UNITED STATES DISTRICT COURT
WESTERN DISTRICT OF TEXAS
WACO DIVISION

Intellectual Ventures I LLC and Intellectual Ventures II LLC,	)	Civil Action No. 6:20-cv-749
Plaintiffs,	)	
v.	)	
Arista Networks, Inc.	)	
Defendant.	)	**JURY TRIAL DEMANDED**

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiffs, Intellectual Ventures I LLC and Intellectual Ventures II LLC (together "IV"), for their complaint against defendant, Arista, Inc. ("Arista"), hereby allege as follows:

THE PARTIES

1. Intellectual Ventures I LLC ("Intellectual Ventures I") is a Delaware limited liability company having its principal place of business located at 3150 139th Avenue SE, Bellevue, Washington 98005.

2. Intellectual Ventures II LLC ("Intellectual Ventures II") is a Delaware limited liability company having its principal place of business located at 3150 139th Avenue SE, Bellevue, Washington 98005.

3. Upon information and belief, Arista is a Delaware corporation with its principal place of business at 5453 Great America Parkway, Santa Clara, California. Arista has a regular and established place of business in this District at its Research and Development Office located

at The Terrace, Building II, Suite 420, 2700 Via Fortuna, Austin, Texas. Arista may be served with process through its registered agent, Corporation Service Company, 251 Little Falls Drive, Wilmington, Delaware 19808.

JURISDICTION

4. This is a civil action for patent infringement under the patent laws of the United States, 35 U.S.C. § 271, *et seq.* This Court has subject matter jurisdiction under 28 U.S.C. §§ 1331 and 1338(a).

5. This Court has general personal jurisdiction over Arista because Arista is engaged in substantial and not isolated activity at its regular and established places of business within this judicial district. This Court has specific jurisdiction over Arista because Arista has committed acts of infringement giving rise to this action and has established more than minimum contacts within this judicial district, such that the exercise of jurisdiction over Arista in this Court would not offend traditional notions of fair play and substantial justice.

6. Venue is proper in this judicial district pursuant to 28 U.S.C. §§ 1391(b)-(c) and 1400(b) because Arista maintains a regular and established place of business and has committed acts of patent infringement within this judicial district.

FACTUAL BACKGROUND

7. Intellectual Ventures Management, LLC ("Intellectual Ventures") was founded in 2000. Since then, Intellectual Ventures has been involved in the business of invention. Intellectual Ventures fosters inventions and facilitates the filing of patent applications for those inventions; collaborates with others to develop and patent inventions; and acquires and licenses patents from individual inventors, universities, corporations, and other institutions. A significant

aspect of Intellectual Ventures' business is managing the plaintiffs in this case, Intellectual Ventures I and Intellectual Ventures II.

8. One founder of Intellectual Ventures is Nathan Myhrvold, who worked at Microsoft from 1986 until 2000 in a variety of executive positions, culminating in his appointment as the company's first Chief Technology Officer ("CTO") in 1996. While at Microsoft, Dr. Myhrvold founded Microsoft Research in 1991, and was one of the world's foremost software experts. Between 1986 and 2000, Microsoft became the world's largest technology company.

9. Under Dr. Myhrvold's leadership, IV acquired more than 70,000 patents covering some important inventions of the Internet era. Many of these inventions were created during or just after Dr. Myhrvold's successful tenure at Microsoft.

10. A significant consequence of the emergence and widespread usage of the Internet has been the migration of computing from isolated environments centered around large mainframe systems, to distributed systems implemented on large numbers of physical computers that often each host large numbers of virtual private servers. As a result, a very large number of virtual private servers have been deployed that each require a secure and stable file system. File systems that were known at the turn of this century, however, proved insufficient as they required copying the entire file system of a host computer for use by each virtual private server thus wasting an immense amount of storage and compute power, or exposed users to security flaws by insufficiently isolating one user's data from another user's data. They also could not be efficiently backed up. Therefore, there was a need for file systems that could serve many virtual private servers without requiring extensive copying or wasted storage space, and that could be efficiently backed up.

11. Another consequence of the emergence and widespread usage of the Internet has been the creation of massive "data centers," which are physical facilities containing large numbers of computers that need to be interconnected with each other, as well as with computers in other data centers. Such data centers were being tasked with processing massive amounts of data at ever-increasing speeds. Data centers consequently required networking devices (e.g., switches/routers) that could each switch massive amounts of data between large numbers of inputs and outputs. This in turn made it desirable to develop a higher throughput and non-blocking switching fabric. At the same time however, such a fabric needed to avoid the complexity of resorting to a material speed-up in its operation, relative to the operation of the inputs and outputs of the switch. Such a fabric also needed to avoid causing data units to problematically fall out-of-sequence as they flowed from the inputs to the outputs.

12. Another consequence of the emergence and widespread usage of the Internet has been the proliferation of distributed computing platforms. Previously existing distributed computing platforms did not optimally manage, control and coordinate the various client systems comprising such a platform as it worked on a single project, such as a network site testing project. While such a platform could work on a project by statically assigning various portions of the project to different client systems, it was not possible to dynamically manage and control the systems in a coordinated way, for instance by automatically increasing the number of systems as they were working on the project, in response to some pre-determined condition that impacts the desired processing power assigned to the project. Such a shortcoming was problematic when attempting to work on a project that required unexpected and extreme surges in processing, such as surges in computation required to test a network site using a simulated Denial of Service attack (DoS).

13. Arista provides networking solutions and services to its customers. Arista's product and service offerings include: its 7800R, 7500R and 7280R series of switches (and applicable line cards), Arista Extensible Operating System (EOS) including but not limited to EOS which is sold as part of every Arista networking device, a containerized version of EOS (cEOS) which is sold as a containerized application for deployment on third-party hardware, Virtual EOS (vEOS) which is sold for deployment on VM-based hypervisors running on third-party hardware, CloudEOS which enables deployment of various combinations of EOS instances (e.g., vEOS, and/or cEOS) across public and private clouds, and Arista CloudEOS Router for Kubernetes, which comprises a combination of CloudVision (Arista's network management platform) and cEOS. With 15 worldwide offices, Arista markets and sells these solutions and services throughout the globe, including in the United States and Texas.

THE PATENTS-IN-SUIT

15. On September 9, 2003, the PTO issued United States Patent No. 6,618,736 ("the '736 patent"), titled TEMPLATE-BASED CREATION AND ARCHIVAL OF FILE SYSTEMS. The '736 patent is valid and enforceable. A copy of the '736 patent is attached as Exhibit A.

16. Intellectual Ventures I LLC is the owner and assignee of all rights, title and interest in and to the '736 patent, and holds all substantial rights therein, including the rights to grant licenses, to exclude others, and to enforce and recover past damages for infringement of that patent.

17. The inventions claimed in the '736 patent were conceived while its inventor, Paul Menage, worked at Ensim Corporation. Dr. Menage is highly respected in his field with over 20 years of experience at companies such as Ensim, Google, and Facebook, to name a few. Dr. Menage holds a bachelor's degree and a Ph.D. in computer science from the University of

Cambridge. Dr. Menage has published several papers and articles on containerization, resource management and virtualization throughout his career and played integral roles in implementing solutions in his capacity at Google and Facebook.

18. The ’736 patent covers a system, method and/or apparatus for an improved way to create, manage and archive file systems, particularly in a virtualized environment, through the use of shared storage units and private storage units, combined into templates and correlated via a usage map. This novel approach allows for the creation and management of separate file systems for a plurality of virtual private servers running on a physical host computer vying for the resources of that host, without requiring extensive copying or wasted storage space. This in turn enables users to gain more storage resources from their existing computers without the need to purchase access to additional hardware storage resources for their existing virtual private servers. It also enables the efficient backing up of a file system of a virtual private server, in which a snapshot of the file system at a particular point in time may be obtained. This was not possible with preexisting file systems.

19. On February 6, 2007, the PTO issued United States Patent No. 7,173,931 (“the ’931 patent”), titled SCHEDULING THE DISPATCH OF CELLS IN MULTISTAGE SWITCHES. The ’931 patent is valid and enforceable. A copy of the ’931 patent is attached as Exhibit B.

20. Intellectual Ventures I is the exclusive licensee of the ‘931 patent, and holds all substantial rights therein, including the rights to grant licenses, to exclude others, and to enforce and recover past damages for infringement of that patent.

21. The inventions of the ‘931 patent were developed by H. J. Chao and Eiji Oki, both world-renowned researchers in their respective fields. Mr. Chao is an expert in networking, datacenters and switches/routers, and currently heads the High-Speed Networking Lab at NYU

Polytechnic Institute, where he has been a professor since 1992. Mr. Chao has written over 200 journal and conference papers in the field of networking, is a named inventor on 58 patents and has been recognized as a top expert in his field by the National Academy of Inventors and the Institute of Electrical and Electronics Engineers (IEEE), among others. Mr. Chao holds a bachelor's and master's degree in electrical engineering from National Chiao Tung University, Taiwan, and a PhD in Electrical Engineering from Ohio State University.

22. Mr. Oki is an expert in the fields of optical networks, design and control networks, and networking systems with a focus on optimization and algorithms. He is currently a professor at Intelligent Communication Networks (Oki Lab) at the Kyoto University Graduate School of Informatics. Mr. Oki is a prolific researcher and inventor; is a named inventor on numerous patents, authored or co-authored over 490 technical papers and articles, and has been recognized by the IEEE for outstanding contributions to the field of high-performance packet switching and path computation technologies.

23. The '931 patent covers a system, method and/or apparatus for an improved scheduling mechanism and architecture for the dispatch of cells (data units created by segmenting packets received from or sent over network links) as they traverse a multistage network switch from an input port ("input") to an output port ("output"). The invention avoided problems with prior art single-stage switches, which become ineffective once scaled to exceed certain throughputs. The invention also improved upon then existing multistage switches, which required a material increase of the speed at which the fabric stage operated relative to the speed at which the input/output modules operated (known as fabric "speed-up"), or caused cells to problematically fall out-of-sequence as they traversed the fabric stage. The '931 patent avoided these problems with then existing switch fabrics by providing a novel cell dispatch scheduling

architecture for use in a multi-stage switch, which architecture comprised input/output modules, and fabric modules for interconnecting the input/output modules. The novel architecture of the '931 patent also comprised virtual output queues (VoQs) that each queued cells arriving at each input module that were destined for a given port on a given output module, at the input module. The fabric modules of the novel '931 patent's architecture are able to connect any VoQ on the switch to its respective given output module/port, using varying connections through the central/fabric modules that are determined by a credit request/grant loop originating at the input module. Thereby, the '931 patent's novel architecture enabled an extremely high-throughput and non-blocking switch fabric, that avoided the resequencing of cells, and that avoided any material speed-up of the switch relative to the input/output modules.

