modeled failure detection mechanism is the immediate physical failure detection; only BPDU messages are simulated and no user traffic is modeled unless otherwise stated; and we take as a reference for the BPDU processing and transmitting delay the study in [30] that assumes a delay of $1.33ms$ per message.

We use the regular network topologies shown in figure 8.8 to capture the behavior of the protocols under different physical deployments: two-dimensional grids of degree 4 (grid4) and degree 8 (grid8); and a more realistic structured topology composed of a meshed core with dual-homed edges (mesh). The table in the same figure includes additional topological characterization aspects of the topologies: the number of nodes (Size), the number of links (Links), the resulting average node degree (Deg), and the network diameter (Diam). Nodes are connected with point-to-point links at 10Gbps and with a propagation delay of 100us and a unitary cost is assumed in all links. In all tests, nodes are configured with random BridgeIDs.

The performance evaluation focuses on the time to construct the trees (*Con-*



(a) grid4 (b) grid8 (c) mesh

Type	*grid4*	*grid8*	*mesh*
Size	64	64	50
Links	112	210	110
Deg	3.5	6.5	4.2
Diam	14	7	4

(d) Topological characterization

Figure 8.8: Topologies used in the performance evaluation of RSTP-SP

vergence Time, CT) and the amount of information exchange required for such action (*Message Overhead*, MO). CT is defined as the time between the failure and the last node reconfiguring the tree. We also measure CT in hop delays as a normalized time unit. That is, a CT of 5 hops means that the protocol takes 5 times the hop delay to converge. MO refers to the amount of messages that the nodes need to exchange in order to recover the tree. The number of messages observed is used to evaluate the overhead in terms of (1) link capacity used and (2) node processing power required.

8.5.1 Convergence time

In the first tests we use the two-dimensional grid of degree 4 and 64 nodes (grid4 in 8.8(a)). We evaluate the performance in the three scenarios: cold-start (network start-up), link failure in the center of the grid, and node failure also in the center. For each scenario we execute 100 simulations with different BridgeIDs. Figure 8.9 shows the average CT, with 95% confidence intervals, for each protocol and scenario. In the cold-start case, all protocols perform equally because they are all based on flooding information that starts at each node and spans the entire network. This results into a CT depending on the topology diameter as it is the longest path the flooding follows. In the event of a central link failure, RSTP-SP and SPB also provide a similar performance because the messages that they issue are propagated from the failure location to the furthest node, and hence it depends on the diameter as well. The small difference is because SPB needs to propagate the entire path, while RSTP-SP only needs to reconfigure the branches that are affected by the failure. The larger CT observed in RSTP-SP-Conf is because of the delay introduced by the confirmation mechanism. The last set of columns

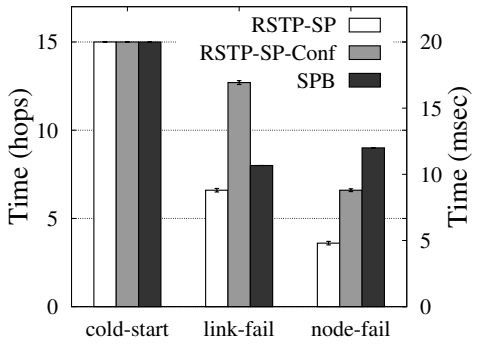


Figure 8.9: Average CT (with 95% conf. inter.) in cold-start, central link failure, and central node failure (100 executions with random BridgeIDs)

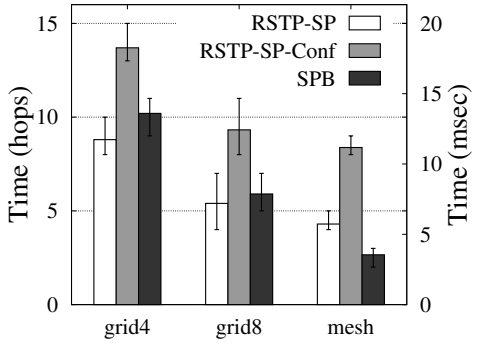


Figure 8.10: Average CT (with 25%-75% percentiles) failing all possible links in different topologies.

in figure 8.9 shows the observed CT after the failure of a central node. First note that the values for RSTP-SP do not consider the dead tree without traffic (actually experiencing count-to-infinity). In this case the CT of both RSTP-SP protocols is reduced because the largest distance between failure and furthest nodes has decreased (an entire node has failed).

We have done an additional analysis to observe the behavior of the protocols in other link failures and in other topologies. We use the grid4 topology, a two-dimensional grid of degree 8 and 64 nodes (*grid8*), and a structured topology of 50 nodes with a meshed core with dual-homed edges (*mesh*). For each topology we fail one link at each execution, and we do as many executions as links to test all possible link locations. This results in 112-210-110 runs for the grid4, grid8 and mesh topologies, respectively. Figure 8.10 shows the average CT with the 25%-75% percentiles (top and bottom part of the vertical lines in each bar). As previously stated, RSTP-SP and SPB perform similarly and RSTP-SP-Conf introduces the confirmation delay. The values for all protocols are proportional

to the topology diameter. We have also done experiments with the previous topologies and varying the size. They confirm that the CT in all protocols and in all networks is related to the network diameter.

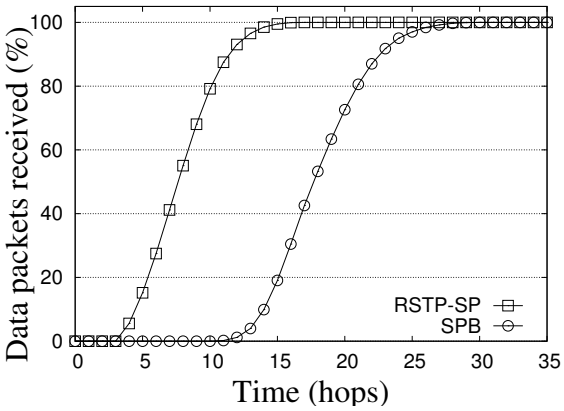
The convergence time can also be analyzed from the data traffic perspective. In the following test we measure the amount of data packets that nodes receive during the tree construction in the different scenarios. This metric is related to the convergence time because the longer the protocol takes to configure the active topology, the longer the period where data traffic is not received as the tree is not active yet (ports are not yet in Forwarding state). Plot in figure 8.11(a) shows the time-line of total received packets in the grid4 topology of 64 nodes. The horizontal axis is measured in hops and the vertical axis in percentage of total received packets. RSTP-SP provides the fastest configuration because the ports are set to Forwarding as the wave-fronts advance (agreement of the handshake). This is why after the CT of 15 hops the 100% of the traffic is already received. Contrarily, in SPB it is not until the reception of the last LSP by the furthest node (after a CT of 15 hops) that this last device does not effectively start sending traffic. These packets are then received at the opposite edge after a propagation of 14 hops (hence a 100% of receptions after 29 hops).

