CAKE

CAKE (Common Applications Kept Enhanced)

Description:

CAKE (Common Applications Kept Enhanced) is a modern, advanced queue management algorithm designed to cope with varying network conditions and different types of traffic. It was developed to address the limitations of older algorithms and provide an all-in-one solution to several common network problems.

CAKE's approach is to maintain fairness, minimize bufferbloat, and manage different types of traffic with minimal configuration and tuning. This is achieved through various techniques including flow isolation, bandwidth shaping, and prioritization of small packets.

Property	Description
Bandwith Limit:	This allows you to manually set the bandwidth limit for the CAKE shaper. For instance, if you're aware that your ISP provides you with a 100Mbps connection, you could set the rate to 100Mbps to prevent CAKE from trying to use more bandwidth than available.
Autorate Ingress	This is useful in situations where the connection quality varies. For example, if you're on a cellular data connection that can fluctuate wildly based on signal strength, this option allows CAKE to adjust the shaper bandwidth dynamically to match the estimated capacity of the link.
Overhead	The overhead parameter in the CAKE queue discipline allows you to specify the per-packet overhead (in bytes) to account for when shaping traffic. This is particularly important when dealing with technologies such as DSL, where protocol encapsulation adds additional bytes to each packet. Accurately accounting for these overheads will lead to more precise control of the actual bandwidth being used.
MPU	The Minimum Packet Unit (MPU) parameter allows you to round up the size of each packet to a specified minimum. This is useful for link layer technologies that have a minimum packet size. For example, if your link layer technology has a minimum packet size of 64 bytes, you can configure CAKE with `mpu 64`.
ATM	This is for Asynchronous Transfer Mode, a type of network technology often used in DSL broadband connections. ATM uses fixed 53-byte cells, each of which can carry 48 bytes of payload. The `atm` keyword compensates for this overhead.
Overhead Scheme	The overhead compensation feature in CAKE allows it to account for the extra bytes added by various link layer technologies, which can be significant in some cases. This is important because CAKE operates on the packet sizes reported by the Linux kernel, which do not include these extra bytes. If the overhead is not accounted for, CAKE's shaper might allow more data onto the link than it can actually handle, leading to unexpected packet loss.
RTT	Manually specify an RTT. This is for advanced users who know the exact RTT they want to use.
RTT Scheme	The CAKE queue discipline allows you to specify the Round Trip Time (RTT) it should consider when managing traffic. This is important because the time it takes for a packet to travel from a source to a destination and back impacts how CAKE manages network congestion. If the actual RTT of your network is close to the value you specify, both the throughput and latency of your network should be well managed.
Diffserv	Diffserv (Differentiated Services) is a mechanism used to prioritize and classify network traffic based on the Diffserv field in the IP packet header. CAKE (Common Applications Kept Enhanced) provides Diffserv support, allowing traffic to be divided into "tins" (traffic classes) and applying different treatment to each tin.
Flow Mode	When flow isolation is enabled, CAKE puts packets from different flows into different queues. Each queue maintains its own Active Queue Management (AQM) state, and packets are delivered from each queue fairly using a DRR++ (Deficit Round Robin++) algorithm. This algorithm works to minimize latency for "sparse" flows, or flows that contain fewer packets.
ACK Filter	ACK or acknowledgment filters are a feature of the Cake algorithm that allows for the handling of TCP ACK (acknowledgment) packets. TCP ACK packets are essential to the function of the TCP protocol; they acknowledge the receipt of TCP segments, providing a form of flow control. However, these packets can consume a significant portion of your upload bandwidth, especially when downloading large files. By implementing ACK filtering, Cake can compress these ACKs, reducing their impact on upload bandwidth.
NAT	These options control how CAKE handles traffic when Network Address Translation (NAT) is being used. `nat` makes CAKE consider the post-NAT addresses and ports when isolating flows, which can be useful when you're shaping traffic on a router that is also performing NAT for its connected devices.
Wash	The "wash" option in the CAKE queue discipline is used to address the issue of frequently mis-marked traffic entering or exiting a DiffServ (Differentiated Services) domain. When traffic passes through networks or providers, there is a chance that the DSCP (Differentiated Services Code Point) markings could be modified or incorrectly assigned.