24. On February 15, 2011, the Patent and Trademark Office (PTO) issued United States Patent No. RE 42,153 ("the '153 patent"), titled DYNAMIC COORDINATION AND CONTROL OF NETWORK CONNECTED DEVICES FOR LARGE-SCALE NETWORK SITE TESTING AND ASSOCIATED ARCHITECTURES. The '153 patent is valid and enforceable. A copy of the '153 patent is attached as Exhibit C.

25. Intellectual Ventures II is the owner and assignee of all rights, title and interest in and to the '153 patent, and holds all substantial rights therein, including the rights to grant licenses, to exclude others, and to enforce and recover past damages for infringement of that patent.

26. The inventions of the '153 patent were developed by Edward A. Hubbard, Krishnamurthy Venkatramani, David P. Anderson, Ashok K. Adiga, Greg D. Hewgill, and Jeff A. Lawson. Dr. Anderson is a research scientist at the Space Sciences Laboratory at U.C. Berkeley and an adjunct professor of computer science at the University of Houston. He helped

create and now leads the SETI@home software project, developed the first distributed system for digital audio editing, and served as CTO of United Devices which developed software for distributed computing systems in the early 2000s. Jeff A. Lawson is a software engineer with extensive experience in software development, including with respect to distributed computing. Early in his career, Mr. Lawson worked for NASA's Jet Propulsion Laboratory as a software developer working on image acquisition software flown aboard NASA Space Shuttles. Mr. Lawson, like Mr. Anderson, previously worked at United Devices, designing, architecting, implementing, and supporting highly scalable distributed computing software for enterprise customers. He also co-founded distributed.net, a large-scale distributed computing network of over 50,000 computers worldwide. The other co-inventors of the '153 patent have similarly illustrious credentials.

27. The '153 patent covers a system, method and/or apparatus for the dynamic coordination and control of network connected systems that collectively perform distributed processing projects such as the testing of a network site. More specifically, in the novel architecture covered by the '153 patent, distributed processing of a project collectively occurs on a plurality of client systems that each participate in the project in large part by executing a client agent program. Throughout such distributed project processing, poll communications from the client systems are received at a server to enable it to form a dynamic snapshot of current overall project status. Analysis of the dynamic snapshot status information is then performed by the server to determine if it should decrease or increase the number of client systems that are actively participating in the project, and corresponding poll response communications are sent from the server to the client systems. The poll communications and poll response communications are used by the novel system to coordinate the ongoing project processing

performed by the client systems. This novel '153 patented system differs from prior art systems by using status snapshots about a project generated from poll communications to dynamically control and coordinate project activities occurring across the client systems as those activities are occurring, rather than using such snapshots and poll communications for use in future resource planning well after those project activities have terminated. The novel '153 patented system thus covers a far more dynamic way to adjust a distributed processing system, in response to any required change in processing requirements.

COUNT I
(Arista's Infringement of U.S. Patent No. 6,618,736)

28. Paragraphs 1-27 are reincorporated by reference as if fully set forth herein.

29. The elements claimed by the '736 patent, taken alone or in combination, were not well-understood, routine or conventional to one of ordinary skill in the art at the time of the invention. Rather, the '736 patent claims and teaches, *inter alia*, an improved way to create, manage and archive file systems in virtualized environments, which was not present in the state of the art at the time of the invention. The invention improved upon then existing file system technology by providing an architecture specific to virtualized environments, in which multiple groups of processes, each organized into discrete logical constructs called virtual private servers (today, implemented for example as Docker containers), could be contending for the resources of a single or limited number of physical host computer(s). The invention further improved on prior art solutions by using the aforementioned architecture to allow for certain shared portions of a single file system to be commonly accessed by such functionally unrelated virtual private servers, while keeping other private portions siloed with respect to private data specific to each such container.

30. Instead of having to provide a separate physical file system for each logical group of processes, or to duplicate any common or shared portions of the file system for each group, the inventions of the '736 patent allowed for the segregation of a file system into shared and private portions via a tiered containerized architecture which could be simultaneously utilized without unnecessary replication or insecure isolation methods relative to prior art systems.

31. The invention represented a technical solution to an unsolved technological problem. The written description of the '736 patent describes, in technical detail, each of the limitations in the claims, allowing a person of skill in the art to understand what those limitations cover, and therefore what was claimed, and also understand how the non-conventional and non-generic ordered combination of the elements of the claims differ markedly from what had been performed in the industry prior to the inventions of the '736 patent. More specifically, the claims of the '736 patent recite methods and systems for creating and archiving one or more file systems within one or more servers, that comprise a first set of storage units (which can be private to a virtual private server), each corresponding to a second set of storage units (which can be shared between virtual private servers), a usage map for indicating which of the second storage units contain valid data, an interception module for intercepting an attempt to write a data item to a first storage unit, a writing module for writing the data to the corresponding second storage unit, and storing an indication in the usage map that the corresponding second storage unit contains valid data.

32. The system covered by the asserted claims, therefore, differs markedly from the prior systems in use at the time of this invention, which, *inter alia*, lacked the claimed combination of first (which can be private) and second (which can be shared) storage units, that provide for the interception of write commands and reference to usage maps so as to enable the creation and

management of separate file systems for a plurality of virtual private servers (e.g., containers) running on a physical host computer without requiring extensive copying or wasted storage space. Further, the claimed inventions differ from prior art systems by using the claimed tiered architecture to enable file system snapshotting and thus efficient backing up of a file system, which otherwise would be impracticable in the aforementioned virtualized environment.

33. As described above, the ’736 patent is drawn to solving a specific, technical problem arising in the context of virtualized computing file system access and management. Consistent with the problem addressed being rooted in such file system access and management environments, the ‘736 patent’s solutions consequently are also rooted in that same technology and cannot be performed with pen and paper or in the human mind.

34. Arista has directly infringed, and continues to directly infringe, literally and/or by the doctrine of equivalents, individually and/or jointly, at least claims 17 and 36 of the ’736 patent by making, using, testing, selling, offering for sale and/or importing into the United States products and/or services covered by one or more claims of the ’736 patent. Arista’s products and/or services that infringe the ’736 patent include, but are not limited to, Arista’s Extensible Operating System (EOS), and Containerized Extensible Operating System (cEOS), with native Docker integration, including any Arista hardware running EOS and/or cEOS, and any other Arista products and/or services, either alone or in combination, that operate in substantially the same manner.

35. Claim 17 of the ’736 patent is reproduced below:

> *A method for creating and archiving file systems of a plurality of servers, the method comprising:*
> *providing a set of shared storage units;*
> *for each of the plurality of servers:*
> *providing a first set of private storage units, each of the private storage units corresponding to a shared storage unit; and*

providing a first usage map for indicating which of the private storage units contain valid data;
intercepting an attempt to write a data item to a shared storage unit;
writing the data item to the corresponding private storage unit; and
storing an indication in the first usage map that the corresponding private storage unit contains valid data.

36. The Accused Products provide a method for creating and archiving file systems of a plurality of servers. As one non-limiting example, the Accused Products include Arista EOS with native Docker functionality that creates and manages file systems for use by containers running on Arista hardware:

Arista adds Docker support to EOS

Also a million-route net-state repository

All EOS device states are aggregated to the company's CloudVision management platform, which provides streams of telemetry and collects historical and analytical data.

Also announced in the upgrade is support to run Docker containers in EOS, with enhancements to its ContainerTracer software to give sysadmins a better physical and virtual view of containers in their environment.

Containers in EOS

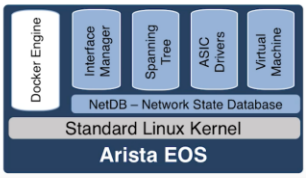
In modern versions of EOS, Docker is included in the OS, as evidenced by the fact that I can issue the `docker` command from Bash just like I did on my Mac in the previous section. This example is run on EOS version 4.21.1F:

Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Docker Containers on EOS

- Simplify development and deployment of applications on switches
- Deploy any of the thousands of apps from Docker Hub
- Deploy enterprise apps with Docker Trusted Registry
- Powerful abstraction for DevOps tools
- Made possible by running a true, unmodified Linux kernel



Copying makes containers efficient

When you start a container, a thin writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored here. Any files the container does not change do not get copied to this writable layer. This means that the writable layer is as small as possible.

When an existing file in a container is modified, the storage driver performs a copy-on-write operation. The

37. As another non-limiting example, the Accused Products include Arista cEOS which is designed to run in a Docker container on any Arista hardware or x86 hardware, and that creates and manages file systems, as seen below:

Union file systems

Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Copying makes containers efficient

When you start a container, a thin writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored here. Any files the container does not change do not get copied to this writable layer. This means that the writable layer is as small as possible.

When an existing file in a container is modified, the storage driver performs a copy-on-write operation. The

38. Furthermore, the Accused Products, as integrated with Docker, provide a set of shared storage units. For instance, when a Docker container runs on EOS, each container is based upon an image which includes a shared set of common OS and application data, as seen below:

Docker registries

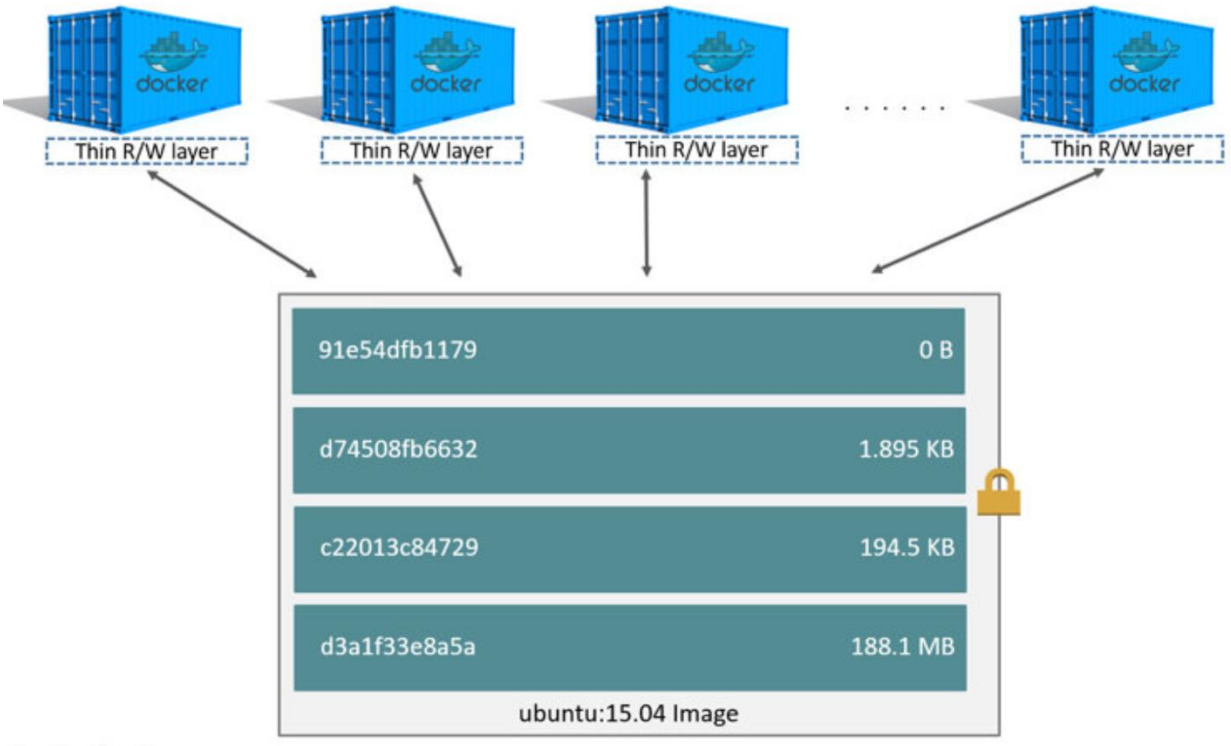
A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the `docker pull` or `docker run` commands, the required images are pulled from your configured registry. When you use the `docker push` command, your image is pushed to your configured registry.