Figure 8.11(b) shows the received data packets during the recovery of a central link failure in the same grid4 topology. First note that both RSTP-SP protocols, with and without confirmation, are practically not affected by the failure because those paths that remain the same in the new topology continue to provide connectivity. Particularly, RSTP-SP-Conf takes more time to provide full connectivity because of the confirmation delay. On the contrary, SPB suffers a higher outage because the new information issued during the reconfiguration creates discordances between topology databases in different nodes. The communication between nodes with different databases is temporary stopped in order to avoid potential forwarding loops.

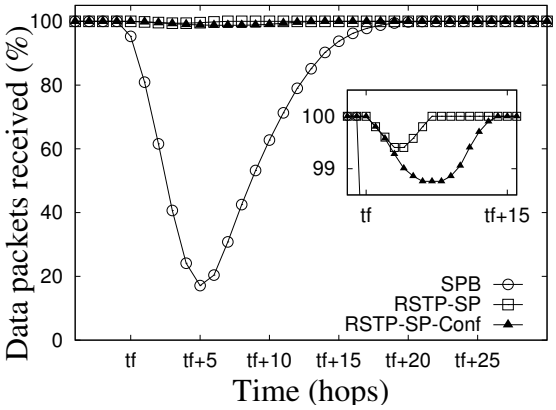
The outage experienced during the recovery from a node failure is shown in figure 8.11(c). The performance of all the protocols is similar to the case of the single link failure, in the sense that SPB experiences an important outage and RSTP-SP protocols almost do not observe loss of connectivity. In the node failure case both RSTP-SP solutions, with and without confirmation, have the same performance because the confirmation mechanism is actually triggered in the tree of the failed Root. Since this failed node does not inject more traffic, the convergence time from the traffic perspective is not affected.

8.5.2 Message overhead

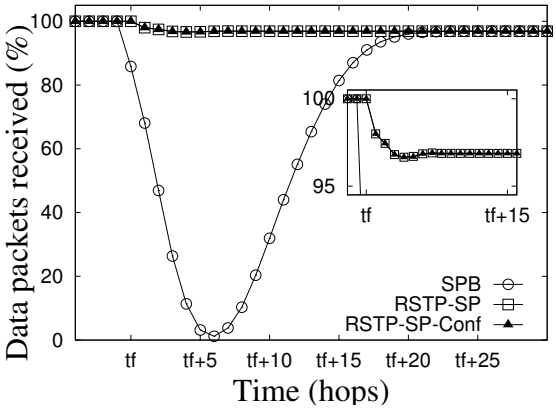
Figure 8.12(a) shows the average MO per node measured in the 64 nodes grid4 for the different scenarios. Observe the logarithmic scale on the vertical axis. MO in a cold-start is similar in all protocols because all are based on flooding of messages. The differences appear when we compare the protocols in the event of failures: SPB clearly outperforms RSTP-SP protocol. The reason is that, for example, in the link failure scenario SPB only floods the link-state updates of the two nodes detecting the failure. Differently, RSTP-SP reconfigures many trees



(a) Cold start

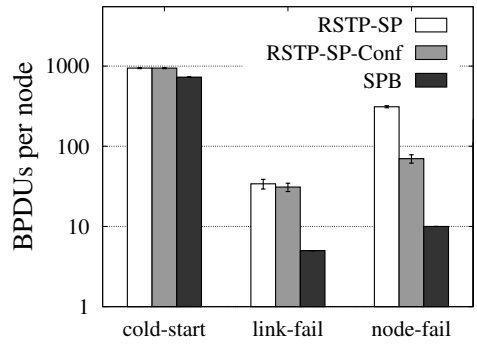


(b) Link failure in the center of the grid

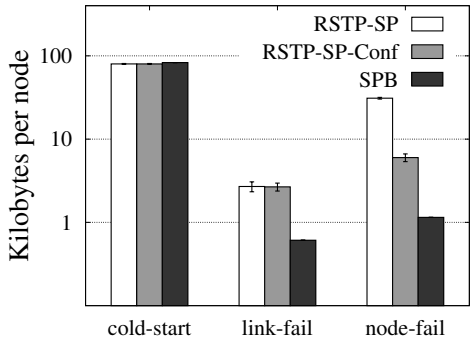


(c) Node failure in the center of the grid

Figure 8.11: Data traffic received during the construction of the tree in different scenarios in the grid4 topology of 64 nodes



(a) Messages



(b) Kilobytes

Figure 8.12: Average message overhead (measured in messages and kilobytes with 95% conf. inter.) in cold-start, central link failure, and central node failure (100 executions with random BridgeIDs).

where each one issues BPDUs to update the paths. Also note that in the link failure scenarios the confirmation mechanism in RSTP-SP does not represent a big difference because it only introduces delay (effect already seen evaluating the CT).

