

Spanning Tree Protocol, from a feature CCNA's Perspective.

written by **Gerald C. Paciello**

Jan. 29, 2015

A little bit of history.

Before we talk about **Spanning Tree Protocol**, let's organize the different variants of STP. The original STP was developed by **Radia Perlman** while working for **DCE** back in 1990. At this time there were no switches yet, only bridges, but because bridges and switches technically do the same job -only switches do it more efficiently and have more features- they suffer from the same issues. Also, because STP was developed during a time when there were only bridges, the terminology used in STP, even today, makes reference to bridges a lot (**Root Bridge, Bridge-ID**, etc.). So, when you read about STP from different sources, remember that the terms Switch and Bridge might be used interchangeably.

The original STP, developed by Mrs. Perlman, was later standardized by the **IEEE** as the **802.1D** standard. This two variants, Perlman's STP and 802.1D, implemented only one STP instance for the whole network (This was sometimes referred to as **Common Spanning Tree-CST**), because Vlans had not been invented yet, so one instance of STP was enough.

When Vlans were introduced, Cisco made some improvement on the 802.1D standard to accommodate Vlans and called it **Per-Vlan ST Plus (PVST+)** or sometimes **PVSTP**. This improvements allowed for multiple STP instances (one for each Vlan on the network). This means that each Vlan could have its own STP topology. With this Cisco improvement, a few other extension were also introduce such as **PortFast**.

Later, the IEEE made its own improvement on its standard 802.1D, which was considered to have a very **slow convergence time**, it was called **Rapid STP (RSTP)** and its standard is **802.1W** and, as the first part of the name indicates, it was much faster (rapid) to converge than its predecessor.

Then **Cisco** took the **802.1W** standard and made some proprietary changes as well and called it **Rapid Per-Vlan ST+ or RPVST+**.

Note: Cisco switches support **PVST+, RPVST+ and MST** and because by default cisco switches use PVST+, we are going to talk about that STP variant on this paper.

What STP does and why.

STP was developed to prevent loops (aka, **Switching loops** or **layer 2 loops**) at layer 2 in a switched network with parallel links between switches, thus creating more than one path to a single destination. These loops, in turn, will create a much worst condition called a **Broadcast Storm** which is a result of **broadcast, multi-cast** and/or **unknown-destination frames** looping endlessly around the network, and this happens because of two main reasons; First, because switches will forward these frames out of every port except the receiving port (This is called **Flooding**) every single time they receive them. Second, at layer 2 (as opposed to layer 3) there is no **Time To Live (TTL)** field in the frames header, so frames will not "expire". They will go around and around in a network until the network is shutdown or it fails.

So, when broadcast storms happen, frames will start crowding the network's bandwidth leaving very little room for "good" frames, so the network will start slowing down. Also, PCs will need to process an enormous amount of broadcasts, so their performance will be negatively impacted as well.

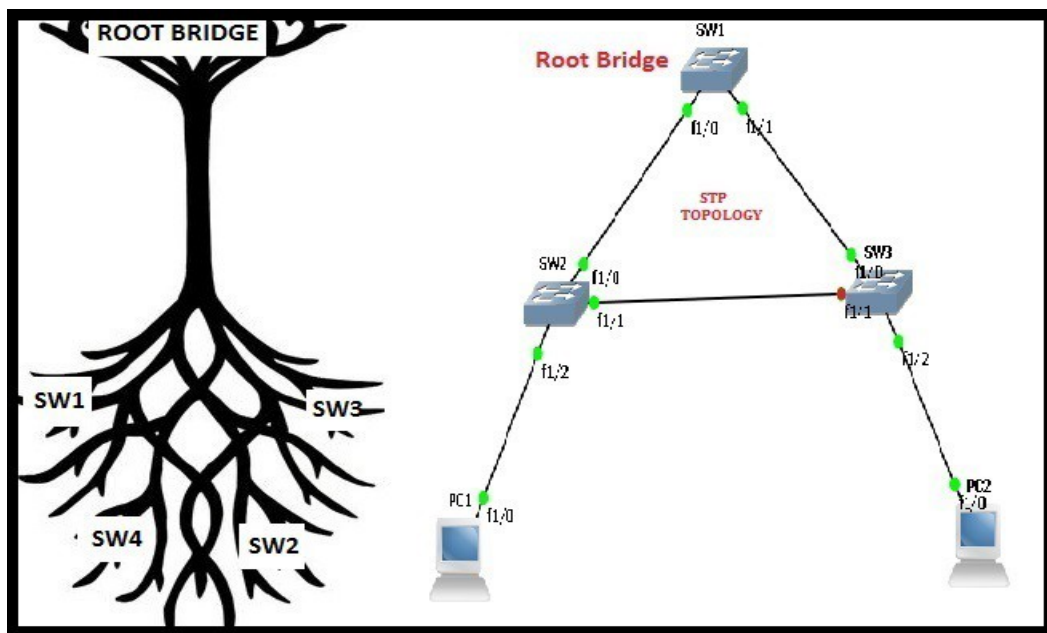
Even though these loops happen when there are multiple paths between switches, having multiple paths to a single destination, called **Redundancy**, is not such a bad thing to have in a network. In fact, redundancy is convenient, very often essential for a network, because it provides alternative paths in case one path fails. In fact, **STP** was developed so that we can have redundancy in our networks.

STP also prevents another switching issue called **MAC table corruption**, which happens when a switch learns about the same destination **MAC** address from duplicate frames received on more than one of its ports.

Protocol Operation.