IMAGES

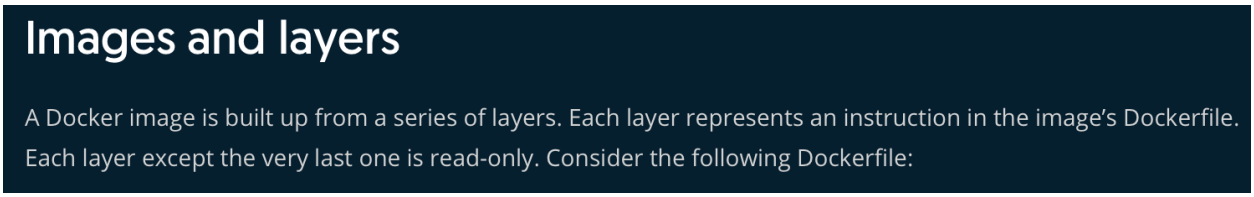
An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the `ubuntu` image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 18.04 image.

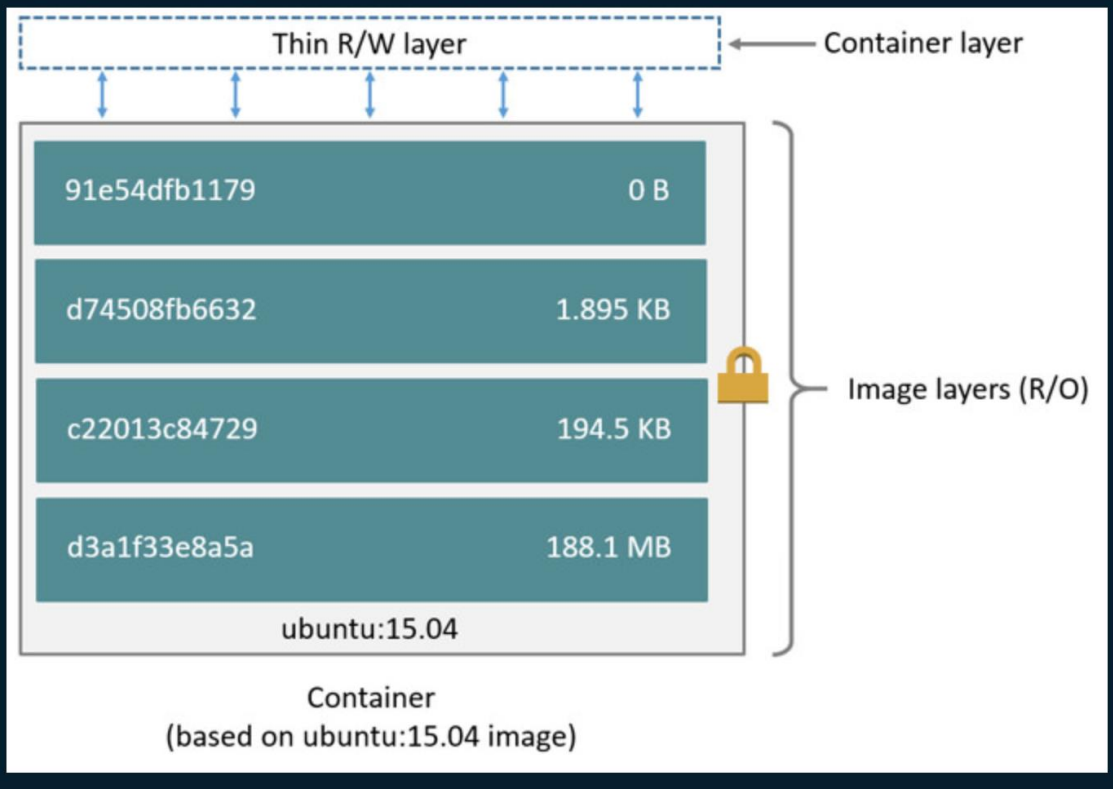


39. As another example, when cEOS is run as a Docker container, Arista provides cEOS images as seen below, which are images for creating a Docker container and therefore include a shared set of common OS and application data:

40. Additionally, the Accused Products, as integrated with Docker, for each of the plurality of aforementioned servers, provides a first set of private storage units, each corresponding to a shared storage unit. As one example only, a Docker container (either running on EOS or with cEOS running in it) includes a read/write layer, or cache, that houses hot data specific to that container, the read/write layer itself corresponds to an underlying layer including read-only shared data, as illustrated below:



Each layer is only a set of differences from the layer before it. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on the Ubuntu 18.04 image.



Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 18.04 image.

41. The Accused Products, as integrated with Docker, further provide a first usage map for indicating which of the private storage units contain valid data. For instance, the Docker layering architecture (whether the container is running on EOS or cEOS is running in the container) provides one or more directories (in the below illustrative example only, referred to as "diff") that shows the set of read/write layers and contents, including what data has changed, as illustrated below:

Copying makes containers efficient

When you start a container, a thin writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored here. Any files the container does not change do not get copied to this writable layer. This means that the writable layer is as small as possible.

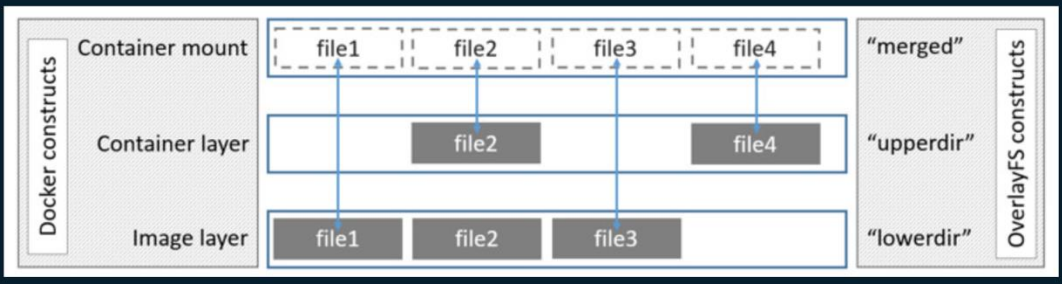
When an existing file in a container is modified, the storage driver performs a copy-on-write operation. The specifics steps involved depend on the specific storage driver. For the `aufs`, `overlay`, and `overlay2` drivers, the copy-on-write operation follows this rough sequence:

- Search through the image layers for the file to update. The process starts at the newest layer and works down to the base layer one layer at a time. When results are found, they are added to a cache to speed future operations.
- Perform a `copy_up` operation on the first copy of the file that is found, to copy the file to the container's writable layer.
- Any modifications are made to this copy of the file, and the container cannot see the read-only copy of the file that exists in the lower layer.

The second-lowest layer, and each higher layer, contain a file called `lower`, which denotes its parent, and a directory called `diff` which contains its contents. It also contains a `merged` directory, which contains the unified contents of its parent layer and itself, and a `work` directory which is used internally by OverlayFS.

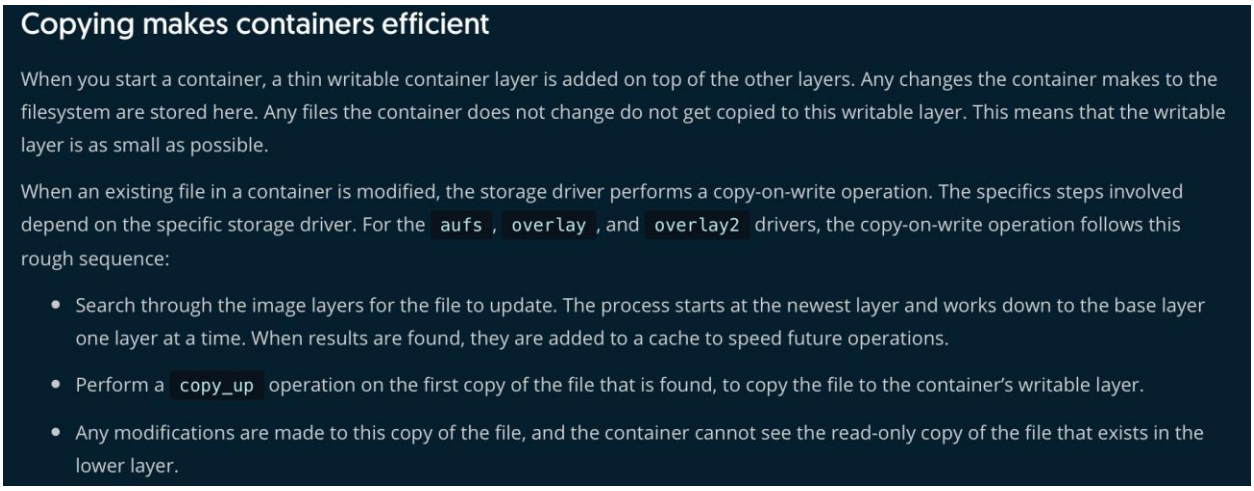
```
$ ls /var/lib/docker/overlay2/223c2864175491657d238e2664251df13b63adb8d050924fd1bfcdb278b866f7

diff  link  lower  merged  work

$ cat /var/lib/docker/overlay2/223c2864175491657d238e2664251df13b63adb8d050924fd1bfcdb278b866f7/lower

l/6Y5IM2XC7TSNIJZZFLJCS6I4I4

$ ls /var/lib/docker/overlay2/223c2864175491657d238e2664251df13b63adb8d050924fd1bfcdb278b866f7/diff/

etc  sbin  usr  var
```

The diagram below shows how a Docker image and a Docker container are layered. The image layer is the `lowerdir` and the container layer is the `upperdir`. The unified view is exposed through a directory called `merged` which is effectively the containers mount point. The diagram shows how Docker constructs map to OverlayFS constructs.