A similar behavior is observed in the event of node failure in the third set of columns in 8.12(a). In this case the reason of the high MO in RSTP-SP is the count-to-infinity occurring in the tree of the failed Root. In addition, even if RSTP-SP-Conf allows removing this effect, the messages transmitted are not reduced to the level of SPB because these are mainly due to the reconfigurations in the trees where a non-Root failure occurs (same order as in the link failure). Figure 8.12(b) shows the message overhead in the same scenarios but measured in kilobytes received per node. The obtained values are very similar to the measurements in number of messages. Looking at the details, in the cold-start scenario we can observe that SPB transmits a higher amount of kilobytes than RSTP-SP

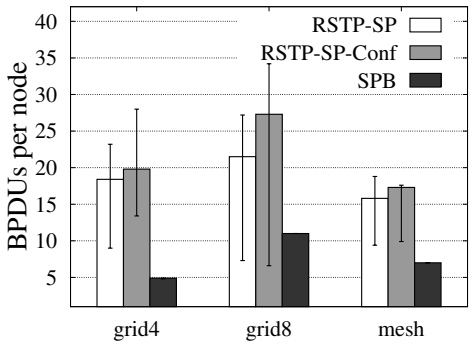


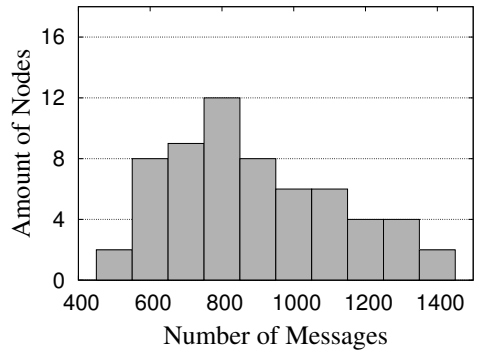
Figure 8.13: Average MO (with 25%-75% percentiles) failing all possible links in different topologies.

protocols. This implies that the average length of the messages received per node during a cold-start is of 114 bytes for SPB and 85 bytes for RSTP-SP. This increase is also observed in the failure situations but SPB still represents a smaller message overhead that RSTP-SP. In conclusion, the difference between observing the message overhead in number of messages or in kilobytes does not represent a big change in performance.

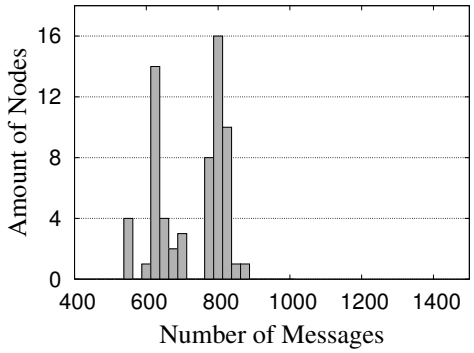
As in the convergence time analysis, we have also measured MO for different topologies and failing all possible links. Figure 8.13 shows the average MO per node for topologies with different average node degree (3.5 for grid4, 6.2 for grid8 and 4.2 for mesh). First, see that these tests confirm that SPB needs fewer messages to recover. Second, also observe that the MO of all protocols is proportional to the average node degree. The difference between RSTP-SP and RSTP-SP-Conf is that the confirmation mechanism is based on flooding and hence it results into more messages in more connected topologies.

The results discussed so far relate to the message overhead observed from the network perspective. A further analysis of the MO performance is presented following by observing the details of a single executions with random BridgeIDs. Figure 8.14 shows the histograms of $\mathrm{MO}_{\mathrm{node}}$ (i.e. amount of nodes that received a concrete number of messages) in a cold-start. The RSTP-SP histogram in figure 8.14(a) indicates that the distribution of messages received is spread. The particularity is that nodes in the corner generally receive less messages, all under 788, than those in the center, above 817. This is because those in the center process more messages belonging to transient configurations than those in the corners. The histogram for SPB is shown in 8.14(b). In this case the distribution is clearly multi-modal with three different blocks centered at 550, 630 and 820 messages. Each one of the clusters refers to the nodes with different number of ports: 2 in the corners, 3 in the edge area, and 4 in the central area. The SPB distribution is based on pure flooding, which results in one message received at each port, and hence the more ports a node has the more messages it receives.

The same individual analysis of $\mathrm{MO}_{\mathrm{node}}$ is done for the scenarios with a central



(a) RSTP-SP (avg: 947, std: 143, min/max: 552/1448)

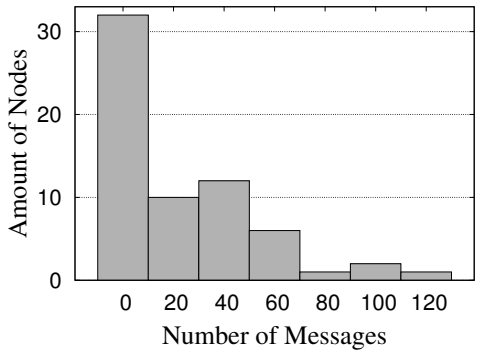


(b) SPB (avg: 727, std: 93, min/max: 545/867)

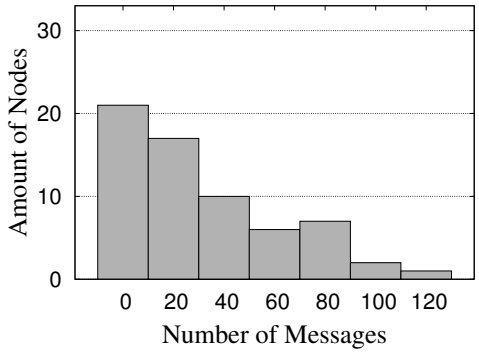
Figure 8.14: Histograms of MO per node during a cold-start

link failure and a central node failure (the histograms are shown in figures 8.15 and 8.16, respectively). For the link failure situations, the distribution of messages for RSTP-SP and RSTP-SP-Conf is quite similar where the majority of nodes receive a small amount of BPDUs (figures 8.15(a) and 8.15(b)). This relates to the locality effect of failure recoveries in distance-vector protocols because the nodes closer to the failure need to reconfigure their paths while those that are located away might not even realize the failure. The only small difference is that RSTP-SP in average transmits a higher amount of messages because of the confirmation mechanisms that are distributed (note that in this link failure case these result into a false alarm and no reboot is triggered). As already pointed out, SPB is more efficient in recovering from a single link failure. The two nodes detecting the link failure disseminate their new adjacencies and the corresponding LSPs are received by all other nodes. Observe in the histogram of figure 8.15(c) how this results in the reception of only from 3 to 6 messages in each node.