Before we talk about the specifics of **STP** operations, I want you think of an **STP topology** as an INVERTED tree (look at the image bellow) with its roots being the **Root Bridge** (for now, let's say it is the main switch) on top, and the branches reaching downstream and spanning towards the **non-root bridge** switches. In turn, these other switches will reach upstream towards the **Root Bridge (RB)**.

I can hear you guys saying "Root what, non-root what?!... We'll get to the good stuff soon enough, for now just know that on **STP**, there's a "boss" switch that is called the **Root Bridge** and the other switches will find the **best path** to get to it.



STP CONVERGENCE

STP prevents issues created by parallel links on switched networks and allow us to have redundant networks, by selecting the **best path** to a **destination** and **blocking** the remaining paths until the best path originally selected, for some reason becomes unavailable.

To do this, **STP** uses what is called the **Spanning Tree Algorithm (STA)**, this algorithm will come up with a loop free **STP topology** by selecting first, a switch that will act as a "*central point of reference*" called a **Root Bridge (RB)** and then it will calculate **best paths to this switch** from all other switches in the **STP domain**. This is accomplished by selectively putting some ports in a **forwarding state** and other ports on a **blocking state**, following specific rules from a process call the **Election Process**, we'll see this process in detail soon.

The process just described (very briefly) is referred to as **Convergence** and we say that something has converged once it has finished its initiation process and it is ready to do what it's supposed to do.





Port States.

In order for **STP** to converge and do what it is supposed to do (which is to create a *loop free* topology), among other things, it has to systematically control the ports and elect which port will forward data and which one will block data, as we stated before. Ports start at the **Blocking** state and then they start *transitioning* through the different states until they get to the **Forwarding** state.

Before we learn about how **STP** does this, let's learn about the port states and their roles:

1. **Blocking:** The port does not learn MAC addresses nor participate in the switching of data frames, it will discard these frames when received. But it does receive and process **STP frames** (called **Bridge Protocol Data Units** or **BPDUs**).
2. **Listening:** In this state the port does not learn MAC addresses nor participates in the switching of data frames but it does process **BPDUs**. Also, *it removes old unused MAC table entries*.
3. **Learning:** During this state, the port still does not forward data frames, but it does forward **BPDUs** and *re-populates* its MAC table. In other words, it **learns** MAC address from data frames received.
4. **Forwarding:** This is the "normal" port state on a switch. When a port is forwarding, it is fully participating in the switching of every frame accordingly.
5. **Disabled:** When a port is disabled, it does **not** participate on any frame switching, it is **shutdown**, either administratively (Manually) or because of failure.

Here's a table that summarizes port states and shows you the LED color on the switch port during on each state:

State	Type	Forwards STP Frames (BPDUs)	Forwards User Data Frames	Learns MAC Addresses	LED Color on Switch.
Blocking	Stable	YES	NO	NO	
Listening	Transitional	YES	NO	NO	
Learning	Transitional	YES	NO	YES	
Forwarding	Stable	YES	YES	YES	
Disabled	Stable	NO	NO	NO	OFF

Timers.

These timers play a big role as they control how often or for how long STP performs its chores. The values, which are **configurable**, are carried inside **BPDU**s sent *only* by the **Root Bridge**, and have a direct impact on **convergence time**, as we'll see just ahead. For now, these are the default values:

Timer	Default Value	Description
Hello	2 Seconds	How often will a BPDU be sent.
Max Age	20 Seconds (10 x <i>Hello</i> Time)	How long will a port remain in Blocking state after a topology change.
Forward Delay	15 Seconds	How long will a port remain in Listening/Learning states, before transitioning to Forwarding state. (15secs each by default, 30secs total)

Convergence Times.

Convergence time is defined by the total time it takes for a port to transition from either, **Listening** to **Forwarding** or **Blocking** to **Forwarding**. We can think about this as **Convergence Time**= *Listening to Forwarding transitions* and **Re-Convergence Time**= *Blocking to Forwarding transitions*.

Convergence Time: When the switch first comes online (turns on):

1. **Listening state** – *Transitional* state. (15secs by default or Forward Delay)
2. **Learning state** – *Transitional* state (15secs by default or Forward Delay)
3. **Forwarding state** - *Stable* state.

As you can see takes a total of **30 seconds** (15 on Listening + 15 on Learning). This is because the ports were not Blocking to start with so it saves 20 seconds (default value). In other words, it did not have to go through the **Max Age** timer.

Re-Convergence: If a process needs to re-converge, it means that it had already converged at least once before, and the most common reason for STP to have to re-converge is a topology change (i.e. a link failure). Even though a failure is a failure, when it comes to **STP**, this same failure will have *different effects on STP Convergence time*, depending on where, in the STP topology, the failure occurred. However, it's **not** a matter of **physical location**, it is a matter of **perspective**. **STP** refers to this as **Direct Failure** or **Indirect Failure**.

Let's elaborate; say we have **SW1 (RB)**, **SW2** and **SW3** connected to each other, and the link between **SW1** and **SW2** fails. From **SW1** and **SW2**'s perspective, this will be a **Direct Failure** causing a *30 seconds re-convergence*, because as soon as the link goes down, **SW2**'s port will move to the Listening state (15secs Forward delay) and start sending BPDUs advertising itself as RB. Then, it'll move to the Learning state (again,

15secs Forward delay) and start learning MAC address and then, after 30 seconds, it transitions to the forwarding state.