42. In addition, the Accused Products intercept an attempt to write a data item to a shared storage unit. For example, the Docker engine integrated with the Accused Products uses

a "Copy-on-Write" mechanism, whereby any time file data are to be modified by a running container running on EOS, or by cEOS (which itself runs in a container), a copy of the file is copied onto the writeable layer of the container and tracked, thereby leaving the read-only copy in the underlying layer unmodified, as illustrated below:



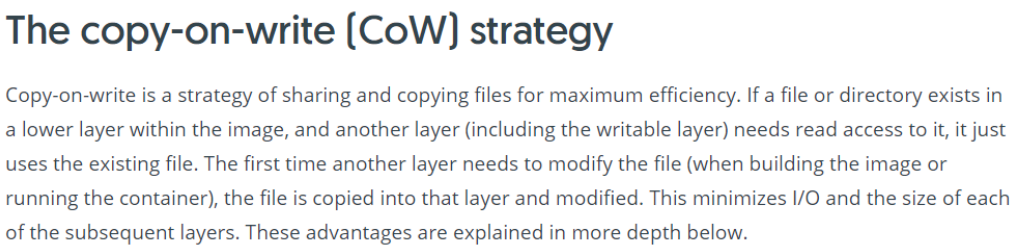
Copying makes containers efficient

When you start a container, a thin writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored here. Any files the container does not change do not get copied to this writable layer. This means that the writable layer is as small as possible.

When an existing file in a container is modified, the storage driver performs a copy-on-write operation. The specifics steps involved depend on the specific storage driver. For the `aufs`, `overlay`, and `overlay2` drivers, the copy-on-write operation follows this rough sequence:

- Search through the image layers for the file to update. The process starts at the newest layer and works down to the base layer one layer at a time. When results are found, they are added to a cache to speed future operations.
- Perform a `copy_up` operation on the first copy of the file that is found, to copy the file to the container's writable layer.
- Any modifications are made to this copy of the file, and the container cannot see the read-only copy of the file that exists in the lower layer.

43. The Accused Products, as integrated with Docker, further write the data item to the corresponding private storage unit. Reiterating the example above, when the attempt to write to a data file that is in the inner read-only layer is made (whether the container is running on EOS or cEOS is running in the container) a copy-on-write operation is performed whereby the data item is copied from the read-only layer to the writeable upper layer where it is modified without alteration of the read-only version in the lower layer, as seen below:

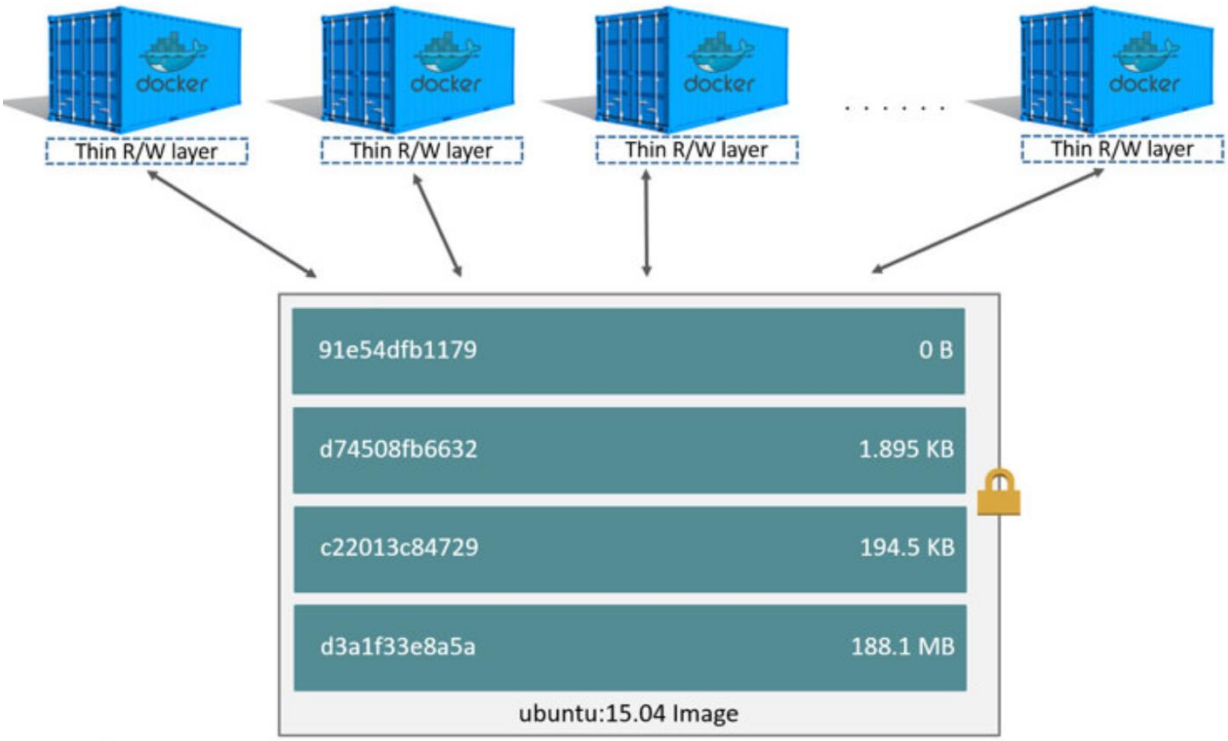


The copy-on-write (CoW) strategy

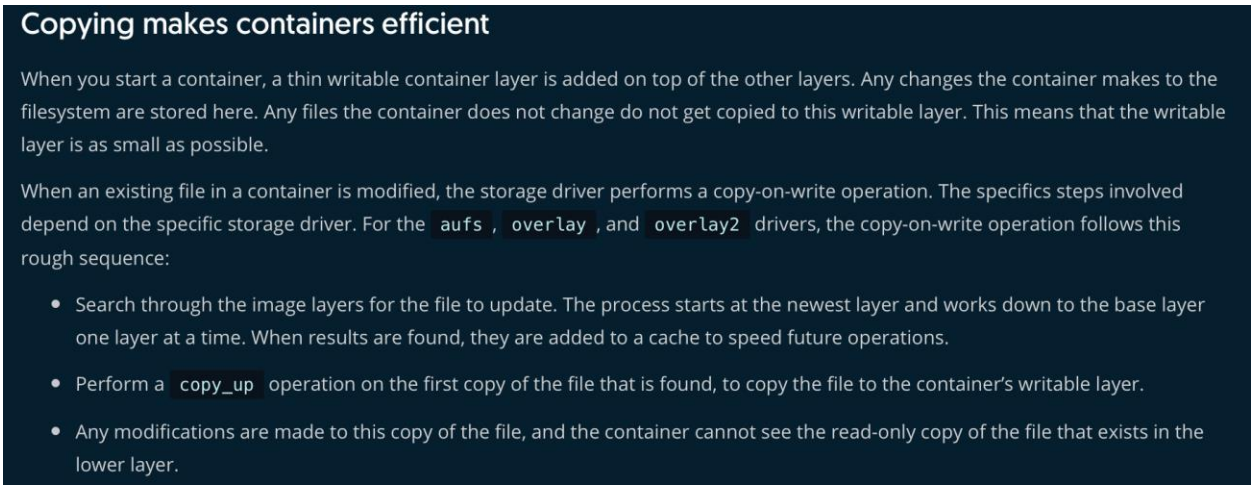
Copy-on-write is a strategy of sharing and copying files for maximum efficiency. If a file or directory exists in a lower layer within the image, and another layer (including the writable layer) needs read access to it, it just uses the existing file. The first time another layer needs to modify the file (when building the image or running the container), the file is copied into that layer and modified. This minimizes I/O and the size of each of the subsequent layers. These advantages are explained in more depth below.

The major difference between a container and an image is the top writable layer. All writes to the container that add new or modify existing data are stored in this writable layer. When the container is deleted, the writable layer is also deleted. The underlying image remains unchanged.

Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 18.04 image.



44. The Accused Products additionally store an indication in the first usage map that the corresponding private storage unit contains valid data. For example, (whether the container is running on EOS or cEOS is running in the container) any changes made to the container's file system are tracked and stored, and one or more indications of the same are stored in the directories noted in the prior elements (e.g., "diff") or are stored in association with the same, as shown below:

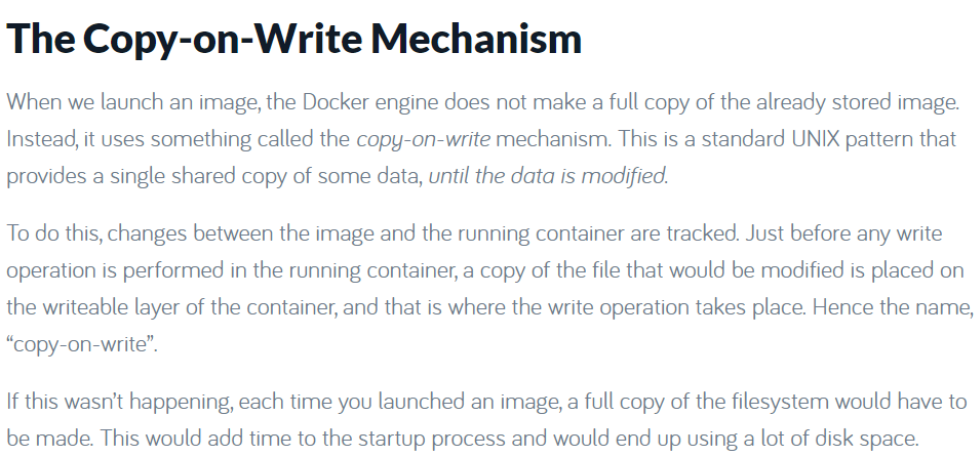


Copying makes containers efficient

When you start a container, a thin writable container layer is added on top of the other layers. Any changes the container makes to the filesystem are stored here. Any files the container does not change do not get copied to this writable layer. This means that the writable layer is as small as possible.