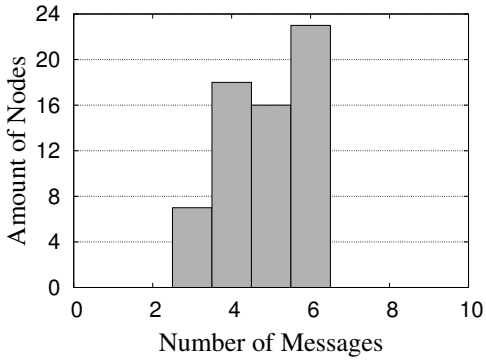
The histograms in figure 8.16 show the MO_{node} measurements in the central node failure case. First observe that in RSTP-SP, in 8.16(a), nodes receive a



(a) RSTP-SP (avg: 24.9, std: 29.4, min/max: 0-113)

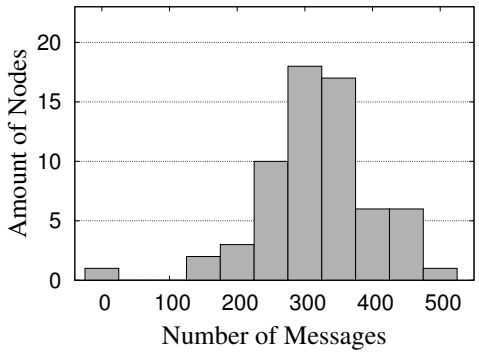


(b) RSTP-SP-Conf (avg: 31.8, std: 29.4, min/max: 0-119)

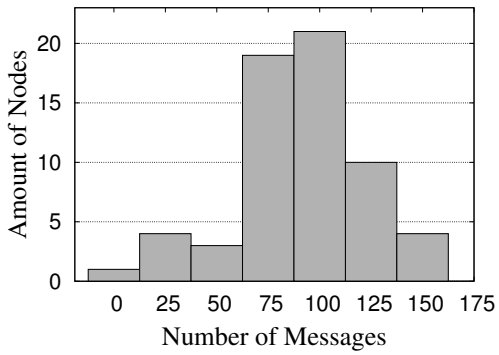


(c) SPB (avg: 4.8, std: 1.1, min/max: 3-6)

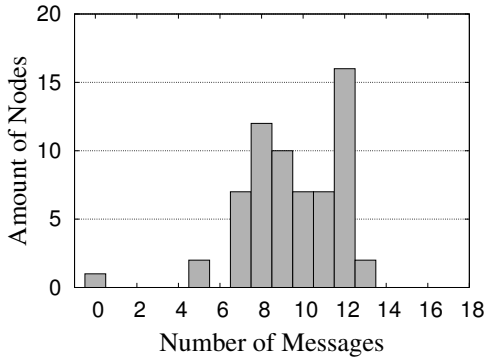
Figure 8.15: Histograms of MO per node during a central link failure



(a) RSTP-SP (avg: 316.5, std: 82.7, min/max: 0-477)



(b) RSTP-SP-Conf (avg: 69.1, std: 33.2, min/max: 0-156)



(c) SPB (avg: 9.5, std: 2.4, min/max: 0-13)

Figure 8.16: Histograms of MO per node during a central node failure

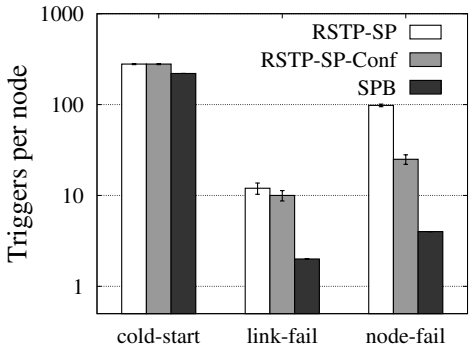


Figure 8.17: Average tree computation triggers in cold-start, central link failure, and central node failure (100 executions with random BridgeIDs)

high amount of messages because of the count-to-infinity that is running in one of the trees (the tree whose Root is the failed node). These messages are quiet distributed among all nodes because once the count-to-infinity is triggered, the looping of BPDUs affects the entire network. When running RSTP-SP-Conf, the analysis of MO_{node} in the histogram of 8.16(b) confirms the absence of count-to-infinity as less messages are transmitted. As in the link failure case, SPB is more efficient in recovering from a node failure from the message overhead perspective. As shown in the histogram of figure 8.16(c), all nodes receive from 5 to 13 messages. This is because all the neighbors of the failed node detect the corresponding link failure and disseminate their new adjacencies.

A different comparison is to evaluate the protocols in the steady state once the cold-start has finished and no failures have occurred. In this situation, all protocols send refreshing messages to keep the topology alive: SPB nodes apply a global refreshing and flood all their link-states; RSTP-SP nodes do it more locally and just send BPDUs from parent to child in all ports of all trees. This results into SPB transmitting the double of messages than RSTP-SP with and without confirmation (216 and 112 respectively).

8.5.3 Tree recomputations

The columns of figure 8.17 indicate the number of messages that, among those received, provide a topology update and hence trigger a tree recomputation. It is important to distinguish between trigger messages and received messages that are discarded because only the first ones require an important computation in the node (electing roles in RSTP-SP or running Dijkstra in SPB). The percentage of triggers over the total messages for SPB and both RSTP-SP protocols remains similar around 30% in all cases. This indicates that the RSTP-SP protocols still require a higher number of triggers than SPB.

Although the comparison of number of triggers can indicate an approximate measure of the node processing complexity required, a further study that analyses

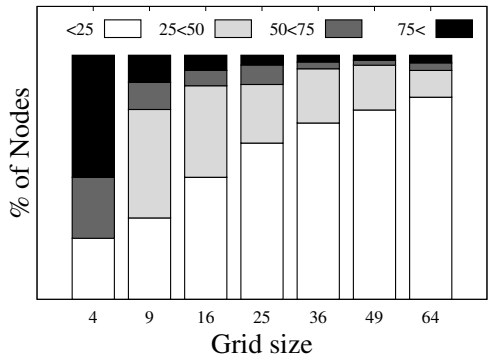


Figure 8.18: Percentage of nodes affected by a link failure recovery in RSTP-SP

complexity aspects of the two protocols is needed and left as future work. There is indeed a big difference between a single tree recomputation in RSTP-SP (comparison of as many vectors as ports) and SPB (execution of as many Dijkstra's as trees, or nodes). Therefore, even if the difference in triggers between the protocols is of one order of magnitude (SPB lower), this could turn over (SPB higher) when analyzing the protocol complexity and measuring number of operations.