Meanwhile, for **SW3** this will be an **Indirect Failure** (it did not happen on any of its links) and this will cause a *50 seconds re-convergence* because as soon as **SW3** receives the BPDU from **SW2** advertising itself as **RB**, **SW3** will start its Max Age timer (20 secs). After Max Age, if it doesn't receive any BPDUs from **SW1**, it will erase **SW1**'s as **RB** and enter **SW2** in its place. However, in this case, this will not happen because **SW3** will still receive a better BPDUs from **SW1** so it'll send BPDUs out to **SW2** saying, "hey I have a better **RB** than you and I can get to in with a cost of 19".

Direct Failure: Failure occurred on one of the switch's links:

1. **No Blocking State, directly to Listening.**
2. **Listening** - *Transitional* state. (15secs by default or Forward Delay)
3. **Learning** - *Transitional* state. (15secs by default or Forward Delay)
4. **Forward** - *Stable* state.

Indirect Failure: Failure **did not** occurred on one of the switch's links:

1. **Blocking** - *Stable* state. (20secs by default or 10 x Hello Timer)
2. **Listening** - *Transitional* state. (15secs by default or Forward Delay)
3. **Learning** - *Transitional* state. (15secs by default or Forward Delay)
4. **Forward** - *Stable* state.

Port Roles.

We are going to get into the details of the **Election Process** just ahead, for now let's just say that during this process, 1st the **RB** is elected and then the ports that are going to participate on the **STP topology**, are elected as one of three **STP Port Roles**, these are the **Root Port (RP)**, **Designated Port (DP)** or **non-Designated Port (non-DP)**.

Here they are:

1. **Root Bridge.**
The **RB** is the center of the universe as far as STP is concerned. The **RB** sends out a BPDU called the **Configuration BPDU** or **CBPDUs** and also controls the various STP timers.
2. **Root Port.**
Root Ports (RP) are the ports on **non-RB** switches (**1** per switch) that have the best path to the **RB** itself. Think of the **RPs** as the upstream ports reaching up towards the **RB**. These ports will be in the **forwarding** state.
3. **Designated Ports.**
A **Designated Port (DP)** is the port on each network segment that connects **its segment** to

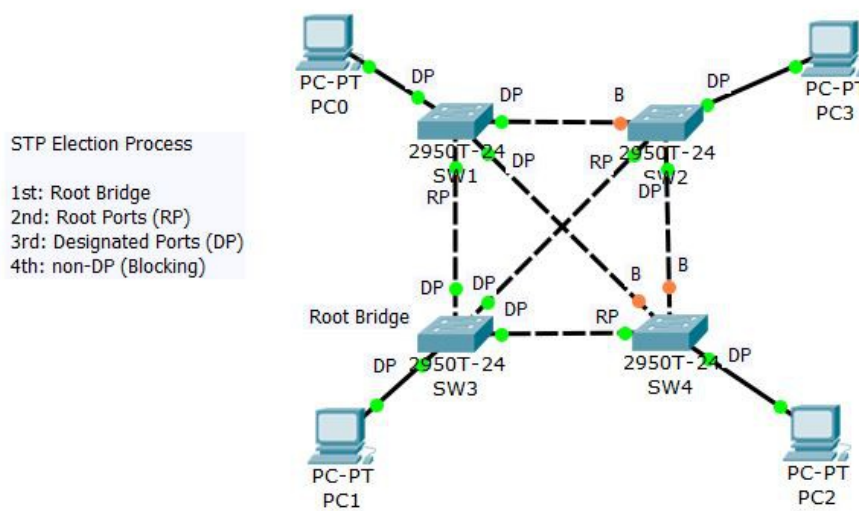
best **segment to the RB**. Think of **DPs** as downstream ports forwarding **CBPDUs** from the **RB**. These ports will be in the **forwarding** state.

4. **Non-Designated Ports.**

Any remaining port that was not elected **RP** nor **DP** will be a **non-DP**. These ports will be **Blocking**.

They will only receive and process STP BPDUs but will discard any other type of frames.

In the picture below, the **Election Process** has ended and all ports are functioning as either, **Root Ports (RP)**, **Designated Ports (DP)** or **non-Designated Ports (non-DP)**. Notice that **RPs** and **DPs** are **Forwarding** data and **non-DPs** are **Blocking**, as stated before.



Check your understanding, compare the picture above with the information in the following table:

Characterization of Port	STP State	Description
All the root switch's ports	Forwarding	The root switch is always the designated switch on all connected segments.
Each nonroot switch's root port	Forwarding	The port through which the switch has the least cost to reach the root switch (lowest root cost).
Each LAN's designated port	Forwarding	The switch forwarding the hello on to the segment, with the lowest root cost, is the designated switch for that segment.
All other working ports	Blocking	The port is not used for forwarding user frames, nor are any frames received on these interfaces considered for forwarding.

Port Cost.

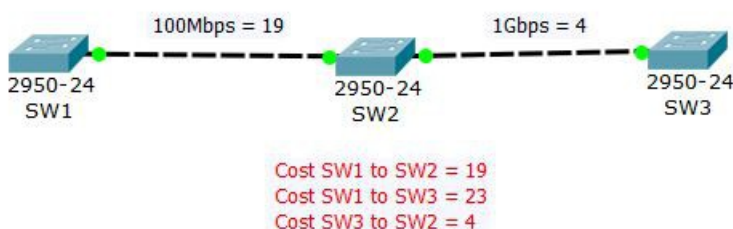
Every link in an **STP** topology has a cost, and this cost is associated with each link's speed. After the **Root Bridge** has been elected, which is the **1st** step in the **Election Process**, the process needs to determine which

are going to be the **RP**s and **DP**s respectively. This is done by adding the link's **cost** (aka **Port Cost**) values along **every individual link** from **non-RBs** to the **RB**, and the path with the **lowest path-cost** wins the **election**.