When an existing file in a container is modified, the storage driver performs a copy-on-write operation. The specifics steps involved depend on the specific storage driver. For the `aufs`, `overlay`, and `overlay2` drivers, the copy-on-write operation follows this rough sequence:

- Search through the image layers for the file to update. The process starts at the newest layer and works down to the base layer one layer at a time. When results are found, they are added to a cache to speed future operations.
- Perform a `copy_up` operation on the first copy of the file that is found, to copy the file to the container's writable layer.
- Any modifications are made to this copy of the file, and the container cannot see the read-only copy of the file that exists in the lower layer.



The Copy-on-Write Mechanism

When we launch an image, the Docker engine does not make a full copy of the already stored image. Instead, it uses something called the *copy-on-write* mechanism. This is a standard UNIX pattern that provides a single shared copy of some data, *until the data is modified*.

To do this, changes between the image and the running container are tracked. Just before any write operation is performed in the running container, a copy of the file that would be modified is placed on the writeable layer of the container, and that is where the write operation takes place. Hence the name, "copy-on-write".

If this wasn't happening, each time you launched an image, a full copy of the filesystem would have to be made. This would add time to the startup process and would end up using a lot of disk space.

45. Additionally, Arista has been, and currently is, an active inducer of infringement of the '736 patent under 35 U.S.C. § 271(b) and contributory infringement of the '736 patent under 35 U.S.C. § 271(c) either literally and/or by the doctrine of equivalents.

46. Arista has actively induced, and continues to actively induce, infringement of the '736 patent by intending that others use, offer for sale, or sell in the United States, products and/or services covered by one or more claims of the '736 patent, including but not limited to Arista EOS and cEOS, as integrated with Docker, and including any Arista hardware they are running on, as well as any Arista product and/or service, alone or in combination, that operates in materially the same manner. Arista provides these products and/or services to others, such as customers, resellers and end-user customers, who, in turn, use, provision for use, offer for sale,

or sell in the United States products and/or services that directly infringe one or more claims of the ’736 patent.

47. Arista has contributed to, and continues to contribute to, the infringement of the ’736 patent by others by knowingly providing products and/or services that, when installed and configured result in a system as intended by Arista, directly infringe one or more claims of the ’736 patent.

48. Arista knew of the ’736 patent, or should have known of the ’736 patent, but was willfully blind to its existence. Upon information and belief, Arista has had actual knowledge of the ’736 patent since at least as early as August 17, 2020 and/or the service upon Arista of this Complaint. By the time of trial, Arista will have known and intended (since receiving such notice) that its continued actions would infringe and actively induce and contribute to the infringement of one or more claims of the ’736 patent.

49. Arista has committed, and continues to commit, affirmative acts that cause infringement of one or more claims of the ’736 patent with knowledge of the ’736 patent and knowledge or willful blindness that the induced acts constitute infringement of one or more claims of the ’736 patent. As an illustrative example only, Arista induces such acts of infringement by its affirmative actions of intentionally providing hardware and or software components that when used in their normal and customary way as desired and intended by Arista, infringe one or more claims of the ’736 patent and/or by directly or indirectly providing instructions on how to use its products and/or services in a manner or configuration that infringes one or more claims of the ’736 patent, including those found at one or more of the following:

- https://eos.arista.com/docker-containers-on-arista-eos/
- https://www.youtube.com/watch?v=-h2GISFM0s0
- https://www.arista.com/assets/data/pdf/DockerContainerTracerBulletin.pdf

- https://www.youtube.com/watch?v=1A-jglEkrLk
- *https://s21.q4cdn.com/861911615/files/doc_financials/2016/Arista_Advantage.pdf*
- https://eos.arista.com/veos-ceos-gns3-labs/
- https://eos.arista.com/ceos-lab-in-gns3/

50. Arista has also committed, and continues to commit, contributory infringement by, *inter alia*, knowingly selling products and/or services that when used cause the direct infringement of one or more claims of the '736 patent by a third party, and which have no substantial non-infringing uses, or include a separate and distinct component that is especially made or especially adapted for use in infringement of the '736 patent and is not a staple article or commodity of commerce suitable for substantial non-infringing use.

51. As a result of Arista's acts of infringement, Plaintiffs have suffered and will continue to suffer damages in an amount to be proved at trial.

COUNT II
(Arista's Infringement of U.S. Patent No. 7,173,931)

52. Paragraphs 1-51 are reincorporated by reference as if fully set forth herein.

53. The elements claimed by the '931 patent, taken alone or in combination, were not well-understood, routine or conventional to one of ordinary skill in the art at the time of the invention. Rather, the '931 patent claims and teaches, *inter alia*, an improved scheduling mechanism and architecture for the dispatch of cells in multistage network switches, which was not present in the state of the art at the time of the invention. The invention improved upon then existing multistage network switch scheduling technology and architecture by providing, among other things, a multi-stage switch with input modules for receiving data from network links (first stage), output modules for sending data to network links (third stage), a plurality of center/fabric modules for interconnecting the input and output modules (second stage), virtual output queues (VoQs) at the input modules that each queued cells destined for a particular port in an output

module, and circuitry for connecting any given one of those VoQs to its respective output module using varying connections through the central/fabric modules as determined by a credit request/grant loop originating at the input module. The invention improved on prior art solutions by using unique matching means to determine and secure for each portion of switched data, a path (i) from the VoQ of an input module, to a chosen link between that input module and one of the many potential center/fabric modules on the switch, and then further (ii) from the chosen link to another link between the central/fabric module and the output module associated with the VoQ. This path is determined and secured while the portion of data is queued on the input module. This specific structure avoids buffering in the center/fabric modules and allows for high speed switching without any material speedup of the center/fabric modules relative to the input/output modules.

54. Instead of granting access to a switch fabric using service request schemes that either require material buffering in the center/fabric modules (and thus, resequencing of cells in an output module) or a speed up of the operation of the center/fabric module relative to the operation of the input/output modules, the inventions of the ’931 patent use a scheme whereby for each VoQ on each input module, the input module requests for each cell several connections across the center/fabric modules, of which one is selected for that cell. This allows for high-speed switching across all the center/fabric modules between each of the input/output modules, without resorting to resequencing on the output module or any speed-up in the central/fabric modules relative to prior art systems.

55. The invention represented a technical solution to an unsolved technological problem. The written description of the ’931 patent describes, in technical detail, each of the limitations in the claims, allowing a person of skill in the art to understand what those limitations

cover, and therefore what was claimed, and also understand how the non-conventional and non-generic ordered combination of the elements of the claims differ markedly from what had been used in the industry prior to the inventions of the ’931 patent. More specifically, the claims of the ’931 patent each recite a plurality of central modules each including outgoing links towards output modules, a plurality of input modules each including virtual output queues and outgoing links coupled to the central modules. Further, the claims recite means for matching a non-empty virtual output queue of the input module with an outgoing link in the input module and means for matching the outgoing link of the input module with an outgoing link of one of the central modules. The foregoing combination enables high switch throughput without the cell sequence being reshuffled in the central/fabric modules, or material speedup of the central/fabric modules relative to the input/output modules.

56. The system covered by the asserted claims, therefore, differs markedly from prior systems in use at the time of this invention, which *inter alia* lacked the claimed combination of multistage switch architecture including VoQs at the input modules, and specific structure for scheduling cell dispatch to avoid buffering and speedup in the central modules while maintaining high throughput switching.

57. As described above, the ’931 patent is drawn to solving a specific, technical problem arising in the context of high speed, high throughput multistage network switching. Consistent with the addressed problem being rooted in such multistage network switching environments, the ‘931 patent’s solutions are also rooted in that same technology and cannot be performed with pen and paper, or in the human mind.

58. Arista has directly infringed, and continues to directly infringe, literally and/or by the doctrine of equivalents, individually and/or jointly, at least claim 1 of the ’931 patent by

making, using, testing, selling, offering for sale and/or importing into the United States products and/or services covered by one or more claims of the ’931 patent. Arista’s products and/or services that infringe the ’931 patent include, but are not limited to, the 7800R, 7500R and 7280R series switches, and any other Arista products and/or services, either alone or in combination, that operate in substantially the same manner.

59. Claim 1 of the ’931 patent is reproduced below:

A combination for use in a multi-stage switch, the combination comprising:
- *a) a plurality of central modules, each including outgoing links towards output modules including a plurality of output ports;*
- *b) a plurality of input modules, each including*
- *i. virtual output queues, and*
- *ii. outgoing links coupled with each of the plurality of central modules; and*
- *c) means for matching a non-empty virtual output queue of the input module with an outgoing link in the input module; and*
- *d) means for matching the outgoing link of the input module with an outgoing link of one of the central modules*

wherein high switch throughput can be achieved without speedup of the central modules.

60. As one non-limiting example, the Accused Products are multi-stage switches, as illustrated below:

All Arista 7800R3 systems share a common architecture with identical fabric bandwidth and forwarding capacity per slot. Line cards, supervisor modules and power supplies are common across systems; the only differences are the physical size, fabric/fan module size, number of line card slots and quantity of power supplies. Airflow is always front-to-rear and all data cabling is at the front of the chassis.

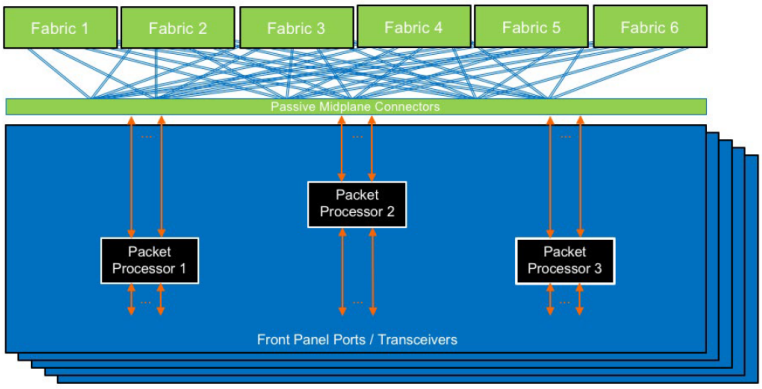


Figure 4: Distributed Forwarding within an Arista 7500R3 Series

Arista 7800R3 Universal Spine platform line card modules utilize packet processors to provide distributed dataplane forwarding. Forwarding between ports on the same packet processor utilizes local switching and no fabric bandwidth is used. Forwarding across different packet processors uses all fabric modules in a fully active/active mode. There is always Virtual Output Queuing (VoQ) between input and output, for both locally switched and nonlocally switched packets, ensuring there is always fairness even where some traffic is local.

Fabric Modules

Within the Arista 7800R3 up to six fabric modules are utilized in an active/active mode. Each fabric module provides up to 45.6 Tbps fabric bandwidth full duplex (22.8 Tbps transmit + 22.8 Tbps receive) to each line card slot, and with six active fabric modules in a 7808R3 system there is 273.6 Tbps (136.8 Tbps transmit and 136.8 Tbps receive) available. If a fabric module were to fail, the throughput of the fabric degrades gracefully. Fabric modules support hot swap and may be inserted or removed while the system is in operation.