Overhead Schemes

The overhead compensation feature in CAKE allows it to account for the extra bytes added by various link layer technologies, which can be significant in some cases. This is important because CAKE operates on the packet sizes reported by the Linux kernel, which do not include these extra bytes. If the overhead is not accounted for, CAKE's shaper might allow more data onto the link than it can actually handle, leading to unexpected packet loss.

1. **Manual Overhead Specification**: You can manually specify the overhead in bytes. For instance, if you know your link layer adds 18 bytes of overhead to each packet, you can configure CAKE with `overhead 18`. Negative values are also accepted, within a range of -64 to 256 bytes.
2. **MPU**: The Minimum Packet Unit (MPU) parameter allows you to round up the size of each packet to a specified minimum. This is useful for link layer technologies that have a minimum packet size. For example, if your link layer technology has a minimum packet size of 64 bytes, you can configure CAKE with `mpu 64`.
3. **ATM**: This is for Asynchronous Transfer Mode, a type of network technology often used in DSL broadband connections. ATM uses fixed 53-byte cells, each of which can carry 48 bytes of payload. The `atm` keyword compensates for this overhead.
4. **PTM**: This is for Packet Transfer Mode, another network technology often used in VDSL2 connections. PTM uses a 64b/65b encoding scheme, which effectively reduces the usable bandwidth by a small amount. The `ptm` keyword compensates for this overhead.
5. **Failsafe Overhead Keywords**: The `raw` and `conservative` keywords are provided for quick-and-dirty setup. `raw` turns off all overhead compensation in CAKE, and `conservative` compensates for more overhead than is likely to occur on any widely-deployed link technology.
6. **ADSL Overhead Keywords**: Most ADSL modems use ATM cell framing and have additional overhead due to the PPPoA or PPPoE protocol used. Keywords such as `pppoa-vcmux`, `pppoe-llc`, etc. are provided to account for these overheads.
7. **VDSL2 Overhead Keywords**: VDSL2 uses PTM instead of ATM and may also use PPPoE. Keywords such as `pppoe-ptm` and `bridged-ptm` are provided to account for these overheads.
8. **DOCSIS Cable Overhead Keyword**: DOCSIS is the standard for providing Internet service over cable-TV infrastructure. The `docsis` keyword is provided to account for the overhead of DOCSIS.
9. **Ethernet Overhead Keywords**: These keywords account for the overhead of Ethernet frames, including the preamble, inter-frame gap, and Frame Check Sequence. `ethernet` and `ether-vlan` are provided for Ethernet and Ethernet with VLAN respectively[1].

RTT Schemes

The CAKE queue discipline allows you to specify the Round Trip Time (RTT) it should consider when managing traffic. This is important because the time it takes for a packet to travel from a source to a destination and back impacts how CAKE manages network congestion. If the actual RTT of your network is close to the value you specify, both the throughput and latency of your network should be well managed.

Here are the RTT settings you can use, what they mean, and when you might use them:

1. `rtt TIME`: Manually specify an RTT. This is for advanced users who know the exact RTT they want to use.
2. `datacentre`: This is for extremely high-performance networks, such as a 10 Gigabit Ethernet (10GigE) network in a data center. The RTT is assumed to be 100 microseconds. Example: Use this if you're managing network traffic in a high-capacity data center.
3. `lan`: This is for pure Ethernet networks, such as those you might find in a home or office environment. The RTT is assumed to be 1 millisecond. Example: Use this if you're managing traffic on a wired home or office network, but not when shaping for an Internet access link.
4. `metro`: This is for traffic mostly within a single city. The RTT is assumed to be 10 milliseconds. Example: Use this if you're managing traffic for a network in a single city, like a city-wide corporate network.
5. `regional`: This is for traffic mostly within a European-sized country. The RTT is assumed to be 30 milliseconds. Example: Use this if you're managing traffic for a network that spans a country.
6. `internet`: This is suitable for most Internet traffic. The RTT is assumed to be 100 milliseconds. Example: Use this if you're managing general Internet traffic on a typical broadband connection.
7. `oceanic`: This is for Internet traffic with generally above-average latency, such as that suffered by Australasian residents. The RTT is assumed to be 300 milliseconds. Example: Use this if you're managing traffic in a location with high latency, like Australia or New Zealand.
8. `satellite`: This is for traffic via geostationary satellites. The RTT is assumed to be 1000 milliseconds (1 second). Example: Use this if you're managing traffic for a network that uses a satellite internet connection.
9. `interplanetary`: This disables Active Queue Management (AQM) actions, because the RTT is so long (3600 seconds). It's named "interplanetary" because the distance from Earth to Jupiter is about one light-hour. Example: This is not typically used in standard networking situations, but might be useful in extremely high latency situations, such as experimental long-distance communication scenarios.

Remember, these are guidelines, and the best setting depends on the specific characteristics and requirements of your network. If you are unsure, the `internet` setting is a good starting point for most scenarios[1].

FLOW ISOLATION PARAMETER

When flow isolation is enabled, CAKE puts packets from different flows into different queues. Each queue maintains its own Active Queue Management (AQM) state, and packets are delivered from each queue fairly using a DRR++ (Deficit Round Robin++) algorithm. This algorithm works to minimize latency for "sparse" flows, or flows that contain fewer packets.

The key aspect here is the method by which CAKE determines different flows, known as "flow isolation." CAKE uses a set-associative hashing algorithm to reduce flow collisions.

1. **flowblind** - This parameter disables flow isolation, and all traffic goes through a single queue for each 'tin' or traffic class. *Useful in scenarios where specific flow isolation is not needed or desired, such as when you want to process all traffic the same way regardless of source or destination.*
2. **srchost -** Here, flows are determined solely by the source address. This could be beneficial on the outgoing path of an Internet Service Provider (ISP) backhaul. *A telecom company might use this to ensure fair use of its backbone network by different regions or customers.*
3. **dsthost -** With this parameter, flows are characterized only by their destination address. This might be beneficial on the incoming path of an ISP backhaul. *An enterprise could use this to balance incoming traffic to its different servers.*
4. **hosts -** In this case, flows are defined by source-destination host pairs. This is host isolation rather than flow isolation. *This might be useful in a data center network, where you want to ensure that communication between specific pairs of servers is fair.*
5. **flows** - Flows are characterized by the entire 5-tuple: source address, destination address, transport protocol, source port, and destination port. This is the kind of flow isolation performed by SFQ and fq_codel. *This could be used by a cloud provider to ensure fairness among different virtual machines communicating over various protocols and ports.*
6. **dual-srchost** - Here, flows are defined by the 5-tuple, and fairness is applied first over source addresses, then over individual flows. This is a good choice for outgoing traffic from a Local Area Network (LAN) to the Internet. *A university might use this to prevent any single user or device from hogging the internet connection, no matter how many different connections they're using.*
7. **dual-dsthost** - In this case, flows are defined by the 5-tuple, and fairness is applied first over destination addresses, then over individual flows. This is suitable for incoming traffic to a LAN from the internet. *A large company could use this to prevent any single server or system from overwhelming the network's incoming bandwidth.*
8. **triple-isolate** - This is the default setting where flows are defined by the 5-tuple. Fairness is applied over both source and destination addresses intelligently, as well as over individual flows. This prevents any one host on either side of the link from monopolizing it with a large number of flows. *A Internet Service Provider (ISP) might use this to ensure fair service to all its customers, regardless of how many connections they have and whether they*

Ack Filter

ACK or acknowledgement filters are a feature of the Cake algorithm that allows for the handling of TCP ACK (acknowledgment) packets. TCP ACK packets are essential to the function of the TCP protocol; they acknowledge the receipt of TCP segments, providing a form of flow control. However, these packets can consume a significant portion of your upload bandwidth, especially when downloading large files. By implementing ACK filtering, Cake can compress these ACKs, reducing their impact on upload bandwidth.