The STP **Ports cost** is a **configurable** value, but its default values are shown below.

Port Speed	STP Port Cost
10 Mbps	100
100 Mbps	19
1 Gbps	4
10 Gbps	2

Port costs is probably the most easy to understand concept in **STP** because it is pretty straight forward, but here is a picture that makes it easier:



Election Process.

Ok, so after mentioning the infamous “**Election Process**” several times now, let’s see what it is about. I remember when I started reading about the election process, I was really confused about the values the **STA** uses to make each election. Well, it wasn’t the values exactly, those are pretty straight forward, it was the tiebreakers I couldn’t find a place for and I think it was because of the way it was explained in the literature I was reading that time. So before we get into the details about the process itself, let me show you what those values are and **when** they are used.

Lowest Bridge ID (BID)
Lowest Path-Cost to RB
Lowest Sender’s BID
Lowest Sender’s Port ID

The values listed above are the only 4 values that **STA** considers for **RB** election **and** for **RPs / DP**s election: Now, the 1st entry, **Lowest Bridge ID**, is the only value the **STA** uses to elect the **Root Bridge** and it is called **Bridge-ID** or just **BID**. We will see how and what the **Bridge-ID** is in just a second, for now remember that the **Bridge-ID** is the value used to choose the **RB**.

The 2nd entry is the **Lowest Path-Cost to RB** and it is used when the **STA** elects **RPs** and also **DP**s. The 3rd

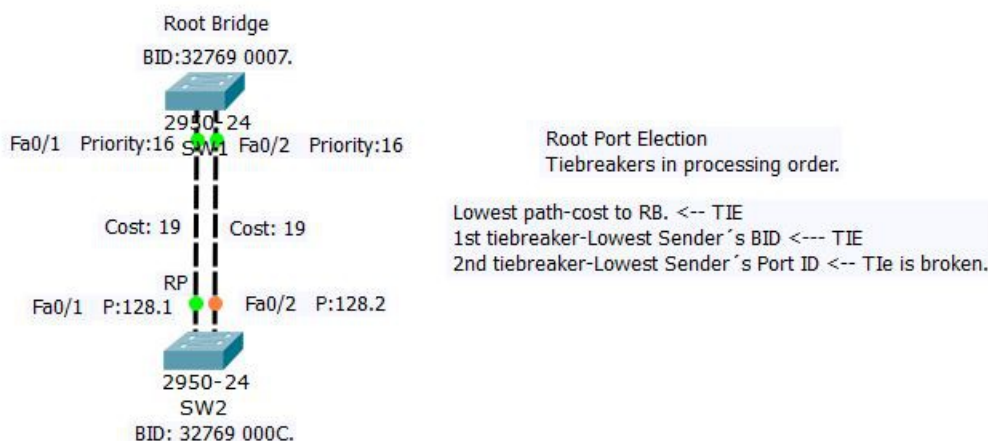
and 4th entries are **ONLY** used for **RP/DP** if and **ONLY** if there is a tie, specifically, a path-cost tie for RP and DP election.

So, summarizing:

Lowest Bridge ID >>>> Root Bridge election only.
Lowest Path-Cost >>>> RPs and DPs election only.
Lowest Sender's BID >>>> Only if path-cost tie.
Lowest Sender's Port ID >>> Only if sender's BID tie.

Let's elaborate; if, and only if, there is a tie for **RP** or **DP** election (the **path-costs are equal**), the process will examine the next value looking for a tie breaker, and this value is **Lowest Sender's BID**, which is the **Bridge ID** of the switch that sent the **BPDU**. If the tie persists, the process moves on to the second and final tiebreaker; the **Lowest Sender's Port ID**, which is the lowest port number on the switch that sent the **BPDU**.

Let's examine the topology below:



As you can see, the **STP** topology above has already converged and we can see that the **RB** elected is **SW1** and it's easy to see why, it's because its BID, **327690007.xxxx.xxxx.xxxx** is **lower** than **SW2's BID**. Remember the first value on our list a minute ago? It was **Lowest Bridge-ID** and **SW1** has the lowest **BID**. I'll explain why a little bit ahead..

We can also see that **SW2** has already elected its **RP** which is **Fa0/1**, but let's see why:

The **STA** will look for **Lowest Path-Cost to RB**, but this was not possible because both paths to the RB have a path-cost of **19**.

The **1st tiebreaker** calls for **Lowest Sender's BID**, again this is not possible because there is only one neighbor (the **RB**) and the **BPDU**s received by **SW2** from **SW1** will have the same **BID**, so tied again.

So we come to the **2nd tiebreaker**, **Lowest Sender's Port ID**, the lowest port on the **RB** is F0/1, so this is the

port that will be elected **RP** or **DP**, depending on what was being elected. The process for **RP** or **DP** election uses the same value, lowest path-cost, the difference is that for **RP** the **STA** looks for lowest cost to **RB** from the **non-RB** switch to the **RB** and for **DP** it's from the **network segment** to the **RB**... it is just about perspective.

Bridge Protocol Data Unit - BPDU.