61. In addition, the Accused Products comprise a plurality of central modules, each including outgoing links towards output modules including a plurality of output ports. For instance, the Accused Products include several fabric modules (in this example six) (i.e., central modules) that link to output modules (e.g., outgoing line cards) containing several ports as can be seen below:

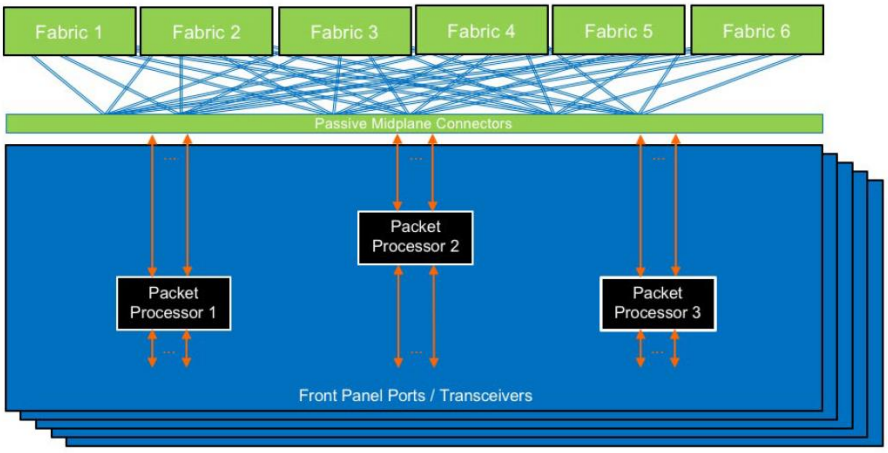


Figure 4: Distributed Forwarding within an Arista 7500R3 Series

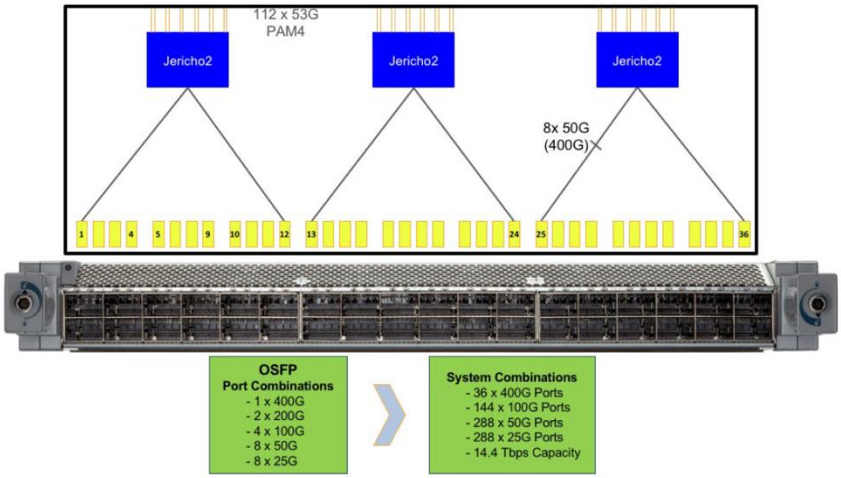


Figure 5: Arista DCS-7800R3-36P-LC module architecture

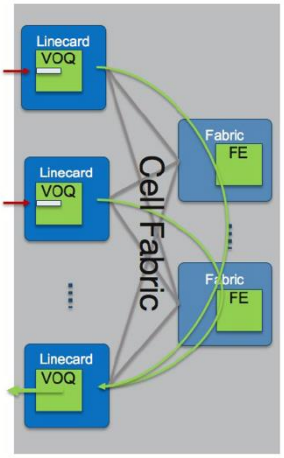
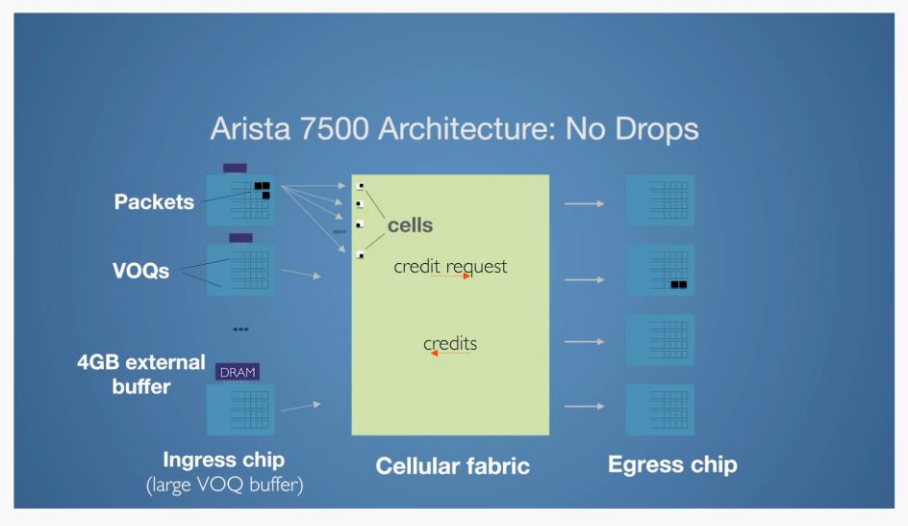


Figure 11: Physical Buffer on Ingress allocated as Virtual Output Queues



62. Furthermore, the Accused Products include a plurality of input modules. As one example, the Accused Products include between 4 and 12 Arista line cards (input modules), as can be seen below:

Table 2: Arista 7800R3 and 7500R3 System Capacity

Characteristic	7504R3	7508R3	7512R3	7804R3	7808R3
Line card Slots	4	8	12	4	8
100G Maximum Density (QSFP100)	144	288	432	192	384
400G Maximum Density (OSFP or QSFP-DD)	96	192	288	144	288
Max forwarding throughput per System (Tbps)	76.8Tbps	153Tbps	230Tbps	115Tbps	230Tbps
Max packet forwarding rate per System (pps)	16Bpps	32Bpps	48Bpps	24Bpps	48Bpps

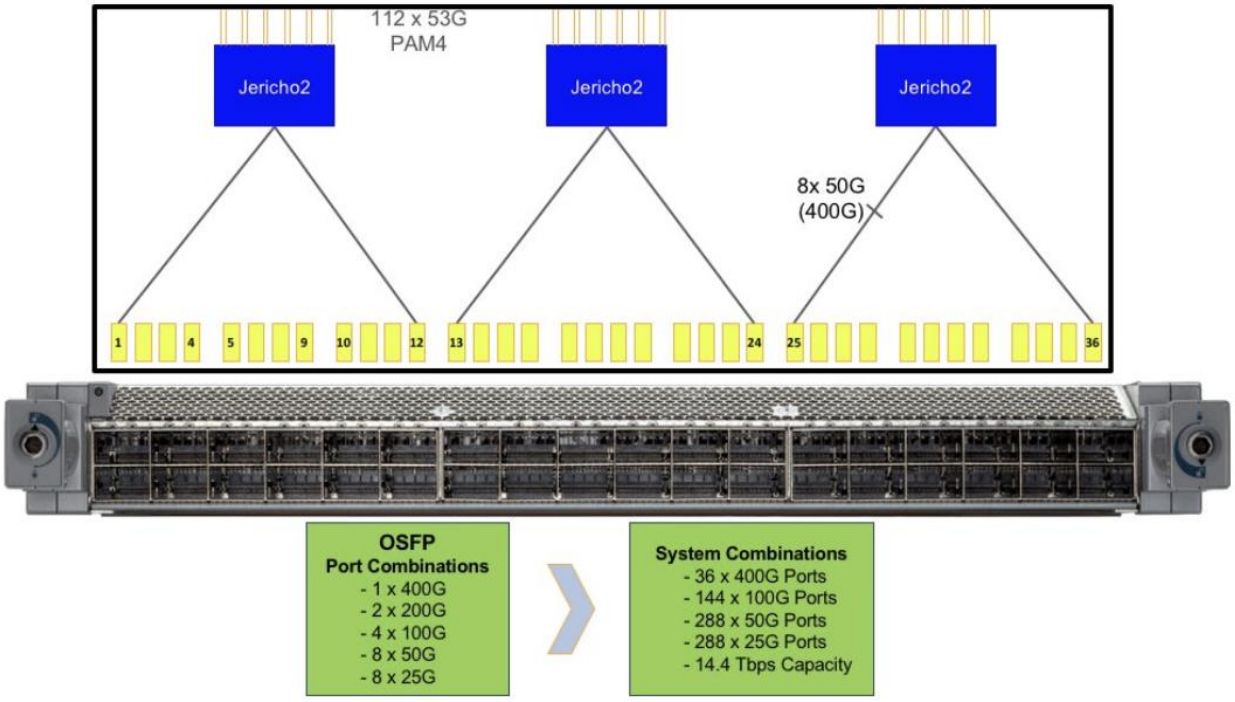


Figure 5: Arista DCS-7800R3-36P-LC module architecture

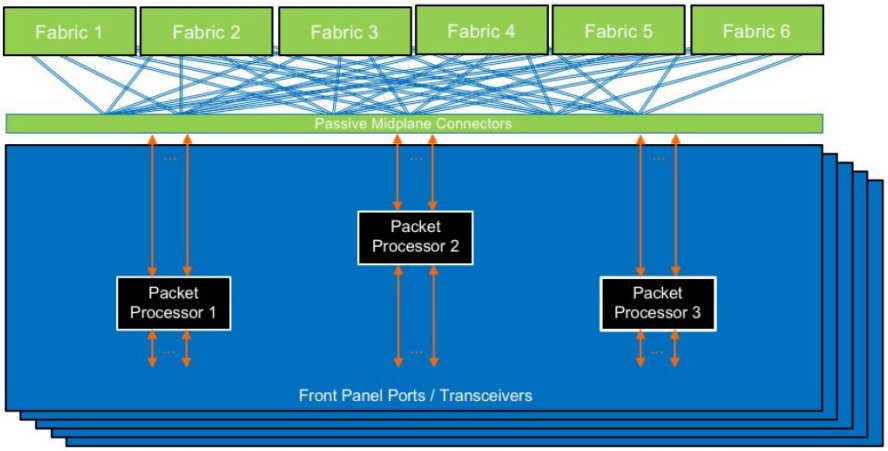
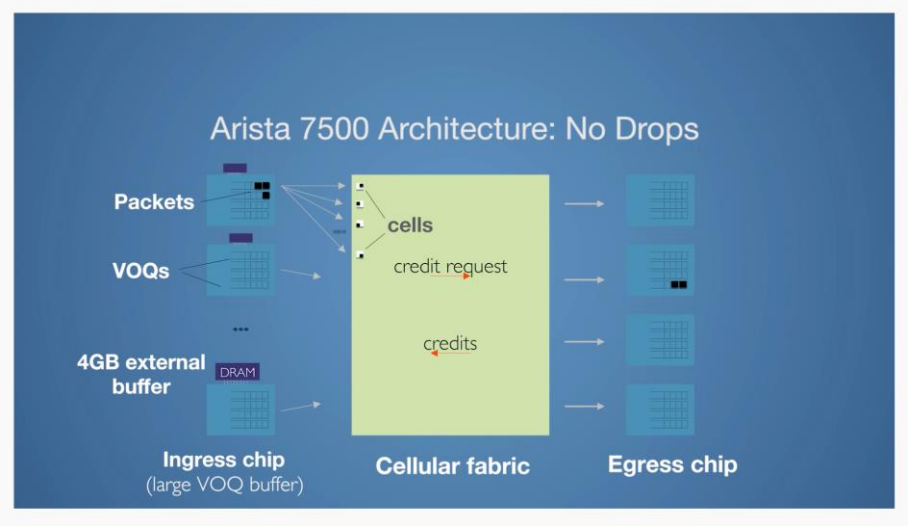