In relation to the messages that trigger a tree recomputation, we also analyze the affectation of a central link failure in terms of trees that need to be recomputed because the link failure really affects them. A particularity of the link-state protocols is that any failure always leads to a flooding of a new physical topology, which results into a recalculation of all trees in all nodes. On the other hand, a distance-vector protocol only reconfigures the affected nodes and trees. This can be observed in figure 8.18 showing the percentage of nodes that reconfigure a certain amount of affected trees after a central link failure. For example, black boxes indicate the percentage of nodes that are affected in more than 75% of the trees (e.g. in a network of 4 nodes, half of them are affected in more than 3 trees). Note that when the network size grows, the percentage of affected trees decreases and the majority of the nodes only reconfigure less than 25% of the trees. Note that SPB in this plot would always indicate 100% of affected trees in 100% of the nodes.

§ 9. Conclusion

Network providers are really considering the migration of their transport networks to Ethernet technology. This represents a new application for Ethernet and the new identified requirements involve a revision of some operational aspects. On this background, we were motivated to study the limitations that the spanning tree protocols introduce into the path of Ethernet extension towards provider networking. Clearly identifying these drawbacks has allowed us to design the required RSTP extensions to (1) provide quick recovery times in all failure situations, (2) make use of all network links and hence take advantage of the available redundancy, and (3) operate with shortest-path communication between all pair of nodes.

An additional remark is that a fundamental requirement of the proposed solution is that it had to lie in accordance with the Ethernet framework. This means that keeping essential aspects such the broadcast operation, plug-n-play property or backwards compatibility, is a must in any proposal in Ethernet networking. This requirement not only has driven the design of particular operational extensions but also has determined methodological aspects. This is why we have focused our study in first clearly understanding the nature of the problem. Identifying the detailed particularities of the protocol operation that cause the limitations has allowed for the design of the right extensions that solve such problems but maintain the rest of properties.

The first contribution of the thesis is the analysis of the tree construction included in chapter 6. The objective was to understand the operation and characterize the convergence time of the protocol. Our contribution in this aspect is that we have analyzed the protocol focusing on how it evolves step-by-step rather than only observing performance values. Our contribution in this aspect is that we have analyzed the protocol focusing on how information propagates independently on what exact information (and hence solution) is propagated. In this way, we study the nature of the problem instead of a particular solution. An exhaustive analysis of the RSTP operation at network start-up has allowed identifying that the behavior of the protocol can be easily understood as the evolution of propagating wave-fronts originated at each node. This simple characterization has also allowed deriving a theoretical bound of the protocol convergence time that depends on the length of the longest branch in the tree configured. Evaluation results confirm this characterization and conclude that 50ms bound for the convergence time for RSTP is theoretically possible. The exact time highly depends on the physical topology underneath and more concretely on the network

diameter. In addition, we have also analyzed the message overhead required by the protocol and the results show that, since it is based on flooding of information, it grows with the level of network connectivity (or the average node degree).

The main disadvantage of RSTP is that it experiences the count-to-infinity problem when the Root of the tree fails (see section 3.5). Analyzing with detail this phenomenon we have further identified that the looping BPDUs lead to the creation of deadlocks in a link where the two neighbors believe that the Root is in the opposite direction (as described in section 7.1). The occurrence of such deadlocks requires a timer expiration before it is released, and this introduces delays of tens of seconds when recovering from the Root failure. To the best of our knowledge, while the count-to-infinity problem is known and studied in RSTP, the identification of the deadlocks occurrence and the consequent recovery delay is a novel contribution of our work.

A detailed study of the count-to-infinity behavior has allowed us to discern that the main cause is the utilization of false information in the Alternate ports once the Root has failed. For this reason in chapter 7 we have described RSTP-Conf as an extension to RSTP that makes a safer use of this information. RSTP-Conf is based on a confirmation mechanism that (1) checks the availability of the Root before using an Alternate port, and (2) triggers a global network reboot in case the Root really fails. The new protocol efficiently avoids the count-to-infinity in Root failure scenarios and provides a recovery time bounded by one round-trip delay. Moreover, it recovers as quick as the original RSTP in other failure situations. The only drawback is that it triggers a false alarm confirmation when a single Root link fails, which results in some more exchanged messages and a slightly larger convergence time (but always at the order of hop delays).

In chapter 8 we have presented the description of RSTP-SP as the protocol that extends RSTP in order to operate with optimal paths. The extensions are based on (1) configuring one tree rooted at each node so all paths are optimal, and (2) implementing the path-array in a distributed distance-vector environment in order to ensure symmetrical path selections. We have compared RSTP-SP and SPB, the last evolution of the IEEE standard that also deploys optimal paths but based on a link-state approach, by means of simulation. Results show that RSTP-SP and SPB perform similar in terms of convergence time but SPB experiences a much larger outage during recoveries. In contrast, the RSTP-SP results into higher message overheads compared to SPB. This is because link-state protocols are based on computation while distance-vectors rely on message dissemination. Even though this trade-off between the distance-vector RSTP-SP and the link-state SPB, we have shown that a shortest-path solutions extending the current spanning tree protocols is possible. As a global conclusion, we can state that a global solution that introduces the extensions described for RSTP-Conf and RSTP-SP is able to (1) provide quick recovery times in all failure situations, (2) take advantage of the available redundancy, and (3) operate with shortest-path communication between all pair of nodes.

9.1 Open issues and future guidelines

Although the presented study including the protocols descriptions and their performance evaluation indicates that the addressed Ethernet limitations have been resolved, a number of possibilities for further research remain open.