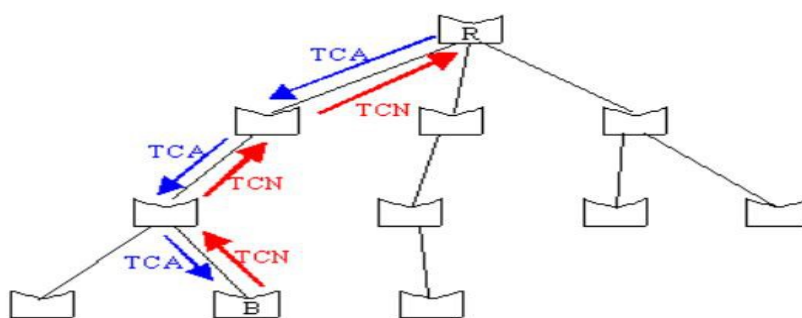
We've also mentioned through out this document that switches, in an **STP domain**, use a special frame to **exchange STP data between them**, so let's talk about the **BPDU**. There are 2 types of **Bridge Protocol Data Unit - BPDUs**; **Configuration BPDUs (CBPDU)** and **Topology Change Notification BPDUs (TCN)**.

The **CBPDUs**, which is sent out **ONLY** by the **RB**, carries the necessary information downstream for switches to make their decisions during the **Election Process** for **Root Bridge (RB)**, **Root Port (RP)**, **Designated Port (DP)** and **Non-Designated Port (non-DP)** among other things. It also holds the timers mentioned earlier as the **RB sets all the different timers as well**.

Every **non-RB** switch will receive **every 2 seconds** by default (or *Hello Timer*), a **CBPDU** from the **RB** on its **Root Port (RP)**, notice that this is downstream. The **non-RB** switch will only send a **BPDU**, the **TCN BPDU** out of its **RP** (upstream) when it has to inform about a topology change (i.e. a link went down). When the other switches receive the **TCN**, they will forward the **TCN** upstream and reply with a **TCAck** downstream to the switch that sent the **TCN**, and the process will repeat until the **RB** is reached.

Note: Think of RPs and DPs like this: RPs send BPDUs UPSTREAM towards the Root Bridge (hence the name, Root Port) and DPs send BPDUs DOWNSTREAM towards non-RB switches.

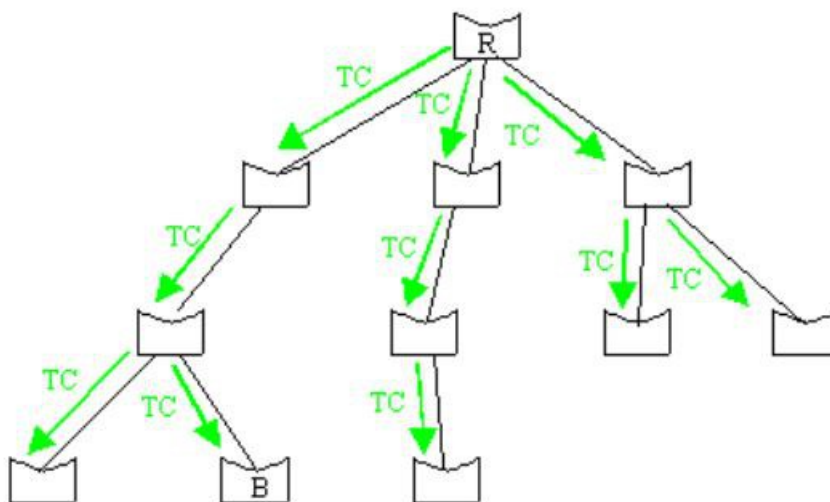
The picture below (from <http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/12013-17.html#topic2>) demonstrates this.



Bridge B notifies a topology change by sending a TCN on its root port. The TCN is acknowledged and forwarded up to the root bridge R.

When the **RB** finally receives the **TCN**, it will reply with **CBPDU** that has its **Configuration Change (TC)** bit **set to 1** and NON-RBs will forward this BPDU downstream so every switch knows about the topology change event (switches receive and process this BPDU, even on blocking ports) and they can start the re-convergence process to accommodate the topology change.

The picture below (from <http://www.cisco.com/c/en/us/support/docs/lan-switching/spanning-tree-protocol/12013-17.html#topic2>) demonstrates this.



The root R sets the TC bits in its bpdus. This bpdus is relayed to the whole network.

BRIDGE ID - BID.

As I said before, the **CBPDU** is the **BPDUs** used during convergence, it contains, among other fields, the **Bridge-ID or BID** which is used for **RB** election.

FIELD	DESCRIPTION
Root BID	BID of the switch that the sender of this BPDU believes to be the RB.
Sender's BID	BID of the switch sending the Hello BPDU.
Cost to RB	The cost between this switch & the RB.
Timer values on RB	Hello, Max Age and Forward Delay Timers.

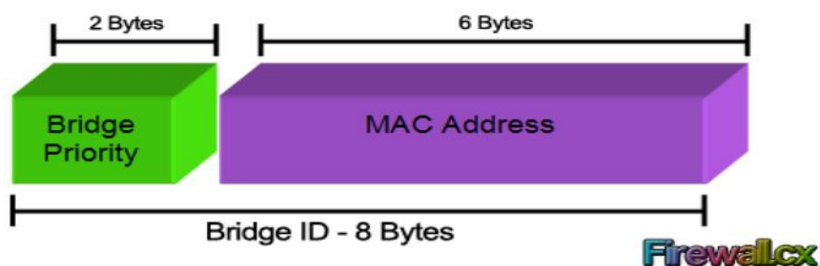
Let's see how **STP** come up with the Bridge Id. It consists of the switch **Priority** value, this value is **configurable** (in multiples of 4069, we'll see why later) and by default is **32768**, plus the **System ID Extension**, which is the **Vlan number** and finally the **MAC address**.

Priority	System ID Extension (Vlan#)	MAC
32768	100	00E0.xxxx.x
BID= 3286800E0.xxxx.xxxx.xxxx		

Notice that the priority went up from **32768** to **32868**, as a result of adding the default **priority + sys-id-ext**. Again, the **sys-id-ext**. Is the Vlan value, in this case **100**.