Figure 4: Distributed Forwarding within an Arista 7500R3 Series



63. In addition, the plurality of input modules included in the Accused Products also include virtual output queues. As one non-limiting example, the Arista DCS-7800R3-36P-LC line card (and other line cards, which are input modules) supports VoQs that involve buffering on the input modules as illustrated below:

Arista 7800R3: Platform Overview

The Arista 7800R3 Universal Spine Platform is the evolution of the R-Series family of modular switches, with a consistent architecture of deep buffers, VoQ and non-blocking lossless forwarding. The 7800R Series are initially available in a choice of 4-slot and 8-slot systems that support a rich range of line cards providing high density 100G and 400G with choice of forwarding table scale and MACsec encryption options.

Virtual Output Queuing (VoQ): a distributed scheduling mechanism is used within the switch to ensure fairness for traffic flows contending for access to a congested output port. A credit request/grant loop is utilized and packets are queued in physical buffers on ingress packet processors within VoQs until the egress packet scheduler issues a credit grant for a given input packet.

Arista 7800R3 Universal Spine platforms utilize Virtual Output Queuing (VoQ) where the majority of the buffering within the switch is on the input line card. While the physical buffer is on the input packet processor, it represents packets queued on the output side balanced across all fabric paths. Since a packet is only transferred across the fabric once there is a VoQ grant, there is no queuing within the fabric and there are guaranteed resources to be able to process the frame on the egress packet processor.

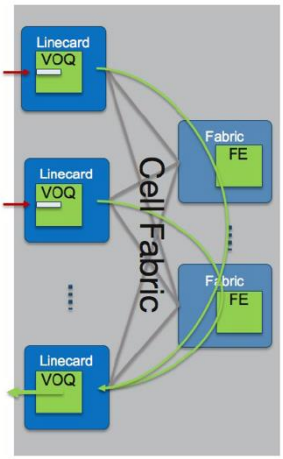
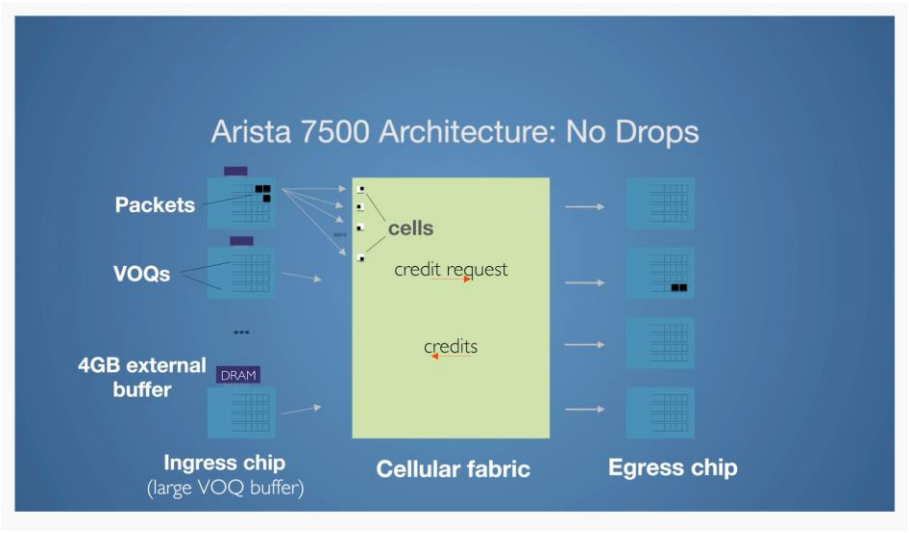


Figure 11: Physical Buffer on Ingress allocated as Virtual Output Queues



64. The plurality of input modules in the Accused Products also have outgoing links coupled with each of the plurality of central modules. For example, the Accused Products are built according to what is called an 'envelope' or 'folding' architecture, in which the input line cards and output line cards (input and output modules, respectively, in which each line card can be an input module or an output module for a given packet) are conceptually configured in a row preceding the central/fabric modules, and in which cells generally travel in a conceptual "U" path through the switch from an input module to a center/fabric module and out to an output module. As shown below the input line cards include outgoing paths to the central modules:

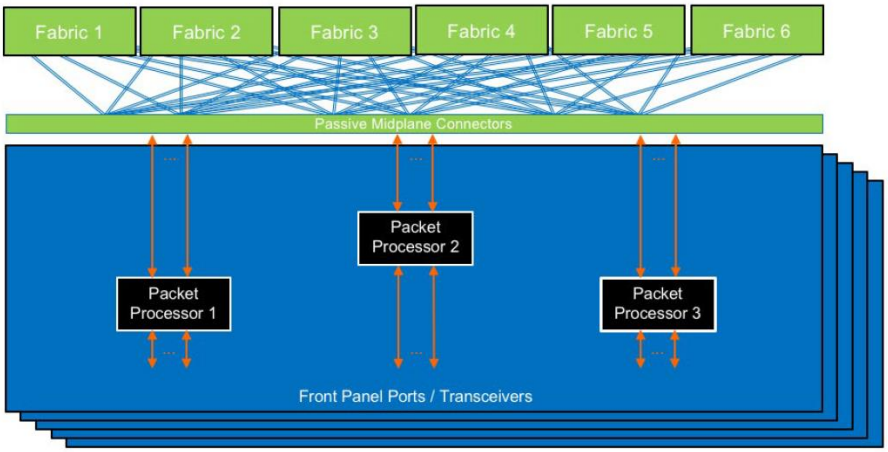


Figure 4: Distributed Forwarding within an Arista 7500R3 Series

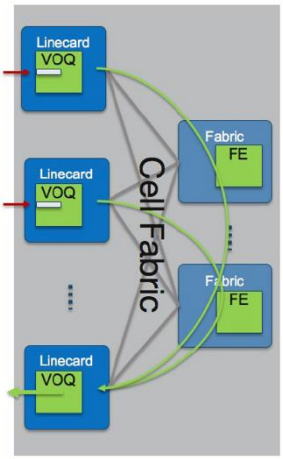
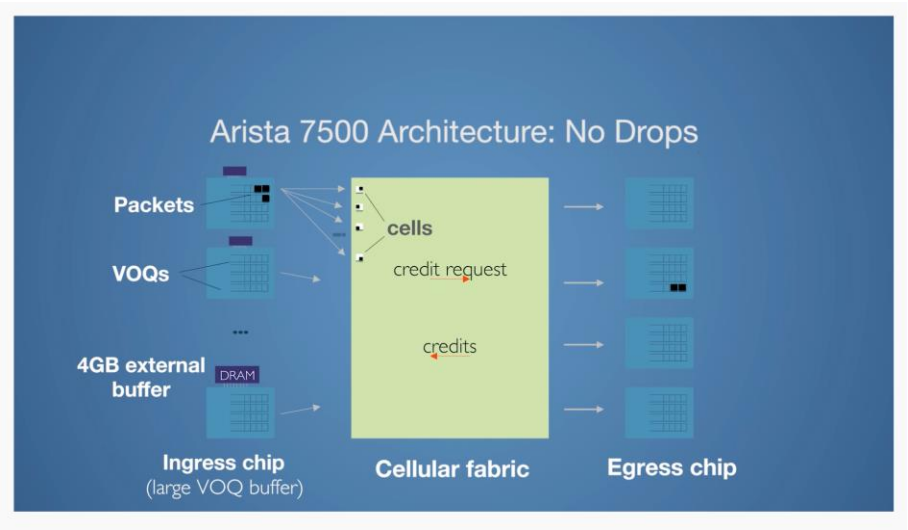


Figure 11: Physical Buffer on Ingress allocated as Virtual Output Queues



65. The Accused Products further include means for matching a non-empty virtual output queue of the input module with an outgoing link in the input module. For instance, as can be seen below, the Accused Products are multi-stage network switches including input modules, center/fabric modules and output modules. The input module can send a request for its VoQs to several central/fabric modules, seeking to establish a connection through one of them for each cell on each VoQ. A cell remains queued on its VoQ on the input module until a VoQ grant message is returned at which point the cell is forwarded through the input module to one of the central modules, as illustrated below:

- **Virtual Output Queuing (VoQ):** a distributed scheduling mechanism is used within the switch to ensure fairness for traffic flows contending for access to a congested output port. A credit request/grant loop is utilized and packets are queued in physical buffers on ingress packet processors within VoQs until the egress packet scheduler issues a credit grant for a given input packet.

Arista 7800R3 Universal Spine platforms utilize Virtual Output Queuing (VoQ) where the majority of the buffering within the switch is on the input line card. While the physical buffer is on the input packet processor, it represents packets queued on the output side (hence, the term virtual output queuing). VoQ is a technique that allows buffers to be balanced across sources contending for a congested output port and ensures fairness and QoS policies can be implemented in a distributed forwarding system.

When a packet arrives into the Ingress Traffic Manager, a VoQ credit request is forwarded to the egress port processor requesting a transmission slot on the output port. Packets are queued on ingress until such time as a VoQ grant message is returned (from the Egress Traffic Manager on the output port) indicating that the Ingress Traffic Manager can forward

The Ingress Traffic Manager stage is responsible for packet queuing and scheduling.