1. **ack-filter** - This enables the ACK filter feature. With this option, Cake will attempt to identify and filter out ACK packets that do not convey significant new information or do not need to be sent immediately, helping to improve the utilization of the upload bandwidth.
2. **ack-filter-aggressive** - This is a more aggressive version of the ack-filter option. It will result in more ACK packets being compressed or filtered out, which can lead to further improvements in upload bandwidth utilization but may potentially impact TCP performance.The use of the ack-filter or ack-filter-aggressive options depends on your specific network conditions and requirements. For instance, if you find that ACK packets are using a large portion of your available upload bandwidth, then enabling the ACK filter might help. On the other hand, if you're experiencing issues with TCP performance and suspect that ACK filtering might be contributing, you could try disabling it or using the less aggressive option.

Wash

The "wash" option in the CAKE queue discipline is used to address the issue of frequently mis-marked traffic entering or exiting a DiffServ (Differentiated Services) domain. When traffic passes through networks or providers, there is a chance that the DSCP (Differentiated Services Code Point) markings could be modified or incorrectly assigned.

To illustrate this with a real-life example and analogy:

Let's imagine you are running a busy airport. Different airlines have assigned different priority levels to their passengers based on their ticket classes (economy, business, first-class). These priority levels are analogous to the DSCP markings in a network.

However, as passengers move through various transit airports, there is a possibility that the assigned priority levels could be changed or mis-marked due to different airline policies or inconsistencies at the transit airports. This is similar to the mis-marking of traffic in transit networks.

Now, in the context of the airport, let's assume that upon arrival at your airport, passengers are divided into separate queues based on their priority levels. This division ensures that passengers in higher-priority classes are processed more quickly and receive better service.

However, because the priority levels assigned at previous airports may not be reliable, it becomes necessary to "wash" or clear the assigned priority levels before the passengers enter the queues at your airport. This is similar to the "wash" option in CAKE, which clears extra DSCP markings after priority queuing has taken place.

In situations where inbound traffic's DSCP markings cannot be trusted (similar to cases like Comcast Cable), it is recommended to use a single queue "besteffort" mode with the "wash" option. This means that all incoming traffic is treated as the default "best effort" class, and any potentially unreliable or mis-marked DSCP values are cleared before further processing.

In our airport analogy, using a single queue "besteffort" mode with "wash" would mean that regardless of the priority assigned at previous airports, all passengers are initially treated as standard passengers (best effort class) until their priority can be accurately determined and assigned at your airport.

This approach ensures that even if the DSCP markings of incoming traffic have been mis-marked or modified during transit, the traffic is treated fairly and uniformly within your network, without relying on potentially unreliable markings from external sources.

By applying the "wash" option in CAKE, network administrators can mitigate the impact of mis-marked or modified DSCP markings, providing a more consistent and reliable Quality of Service (QoS) treatment within their network domain.

Diffserv RFC2474 and RFC2475

Diffserv (Differentiated Services) is a mechanism used to prioritize and classify network traffic based on the Diffserv field in the IP packet header. CAKE (Common Applications Kept Enhanced) provides Diffserv support, allowing traffic to be divided into "tins" (traffic classes) and applying different treatment to each tin. Here's a breakdown of the Diffserv presets in CAKE along with real-world examples:

besteffort -The "besteffort" preset in CAKE disables priority queuing and places all traffic into a single tin. This means that all traffic is treated equally without any specific prioritization. *This preset can be suitable for non-critical or low-priority traffic, such as general web browsing or background file downloads, where equal treatment is sufficient.*

precedence - The "precedence" preset enables the legacy interpretation of the TOS (Type of Service) "Precedence" field. However, its usage on the modern internet is discouraged, as it is an outdated mechanism. *In the past, this field was used to indicate different levels of priority, such as high, medium, or low, but it is no longer widely used or recommended.*

diffserv4 - The "diffserv4" preset provides a general-purpose Diffserv implementation with four tins:

1. **Bulk**: This tin corresponds to CS1 (Class Selector 1) or LE (Low Extra), and it has a threshold of 6.25%. Traffic in this tin typically has a low priority.
2. **Best Effort**: This tin is for general traffic that doesn't fall into any specific Diffserv class. It has a threshold of 100%, meaning it receives all remaining bandwidth.
3. **Video**: This tin encompasses AF4x, AF3x, CS3, AF2x, CS2, TOS4, and TOS1. It has a threshold of 50%, providing a moderate priority for video traffic.
4. **Voice**: This tin covers CS7, CS6, EF (Expedited Forwarding), VA (Voice Admit), CS5, and CS4. It has a threshold of 25%, giving high priority to voice traffic.

In a network where video streaming, voice over IP (VoIP), and general internet traffic coexist, this preset can ensure that video and voice traffic receive appropriate priority, while bulk and best-effort traffic are handled accordingly.

diffserv3 (default) - The "diffserv3" preset is the default Diffserv implementation in CAKE, providing three tins:

1. **Bulk**: Similar to the "diffserv4" preset, this tin represents CS1 or LE with a 6.25% threshold, serving as a low-priority traffic class.
2. **Best Effort:** This tin is for general traffic and has a threshold of 100%, treating all remaining traffic equally.
3. **Voice**: This tin covers CS7, CS6, EF, VA, and TOS4. It has a threshold of 25% and applies a reduced CoDel (Controlled Delay) interval, giving high priority to voice traffic.

In a network where voice traffic requires high priority, such as a VoIP system, while other traffic falls into a general category, the "diffserv3" preset can ensure appropriate priority for voice packets while maintaining fairness for other traffic.

diffserv8 - is an more advances purpuse diffserver with 8 tins:

The 8 classes in DiffServ8 are mapped to different types of network traffic based on their importance, using decimal values for Differentiated Services Code Point (DSCP):

1. **Network Control** (48-63): Highest priority, often used for critical network traffic like routing information.
2. **Telephony** (46): Traffic sensitive to latency, such as VoIP.
3. **Signaling** (32-47): Control signals for real-time applications.
4. **Multimedia Conferencing** (24-31): Video conferencing traffic.
5. **Real-time Interactive** (40): Interactive applications, such as gaming.
6. **Multimedia Streaming** (16-23): Streaming video and audio.
7. **Low Latency Data** (8-15): Traffic requiring low latency, like financial transactions.
8. **Best Effort** (0): Default traffic class with no special priority.

"unlimited" or "autorate-ingress"

When using the "unlimited" or "autorate-ingress" option in the CAKE queue discipline, CAKE automatically determines the download speed based on observed packet arrival times. Here's how it works:

1. Monitoring Packet Arrival Times: CAKE continuously monitors the arrival times of incoming packets. It keeps track of the time intervals between consecutive packets to estimate the rate at which packets are being received.
2. Calculating Average Packet Arrival Rate: CAKE calculates the average packet arrival rate based on the observed arrival times. By dividing the number of received packets by the total time elapsed, it determines the average rate at which packets are arriving.
3. Deriving Download Speed: From the average packet arrival rate, CAKE infers the download speed or throughput of the network connection. It assumes that the packet arrival rate corresponds to the download speed, given that each packet represents a certain amount of data.
4. Dynamic Adjustment: As CAKE continues to monitor packet arrival times, it adjusts the download speed estimation dynamically. If the arrival rate increases, CAKE will update the download speed estimate to reflect the higher throughput. Conversely, if the arrival rate decreases, CAKE will adjust the estimate accordingly.

By automatically determining the download speed, CAKE adapts to changes in the network conditions and ensures that traffic shaping and queuing algorithms are adjusted to optimize the utilization of available bandwidth.

It's worth noting that while CAKE can estimate the download speed based on packet arrival times, it does not have direct knowledge of the link capacity or the true download speed as reported by the network equipment or service provider. Instead, it relies on observed packet arrival rates to approximate the download speed for traffic shaping purposes.