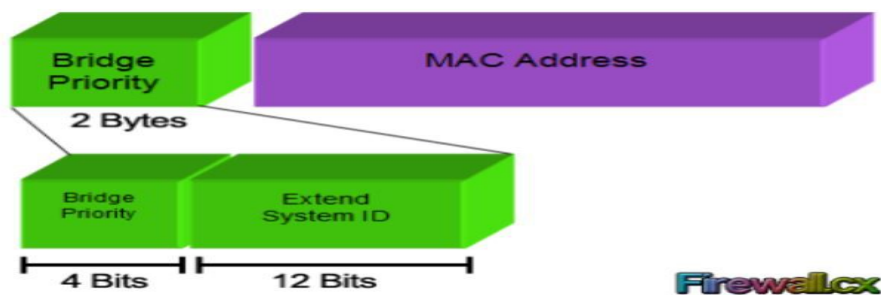
With **STP 802.1D**, the **BID** consisted of 8 bytes divided in two, as follows: the first **2 bytes (16bits)** corresponds to the **Bridge Priority** and the remaining **6 bytes** corresponds to the **MAC** address.

As the following picture from <http://www.firewall.cx/networking-topics/protocols/spanning-tree-protocol/1054-spanning-tree-protocol-root-bridge-election.html> demonstrates.



As I stated before, **STP 802.1D** ran a **single instance** of STP for the entire network, but as networks started getting bigger, more complex and then Vlan's were introduced, it was necessary to run multiple instances of STP, and so it was necessary to include **Vlan** information in the **BID**. This was accomplished by using **12 out of the 16bits** from the **Priority** field in the 802.1D **BID** to include the **Vlan** number. This subfield was called **System ID Extension** and notice that **2 to the 12th** (12 is the number of bits borrowed) is **4096**, exactly the maximum number of Vlan's allowed.

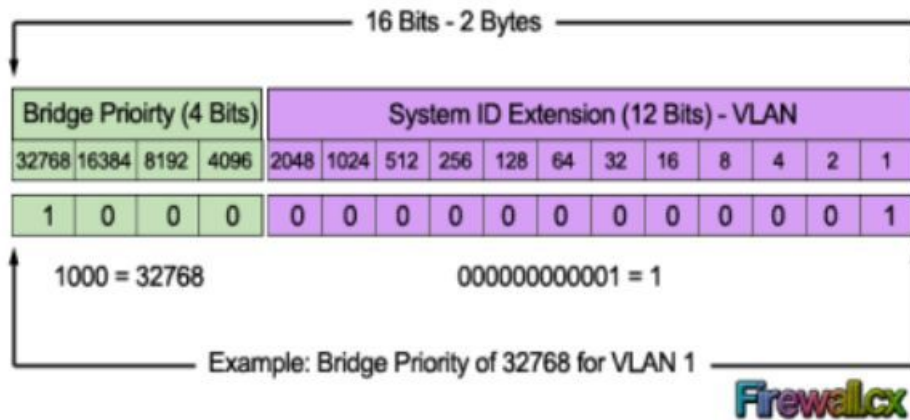
As the following picture from <http://www.firewall.cx/networking-topics/protocols/spanning-tree-protocol/1054-spanning-tree-protocol-root-bridge-election.html> indicates.



So, from the **2 bytes Priority** field, the first **12 least significant bits** (the first 12 bits on the right) were **borrowed to accommodate the Vlan number**. This means, obviously, that the **Priority field has been reduce to only 4bits**. This is why, where before (on **802.1D**) it was possible to have any **Priority** value from **0 - 65,536 (2 to 16th)**, now (on any Per Vlan variant) we can only get from **0 - 61440** but in **multiples of 4096 only**, this gives us a total of **16** possibilities. But this is not a limitation because there are no scenarios (not that I can think of, at least) where more than 16 different Priority values would be needed.

If you look at the picture below, specifically the **place values** on the **Bridge Priority** field, you'll see that the least significant bit has a place value of 4096. This means that the **Bridge Priority** field cannot use the place values 1, 2, 4, 8, 16, 32, ... and so on. It only has **4096, 8192, 16384** and **32768** to work with and therefor we can only have bridge priority values from 0 to 61440, expressed in multiples of 4096.

Picture from <http://www.firewall.cx/networking-topics/protocols/spanning-tree-protocol/1054-spanning-tree-protocol-root-bridge-election.html>



ROOT BRIDGE.

OK, so now that you've read about what is needed for the **Election process**, let's talk about it in detail. The **Root Bridge** is the main switch in the topology and every non-RB will have a port called a **Root Port** pointing upstream through the **lowest path-cost** to the **RB**.

RBs have a couple of very specific tasks:

- It is the **only switch that sends CBPDUs**.
- Imposes all the **STP timers**

Root Bridge Election Process.

When switches come online, they do not know yet if they are the only switch on the network or if there are hundreds of switches, and so they will start **flooding** the network with their own **CBPDUs** advertising their **Bridge-ID** and **themselves as Root Bridge**. In other words, in the **CBPDUs** that they are flooding onto the network, the **Root BID**, which is the **BID** of the switch that switches believe is the **RB** and the **Bridge ID**, which is the **BID** of the switch sending the **CBPDU**, will have identical values.

As soon as they **receive a CBPDU from another switch**, they realize that they are not alone, and at that moment the **election process will begin**.