One of the improvements relates to the study of RSTP-SP and more concretely to extend the performance evaluation with a complexity analysis. The objective would be to evaluate the processing requirements at each node. The evaluation in chapter 8 already provides an initial indication of this metric with the observation of the amount of messages that result into a computation of trees (referred as Triggers). In this case SPB requires less triggers than RSTP-SP in most scenarios. However, a protocol complexity analysis that characterizes the single tree computation, both in RSTP-SP and SPB, can be used to derive the evaluation of the protocol complexity. Note that the number triggers multiplied by, for example, the number of CPU instructions required per trigger provides such evaluation. Actually, the comparison could turn around if we observe the overall processing requirements because one trigger in RSTP-SP results into a few vector comparisons while in SPB it leads to the execution of Dijkstra once per tree. Several consequences of this study about the protocol complexity can be obtained. First, a more powerful device is more expensive. And second, this analysis could be used to evaluate the feasibility of a HW implementation of the protocol: the simpler a single execution is, the easier the implementation. The implementation of a protocol at software level represents an additional delay as each message needs to first reach the client layer before it is processed. A hardware implementation would reduce this bypassing delay until negligible. Additionally, a software implementation signifies an increase of consumed energy against the current necessity of greening the communication networks.

As seen in the evaluation of chapter 8, the main disadvantage of RSTP-SP is that it produces a large message overhead. The main reason is that each tree is actually managed by an independent instance of the single tree protocol. However, the RSTP-SP operation can be optimized in several aspects in order to improve this low performance. One idea to address this could be to share information between different tree instances and reuse some fields in the BPDUs. This might have an important impact especially in the contents of the path-array as many different branches that reach one node are shared between the different trees. Finding a way to re-use information can also have a direct impact on reducing the amount of state information in the node.

Although the path-array in RSTP-SP is introduced to ensure the selection of symmetrical branches across different trees, its functionality can be extended into other aspects. Having the information of the entire path to the Root opens the door to designing more detailed path-control capabilities in order to provide advanced path-selection driven by different metrics. In the current description the path-array only contains the identifiers of the nodes of the path traversed. An immediate extension of the array is to transform it into a matrix structure that contains additional information of each node such as security properties or performance limitations. The use of this information to select the paths that

create the tree branches introduces an important level of flexibility (for example selecting the most secure or the less delayed branch).

Another enhancing that relates to the path-array is a detailed study of the different policies that can be used to decide whether an array is better. As described in chapter 5, in the current description of SPB and RSTP-SP the of array is first sorted and then the elements are compared one to one. The array that has an earlier lower element is considered better. With this policy, the equal-cost paths (or arrays) that contain the lowest node identifiers will always be elected. This might result in a higher number of branches traversing the nodes with these lower identifiers, which might create bottlenecks around these nodes. Therefore, a study about the impact of this sorting policy could provide additional opportunities to improve the solution. This analysis should also lead to the proposal of policies to compare different path-arrays in order to optimize different network aspects. This analysis together with the extension of the array into a matrix that includes additional information can lead to a more complete and flexible framework to provide path control.

9.2 Lessons learned

In addition to the conclusion previously described we have drawn complementary observations along the work of this thesis.

On the Ethernet Philosophy Ethernet has been a successful technology since it first appeared in 1970s. It first won the battle of the LAN in front of other technologies such as Token Ring, Token Bus or FDDI; now it is a very well-positioned candidate to also become the main transport technology in the MAN/WAN areas. Several aspects that comprise to the concept of Ethernet philosophy are the real pillars behind its success.

From the strategic point of view, the high quality standardization effort is the main strength of Ethernet technology. A quick standardization provides easy interoperability between different vendor devices that leads to an increase of number of Ethernet networks. This results into mass production of Ethernet technology that reduces its price, and, by economic laws, increases the sales again. All this also translates into an intense competition that leads to the continual Ethernet innovation. And this feedback effect works because Ethernet keeps growing; if just one step of the previous cycle is broken, Ethernet might start experiencing failure.

From the operational point of view, the success of Ethernet can be explained with concepts like backwards compatibility, plug-n-play or operational simplicity. Operational details like the broadcast condition or the learning operation have been present in Ethernet technology since the beginning and they have been kept in all the steps of the evolution. Any proposal that improves the functionality of Ethernet must consider each one of these aspects or otherwise it fails outside of the Ethernet framework.

Both the strategic and the operational perspectives are two issues to have in mind when working with Ethernet evolutions and really understanding them might mean the success or failure of a technology.

On path-selection paradigms An important observation identified with the study in this thesis is the different aspects to evaluate when selecting between the two main path-selection approaches (distance-vector and link-state). The operational differences between the two approaches have positioned them as successful solutions for different type of technologies.

Link-state protocols such as IS-IS or OSPF handle most of the path-selection solutions in intra-domain routing (routing within the area managed by the same entity). Routers are intelligent but complex devices that need to make path selections based on different criteria. The link-state protocols are hence the right candidate to provide such flexibility thanks to the independent local computation of paths. Differently, distance-vector protocols have traditionally fit network solutions that required less flexibility but where operational simplicity was a plus. This is the case of the spanning tree family in Ethernet networking or BGP in inter-domain routing (routing between areas managed by different entities).

The different philosophies between the two approaches actually determine their application in networks with different flexibility requirements. While both types of protocols are distributed techniques, the differences in their internal operation justify distinction. Link-state protocols operate in a distributed fashion during the distribution of topology information among nodes. However, they operate in a centralized fashion when making the path selections. Note that it is not really centralized because all nodes compute the paths, but do it independently once they all have the entire topological information. This is why properties of centralized techniques, such as flexibility in configuration, can be obtained in a distributed framework. The disadvantage of this operation mode is that the processing requirements are not really distributed because actually each node needs to execute the entire computations to select the network paths. Differently, distance-vector protocols operate in a distributed fashion in both the distribution of topological information and the computation of paths. A completely distributed path selection makes more difficult to obtain the same level of flexibility. Nevertheless, it actually allows to evenly distributing the computation required hence reducing node processing requirements. It is important to note that the comparison in terms of computation requirements is a qualitative observation and would require a detailed complexity analysis to justify such hypothesis.

There is a compromise from the functional perspective that results in deciding for one approach or the other. On one hand, in networks where flexibility in configuration and performance optimization are strong requirements, the link-state approach might be a good solution. On the other hand, in networks where the issues such as robustness and simple management are the key evaluation aspects, a distance-vector approach might be then the best candidate.

On the use of the simulation Analytical models, simulations or test-beds are different evaluation tools that provide different information about a system and that should be used with different objectives. It is very well-known that simulations are a computer-aided technique to provide an initial overview of the performance of a system without the need to implement it in practice. In our case, the evaluation of network protocols can be perfectly carried out with a simulation platform.