Election process:

1. The **first** part of the **BID**, the **priority field**, will be compared.
2. If one of the priority value in the **CBPDUs** is the **lowest**, the corresponding switch will be **elected RB** and the **RB election process will stop**.
3. **If two or more priority values are tied for RB** (their **Priority** values are **identical**), the **MAC** address will be **compared** (the **MAC** address is considered only in case a tie).
4. The switch that has the **BID** with the **lowest MAC** among the switches that are tied for **Priority**, will be **elected RB**.

ROOT PORTS.

A **Root Port** is a port on a **non-root bridge** switch (only 1 per non-RB switch and there is **NO** RP on the Root

Bridge, only DPs remember) that has the **lowest path-cost** to the **Root Bridge**. Remember, on the inverted tree analogy, the RP sends BPDUs upstream towards the RB.

Root Port Election Process.

The **RB** floods its CBPDU advertising a **Cost to RB** value of **0**, because it is the **RB** so there is no cost to get to the **RB**, of course. The other switches will receive this CBPDU and add the cost of the port it received the CBPDU on, to the advertised **Cost to RB** value on the CBPDU received, which is **0**.

For example, if the CBPDU comes in **Fa0/1**, the switch will add **19** to **0** (if you look at the **Port Costs table** you can see that the cost for a **100Mbps** link is **19**) so, **19 + 0=19**. So this switch will forward a copy of the CBPDU received advertising a **Cost to RB** value of **19**. The next switch will receive this BPDU and again add the cost value of the port it received it in (let's say that it is a **1Gbps** link) to the **Cost to RB** value advertised in the BPDU received, which is **19**, so **4 + 19=23**. When this switch send its own BPDU, it will advertise a **Cost to RB** value of **23**. This same process will repeat until the port with the lowest -cumulative- path-cost on each non-RB switch is elected **Root Port**.

Let's follow this same example step by step:

1. **RB** floods a **CBPDU** advertising a **Cost to RB of 0 (zero)**. This makes sense because *there is a cost of 0 for the RB to get to itself*.
2. **Non-RBs** receive this **CBPDU** and they add the cost of its port to the advertised **Cost to RB** in the **CBPDU**. So **Cost to RB(0) + cost of 100Mbps(19) = 19**.
3. The **non-RB** sends its own copy of the **CBPDU** advertising a **Cost to RB of 19**
4. The next **non-RB** downstream receives the **BPDU** advertising a **Cost to RB** of 19. The port in which this **BPDU** came into is a **1Gbps** port, which has a cost of **4**. The switch adds **19 + 4 = 23** and sends out a **BPDU** with an advertised **Cost to RB** of **23**.
5. The process repeats until all the RPs have been elected.

DESIGNAED PORTS.

A **Designated Port** is the port *forwarding BPDUs* to non-RB switches *downstream*. These BPDUs will be coming into the non-RBs **RPs OR non-DPs** (Remember, non-DP are *blocking* ports but they receive and process BPDUs still). There is **1 DP per network segment**, for example, if there are **7** network segments in total, there will be **7 DP** total. You can also think of a **DP** as the port that connects the segment it belongs to, to the lowest patch-cost segment to the RB.

Also, the switch that contains the **DP** for a particular segment, is referred to as the **designated switch** for that segment.

Designated Port Election Process.

The process to elect **Designated Ports** starts where the **RP** process ends and, just like the **RP** process, looks for the lowest path-cost to the **RB**. During the DP election process we need to look at it in a "per segment basis".

Let's look at the process from the beginning and you'll see what I mean:

1. **Root Bridge:** The process looks at the **whole topology** to find a **Root Bridge: Lowest Priority value** or **lowest BID** if there is a Priority tie, **wins**.
2. **Root Port:** Now that everyone knows who is the **RB**, the STP process looks at **each non-RB switch** for the **1** port that has the lowest cost to get to the **RB**.
3. **Designated Port:** In this step, the process looks at **each network segments** and elects the **1** port that has the lowest path cost to the **RB**.
4. The process no longer has to calculate anything. Every port remaining that hasn't been elected **RP** nor **DP**, become **non-DP**.

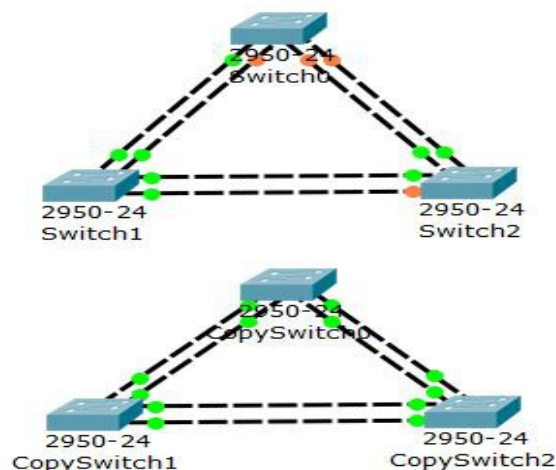
You might get a bit confused on **RP** and **DP** election process because both look for the *lowest path-cost* to the bridge. It seems that the process is looking for two different things considering the same value... and that's correct!. The exact same value is considered, what makes the difference is the point of reference; For RP, is the lowest path-cost **from the switch** and for DP, is the lowest path-cost **from the network segment**.

Additional STP Features.

Being that this technology has been around for a long time, different updates were introduced over the years. Among these updates, there are a few features that we need to talk about, and those features are **Etherchannel**, **PortFast** and **BPDU Guard**.

Etherchannel.

Please observe the picture bellow for a moment. As you can see, they are the same topology but something is different about the ports states, isn't it? **STP** is running on both these topologies yet some ports on the top topology are blocked and some are not. But this is OK, **STP** does this in order to **prevent loops**, right? So, if STP is running on both topologies, how come the one on the bottom has all of its ports in the **Forwarding state**?



Here is the answer; the topology at the bottom has **EtherChannel** configured. Each pair of links from switch to

switch (in our example) are logically bundle together and are **treated as one single link** called **Port-Channel** and since the links are treated as a single interface, all the ports are forwarding.

With **EtherChannel** (aka **Link Aggregation**) we can bundle up to eight, same speed, parallel links together, and this is beneficial in many ways; it provides a higher bandwidth between the switches, it can load-balance traffic across the links involved and it provides redundancy still because even though the **Port-Channel** is treated as a single interface, the links inside can still work independently.

Because of the fact that the links can still work independently, **EtherChannel** eliminates the necessity for **STP** to **re-converge** in case of a failure. This is because, if one of the links inside the **Port-Channel** goes down, **STP** will not **re-converge** and it will continue working as long as **one of the links is still up**.

EtherChannel's Load-balancing.

Let's talk a little bit about load-balancing; when we hear that traffic is going to load-balanced, we imagine bits being distributed evenly through all the links involved, right? Well, that might not be the case here, this is because EtherChannel's load-balancing has a few methods to distribute the load, that we can pick from and configure on our switch using the **port-channel load-balance (method value)** command.

Without going too deep into the details (not a CCNA topic), here's a table from https://ls-a.org/doku.php?id=school:ccnp_1a_chapter6_cert showing these methods:

Table 6-3 *Types of EtherChannel Load-Balancing Methods*

method Value	Hash Input	Hash Operation	Switch Model
src-ip	Source IP address	bits	All models
dst-ip	Destination IP address	bits	All models
src-dst-ip	Source and destination IP address	XOR	All models
src-mac	Source MAC address	bits	All models
dst-mac	Destination MAC address	bits	All models
src-dst-mac	Source and destination MAC	XOR	All models
src-port	Source port number	bits	6500, 4500
dst-port	Destination port number	bits	6500, 4500
src-dst-port	Source and destination port	XOR	6500, 4500

Link Aggregation Protocols.

There are two different protocols for **Link Aggregation** supported by Cisco switches; these are **PAgP (Port Aggregation Protocol, Cisco proprietary)** and **LACP (Link Aggregation Control Protocol)** and this is the industry standard, **IEEE 802.3AD**.

PAgP and LACP Port Negotiation.

When we configure **EtherChannel** on the switches, all ports on each side of the links need to have the same values on; **speed, duplex** and **Vlan assignments**. If that is the case, the ports will be able to negotiate a **Port-**

Channel between them if, the negotiation options (referred to as **channel modes**), are correct at each end. Depending on the **channel mode** the ports are in, they will either send out negotiation requests, wait for a negotiation request, both or neither. If a port on one side is **waiting** for a request, and the port on the other side is **sending out** requests, a **Port-Channel** will be formed. Since there are two **Port Aggregation** protocols supported by Cisco, **PAgP** and **LACP**, there are also two similar sets of channel modes for each of them.

PAgP:

The channel modes for **PAgP** are **ON**, **AUTO** and **DESIRABLE**.

ON: does not send requests **nor** does it **listens** for them.

AUTO: does not send requests but it **listens** for them.

DESIREBLE: sends and **listens** for requests.

PAgP Channel Mode	ON	AUTO	DESIRABLE
ON	Yes	No	No
AUTO	No	No	Yes
DESIRABLE	No	Yes	Yes

LACP:

The only difference between **PAgP** and **LACP** are the modes names, but they work the same way as **PAgP**.

The channel modes for **LACP** are **ON**, **PASSIVE** and **ACTIVE**.

ON: does not send requests **nor** does it **listens** for them.

PASSIVE: does not send requests but it **listens** for them.

ACTIVE: sends and **listens** for requests.

LACP Channel Mode	ON	PASSIVE	ACTIVE
ON	Yes	No	No
PASSIVE	No	No	Yes
ACTIVE	No	Yes	Yes

PortFast and BPDU Guard.

As you already know, when a port is participating in **STP**, before it is ready to forward data, it needs to go through the *Listening* and *Learning* states and this will take 30 seconds by default. What if there is an end device, such as a **PC**, connected to this port? Well... some of the more modern **PCs** are able to boot up in **less than 30 seconds**. If this is the case, the **PC** might **not be able to get an IP address** through DHCP, or it'll simply be sitting there waiting to be able to communicate, wasting time.

There is a way we can go around this, there is a feature introduced by Cisco called **PortFast** and the way it

works is very simple; by configuring **PortFast** on a port, we are telling it that **no BPDUs** will be coming through it, so it should become **active** and **ready to forward frames right away**. In other words, the port will **NOT** go through the 30 seconds delay from the **STP's Forward Delay timer**.

Now, imagine that **PortFast** is configured on a port that runs to an office cubicle. What if someone comes along and plugs in a switch into this port? If that happens, **BPDUs** will start going through this port (because **PortFast** does not block any frames), so something is wrong, correct?... well, we can do something about that also.

By also configuring the feature **BPDU Guard** on the same port that we've configured **PortFast**, we are telling the port that **IF** it receives a **BPDU**, intermediately put the port in a **error disabled** state. A port that is in an error disabled state, will not process any frames until the commands *shutdown / no shutdown* are issued on the port.

References.

Users of The Cisco Learning Network - <https://learningnetwork.cisco.com/welcome>

Cisco CCNA R&S 200-120 Official Cert Guide by Wendell Odom

Cisco CCNA R&S ICND2 200-101 Official Cert Guide by Wendell Odom

Cisco CCNP SWITCH Simplified Vol.1 by Paul Browning