A common sequence to use a simulation platform is: (1) load the input parameters; (2) execute the simulation; (3) observe the results. This workflow can be very efficient with the use of automated scripts that execute different runs with different parameters in order to, for example, do an exhaustive sensitivity analysis. While it is trivial to understand that the simulations can be used as an evaluation tool to obtain performance results, it is also important to consider the simulation as a design tool. For example, observing in detail the output traces of the simulator is very useful to understand the behavior of a protocol. Note that these traces are user-configurable so they can contain the information that is required for each particular analysis. Also, compared to the real system, an advantage of the simulation is that it has access to all the information of all the simulated modules. For example this can be used to identify a particular state of the system and stop the simulation at that moment to obtain a complete snapshot of that instant.

One of the main efforts during the work presented in this thesis has actually been the use of the simulation platform as a design and evaluation tool. For example, comprehending aspects of RSTP such as the wave-fronts propagation or the deadlocks during a count-to-infinity have been possible because of detailed analysis of the simulation traces.

[1] Cisco Systems. Cisco visual networking index: Forecast and methodology, 2008-2013. 2009.

[2] H. Schulze and K. Mochalski. Internet study 2008/2009. *IPOQUE Report*, 2009.

[3] A. Kasim. *Delivering Carrier Ethernet*. McGraw-Hill Education, 2007.

[4] R. Sofia. A survey of advanced ethernet forwarding approaches. *Communications Surveys & Tutorials, IEEE*, 11(1):92–115, 2009.

[5] G. Chiruvolu, A. Ge, D. Elie-Dit-Cosaque, M. Ali, and J. Rouyer. Issues and approaches on extending Ethernet beyond LANs. *Communications Magazine, IEEE*, 42(3):80–86, Mar 2004.

[6] M. Whalley and Mohan D. Metro ethernet networks: A technical overview. *MEF White Papers*, 2004.

[7] IEEE. IEEE Std 802.3ah - Amendment Media Access Control Parameters, Physical Layers and Management Parameters for Subscriber Access Networks. 2004.

[8] CableLabs. DOCSIS specifications. *http://www.cablelabs.com/cablemodem/specifications/specifications20.html*, 2012.

[9] C. Xie, N. Ghani, Q. Liu, W. Shu, A. Gumaste, Y. Qiao, and M. Wu. Multipoint ethernet over next-generation sonet/sdh. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[10] G. Kramer and G. Pesavento. Ethernet passive optical network (epon): Building a next-generation optical access network. *Communications magazine, IEEE*, 40(2):66–73, 2002.

[11] F. Davik, M. Yilmaz, S. Gjessing, and N. Uzun. Ieee 802.17 resilient packet ring tutorial. *Communications Magazine, IEEE*, 42(3):112–118, 2004.

[12] IEEE. IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Bridges. *ANSI/IEEE Std 802.1D, 1998 Edition*, pages i–355, 1998.

[13] IEEE. IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1–269, 2004.

[14] A. Myers, E. Ng, and H. Zhang. Rethinking the service model: Scaling ethernet to a million nodes. In *Proc. HotNets*. Citeseer, 2004.

[15] K. Elmeleegy, A.L. Cox, and TS Ng. Understanding and mitigating the effects of count to infinity in ethernet networks. *IEEE/ACM Transactions on Networking (TON)*, 17(1):186–199, 2009.

[16] IEEE. IEEE standard for local and metropolitan area networks virtual bridged local area networks. *IEEE Std 802.1Q-2005 (Incorporates IEEE Std 802.1Q1998, IEEE Std 802.1u-2001, IEEE Std 802.1v-2001, and IEEE Std 802.1s-2002)*, pages 1–285, 2006.

[17] IEEE. IEEE 802.1aq Shortest Path Bridging (Draft 4.0). *IEEE 802.1 documents*, February 2012.

[18] C.E. Spurgeon. *Ethernet: the definitive guide*. O'Reilly & Associates, Inc., 2000.

[19] R.M. Metcalfe and D.R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.

[20] N. Abramson. The aloha system: another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference*, pages 281–285. ACM, 1970.

[21] F. Backes. Transparent bridges for interconnection of ieee 802 lans. *Network, IEEE*, 2(1):5–9, 1988.

[22] R. Perlman. An algorithm for distributed computation of a spanningtree in an extended LAN. *SIGCOMM Comput. Commun. Rev.*, 15(4):44–53, 1985.

[23] M. Seaman. High Availability Spanning Tree. *IEEE 802.1 documents (/docs1996/n013.pdf)*, October 1996.

[24] M. Seaman. Speedy Tree Protocol. *IEEE 802.1 documents (/docs1999/speedy_tree_protocol_10.pdf)*, January 1999.

[25] M. Seaman. Truncating Tree Timing. *IEEE 802.1 documents (/docs1999/truncating_tree_timing_10.pdf)*, January 1999.

[26] M. Seaman. Loop Cutting in the Original and Rapid Spanning Tree Algorithms. *IEEE 802.1 documents (/docs1999/loop_cutting08.pdf)*, June 1999.

[27] E. Sfeir, S. Pasqualini, T. Schwabe, and A. Iselt. Performance evaluation of ethernet resilience mechanisms. *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on*, pages 356–360, 12-14 May 2005.

[28] L.S. Carmichael, N. Ghani, P.K. Rajan, K. O'Donoghue, and R. Hott. Characterization and comparison of modern layer-2 Ethernet survivability protocols. *System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on*, pages 124–129, 20-22 March 2005.

[29] G. Prytz. Network recovery time measurements of RSTP in an ethernet ring topology. *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 1247–1253, 25-28 Sept. 2007.

[30] R. Pallos, J. Farkas, I. Moldován, and C. Lukovszki. Performance of rapid spanning tree protocol in access and metro networks. In *Access Networks & Workshops, 2007. AccessNets' 07. Second International Conference on*, pages 1–8. IEEE, 2007.

[31] S. McQuerry. Ccna self-study ccna preparation library. *Recherche*, 67:02, 2004.

[32] D. Bertsekas and R. Gallager. Data networks, 1992, 1992.

[33] R. Perlman. *Interconnections: bridges, routers, switches, and internetworking protocols*. Addison-Wesley Professional, 2000.

[34] J.L. Gross and J. Yellen. *Handbook of graph theory*. CRC, 2004.

[35] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[36] H. Gredler and W. Goralski. *The complete IS-IS routing protocol*. Springer-Verlag New York Inc, 2005.

[37] G.S. Malkin. *RIP: an intra-domain routing protocol*. Addison-Wesley Longman Publishing Co., Inc., 2000.

[38] A. Meddeb. Smart spanning tree bridging for metro ethernets. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 492–499. IEEE, 2008.

[39] J. Qiu, G. Mohan, K.C. Chua, and Y. Liu. Local restoration with multiple spanning trees in metro ethernet. In *Optical Network Design and Modeling, 2008. ONDM 2008. International Conference on*, pages 1–6. IEEE, 2008.

[40] M. Huynh, P. Mohapatra, and S. Goose. Cross-over spanning trees enhancing metro ethernet resilience and load balancing. In *Broadband Communications, Networks and Systems, 2007. BROADNETS 2007. Fourth International Conference on*, pages 251–260. IEEE, 2007.

[41] P.M.V. Nair, S.V.S. Nair, M.F. Marchetti, G. Chiruvolu, and M. Ali. Distributed Restoration Method for Metro Ethernet. *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, pages 94–94, 23-29 April 2006.

[42] J. Farkas, C. Antal, L. Westberg, A. Paradisi, T.R. Tronco, and V. Garcia de Oliveira. Fast Failure Handling in Ethernet Networks. *Communications, 2006. ICC '06. IEEE International Conference on*, 2:841–846, June 2006.

[43] A. Meddeb. Ngl01-3: Multiple spanning tree generation and mapping algorithms for carrier class ethernets. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–5. IEEE, 2006.

[44] T. Cinkler, A. Kern, and I. Moldován. Optimized qos protection of ethernet trees. In *Design of Reliable Communication Networks, 2005.(DRCN 2005). Proceedings. 5th International Workshop on*, pages 8–pp. IEEE, 2005.

[45] J. Farkas, C. Antal, G. Tóth, and L. Westberg. Distributed resilient architecture for ethernet networks. In *Design of Reliable Communication Networks, 2005.(DRCN 2005). Proceedings. 5th International Workshop on*, pages 8–pp. IEEE, 2005.

[46] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh. Viking: a multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 4:2283–2294 vol.4, 7-11 March 2004.

[47] K. Lui, W. Lee, and K. Nahrstedt. STAR: a transparent spanning tree bridge protocol with alternate routing. *SIGCOMM Comput. Commun. Rev.*, 32(3):33–46, 2002.

[48] R. Garcia, J. Duato, and J.J. Serrano. A new transparent bridge protocol for LAN internetworking using topologies with active loops. *Parallel Processing, 1998. Proceedings. 1998 International Conference on*, pages 295–303, 10-14 Aug 1998.

[49] Y. Lin and M. Gerla. Brouter: the transparent bridge with shortest path in interconnected LANs. *Local Computer Networks, 1991. Proceedings., 16th Conference on*, pages 175–183, 14-17 Oct 1991.

[50] R. Garcia, J. Duato, and F. Silla. LSOM: A Link State protocol Over MAC addresses for metropolitan backbones using Optical Ethernet switches. *Network Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on*, pages 315–321, 16-18 April 2003.

[51] R. Perlman. Rbridges: transparent routing. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 2:1211–1218 vol.2, 7-11 March 2004.

[52] T.L. Rodeheffer, C.A. Thekkath, and D.C. Anderson. SmartBridge: A scalable bridge architecture. In *SIGCOMM*, pages 205–216, 2000.

[53] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[54] C. Kim, M. Caesar, and J. Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 3–14. ACM, 2008.

[55] G. Ibáñez, B. De Schuymer, J. Naous, D. Rivera, E. Rojas, and J.A. Carral. Implementation of arp-path low latency bridges in linux and openflow/netfpga. In *High Performance Switching and Routing (HPSR), 2011 IEEE 12th International Conference on*, pages 30–35. IEEE, 2011.

[56] M.D. Schroeder, A.D. Birrell, M. Burrows, H. Murray, R.M. Needham, T.L. Rodeheffer, E.H. Satterthwaite, and C.P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *Selected Areas in Communications, IEEE Journal on*, 9(8):1318–1335, Oct 1991.

[57] T.L. Rodeheffer and M. Schroeder. Automatic reconfiguration in Autonet. *SIGOPS Oper. Syst. Rev.*, 25(5):183–197, 1991.

[58] R. Casado, A. Bermudez, F.J. Quiles, J.L. Sanchez, and J. Duato. Performance evaluation of dynamic reconfiguration in high-speed local area networks. *High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on*, pages 85–96, 2000.

[59] G. Ibáñez, A. García-Martínez, J.A. Carral, P.A. González, A. Azcorra, and J.M. Arco. Hurp/hurba: Zero-configuration hierarchical up/down routing and bridging architecture for ethernet backbones and campus networks. *Computer Networks*, 54(1):41–56, 2010.

[60] F. De Pellegrini, D. Starobinski, M.G. Karpovsky, and L.B. Levitin. Scalable cycle-breaking algorithms for gigabit Ethernet backbones. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 4:2175–2184 vol.4, 7-11 March 2004.

[61] D. Starobinski, M. Karpovsky, and L.A. Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Transactions on Networking (TON)*, 11(3):411–421, 2003.

[62] OPNET. OPNET network simulator. *http://www.opnet.com/*, June 2012.

[63] ns3. ns3 network simulator. *http://www.nsnam.org/*, June 2012.

[64] E. Bonada, D. Cavic, and D. Sala. Implementation of a Layer-2 Bridge in ns3 (poster). *Proceedings of SIMUTools 2008*, March 2008.

[65] E. Bonada and D. Sala. On the Theoretical Bounds of the Spanning Tree Algorithm. *Jornadas Telecom I+D*, October 2008.

[66] J. Chiabaut. All pairs shortest paths performance measurements. *IEEE 802.1 documents (/docs2008/aq-chiabaut-all-pairs-shortest-path-0308-v01.pdf)*, March 2008.

[67] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.