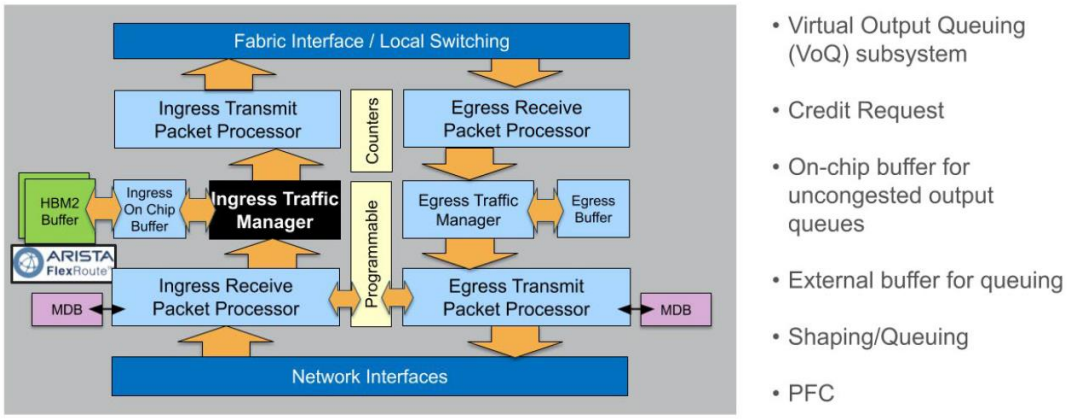


Figure 10: Packet Processor stage 3 (ingress): Ingress Traffic Manager

All available fabric paths are used in parallel to transfer the frame or packet to the output packet processor, with the original group of packets on the same VoQ are packed into 256 byte cells which are forwarded across up to 112 fabric links simultaneously. This

66. The Accused Products include means for matching the outgoing link of the input module with an outgoing link of one of the central modules. For instance, as described above, the Accused Products use VoQs to buffer cells on an input module. Cells are only forwarded to the output modules associated with their respective VoQs after requests for connections across the central modules stage and to an output module have been submitted by the input module, connection grants have been received in response, and connections have been selected. This process results in the definition of each cell's path that comprises a link interconnecting the input module and the selected central/fabric module, and a link interconnecting the selected central/fabric module and the output module. This is depicted below:

The Ingress Transmit Packet Processor stage is responsible for transferring frames from the input packet processor to the relevant output packet processor. Frames arrive at this stage once the output port has signaled, via a VoQ grant message, that it is the allocated slot for a given input packet processor to transmit the packet.

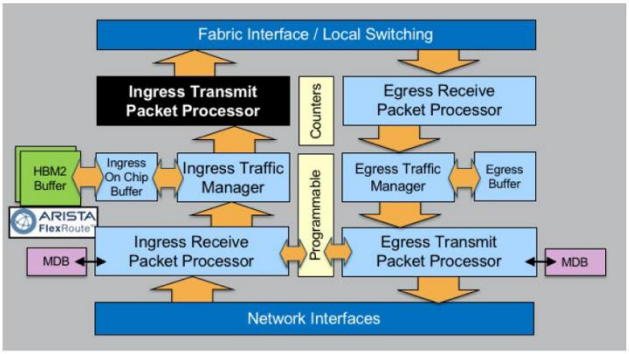


Figure 14: Packet Processor stage 4 (ingress): Ingress Transmit Packet Processor

All available fabric paths are used in parallel to transfer the frame or packet to the output packet processor, with the original group of packets on the same VoQ are packed into 256 byte cells which are forwarded across up to 112 fabric links simultaneously. This mechanism reduces serialization to at most 256 bytes at 50Gbps and ensures there are no hot spots as every flow is always evenly balanced across all fabric paths. Since a packet is only transferred across the fabric once there is a VoQ grant, there is no queuing within the fabric and there are guaranteed resources to be able to process the frame on the egress packet processor.

67. The Accused Products accomplish the above to achieve high throughput without material speedup of the central modules. For example, an analysis of the performance metrics below demonstrates that the central/fabric modules of the Accused Products operate at line speed without any speedup as compared to the input/output modules:

Arista 7800R3: Line Card Architecture

Arista 7800R3 Universal Spine Platform Line Card Layout

Arista 7800R3 line card modules utilize the same packet processor, with the number of packet processors varied based on the number and type of ports on the module. The packet forwarding architecture of each of these modules is essentially the same: a group of front-panel ports (different transceiver/port/speed options) are connected to a packet processor with connections to the fabric modules.

Table 4: Arista 7800R3 Series Line card Module Port Characteristics

Line card	Port (type)	Interfaces					Port Buffer	Fowarding Rate	Switching Capacity
		25G	50G	40G	100G	400G			
7800R3-36P	36 OSFP	-	288	-	144	36	24GB	6.0 Bpps	14.4Tbps
7800R3-48CQ	48 QSFP100	96	96	48	48	-	8GB	2.0 Bpps	4.8Tbps
7800R3K-48CQ	48 QSFP100	96	96	48	48	-	8GB	2.0 Bpps	4.8Tbps

At a system level, the 8-slot Arista 7808R3 with a fabric that scales to 230 Tbps enables 288 x 400G or 384 x 100G (QSFP100) in a 16 RU front to rear power efficient form factor, providing industry-leading performance and density without compromising on features and functionality. The 7808R3 provides a platform for service providers and cloud operators to deliver the next generation of rich services. The 7804R3 4-slot system provides the same capabilities in a more compact system with support for up to 144 400G interfaces and 115Tbps.

Table 1: Arista 7800R3 Key Port and Forwarding Metrics

Characteristic	Arista 7804R3	Arista 7808R3
Chassis Height (RU)	10 RU	16 RU
Linecard Module slots	4	8
Supervisor Module slots	2	2
50G Maximum Density using 8x50G breakout	1152	2304
100G Maximum Density (QSFP100)	192	384
400G Maximum Density (OSFP or QSFP-DD)	144	288
System Usable Capacity (Tbps)	115.2 Tbps (FD)	230.4 Tbps (FD)
Max forwarding throughput per Linecard (Tbps)	14.4 Tbps (DCS-7800R3-36P - 36 x 400G per LC)	
Max forwarding throughput per System (Tbps)	115.2 Tbps (FD)	230.4 Tbps (FD)
Max packet forwarding rate per Linecard (pps)	6 Billion pps (7800R3-36P)	
Max packet forwarding rate per System (pps)	24 Bpps	48 Bpps
Maximum Buffer memory/ System	96 GB	192 GB
Virtual Output Queues / System	More than 2.2 million	

Performance

- 460 Terabits per second fabric capacity
- Up to 96 Billion packets per second
- Up to 28.8 Terabit per second per slot
- Up to 576 wire-speed 400G ports
- 100G, 200G and 400G mode support
- Under 4 microsecond latency (64 bytes)
- 800G Ready

68. Additionally, Arista has been, and currently is, an active inducer of infringement of the ’931 patent under 35 U.S.C. § 271(b) and contributory infringement of the ’931 patent under 35 U.S.C. § 271(c) either literally and/or by the doctrine of equivalents.

69. Arista has actively induced, and continues to actively induce, infringement of the ’931 patent by intending that others use, offer for sale, or sell in the United States, products and/or services covered by one or more claims of the ’931 patent, including but not limited to the 7800R, 7500R and 7280R series switches, and any Arista product and/or service, alone or in combination,

that operates in materially the same manner. Arista provides these products and/or services to others, such as customers, resellers and end-user customers, who, in turn, use, provision for use, offer for sale, or sell in the United States products and/or services that directly infringe one or more claims of the '931 patent.

70. Arista has contributed to, and continues to contribute to, the infringement of the '931 patent by others by knowingly providing products and/or services that, when installed and configured result in a system as intended by Arista, directly infringe one or more claims of the '931 patent.

71. Arista knew of the '931 patent, or should have known of the '931 patent, but was willfully blind to its existence. Upon information and belief, Arista has had actual knowledge of the '931 patent since at least August 17, 2020 and/or as early as the service upon Arista of this Complaint. By the time of trial, Arista will have known and intended (since receiving such notice) that its continued actions would infringe and actively induce and contribute to the infringement of one or more claims of the '931 patent.

72. Arista has committed, and continues to commit, affirmative acts that cause infringement of one or more claims of the '931 patent with knowledge of the '931 patent and knowledge or willful blindness that the induced acts constitute infringement of one or more claims of the '931 patent. As an illustrative example only, Arista induces such acts of infringement by its affirmative actions of intentionally providing hardware and or software components that when used in their normal and customary way as desired and intended by Arista, infringe one or more claims of the '931 patent and/or by directly or indirectly providing instructions on how to use its products and/or services in a manner or configuration that infringes one or more claims of the '931 patent, including those found at one or more of the following:

- https://blogs.arista.com/blog/the-universal-spine-is-born
- https://www.arista.com/assets/data/pdf/Whitepapers/Arista7800R3SwitchArchitectureWP.pdf
- https://www.arista.com/en/solutions/ip-storage-networking#Next-Generation-Platforms
- https://www.arista.com/assets/data/pdf/Datasheets/7800R3-Data-Sheet.pdf
- https://www.youtube.com/watch?time_continue=1&v=2X0b5vDJcfs&feature=emb_logo

73. Arista has also committed, and continues to commit, contributory infringement by, *inter alia*, knowingly selling products and/or services that when used cause the direct infringement of one or more claims of the '931 patent by a third party, and which have no substantial non-infringing uses, or include a separate and distinct component that is especially made or especially adapted for use in infringement of the '931 patent and is not a staple article or commodity of commerce suitable for substantial non-infringing use.

74. As a result of Arista's acts of infringement, Plaintiffs have suffered and will continue to suffer damages in an amount to be proved at trial.

COUNT III
(Arista's Infringement of U.S. Patent No. RE 42,153)

75. Paragraphs 1-74 are reincorporated by reference as if fully set forth herein.

76. The elements claimed by the '153 patent, taken alone or in combination, were not well-understood, routine, or conventional to one of ordinary skill in the art at the time of the invention. Rather, the '153 patent claims and teaches, *inter alia*, an improved dynamic coordination and control architecture for executing projects within a distributed platform that comprises a server system and a plurality of network-connected client systems. Dynamic coordination and control of the execution of a project by the network-connected server and client systems is based on poll communications and responses exchanged between the server system and the client systems. The invention improved upon then existing distributed computing

technology by providing for dynamic snapshot information of a project's status determined based upon poll communications and responses involving the server and client systems, and then taking coordination and control actions related to the ongoing project based on an analysis of the dynamic snapshot information. An example of such coordination and control actions is dynamically increasing or decreasing the amount of client systems actively participating in the project.

77. Instead of statically configuring the coordination and control actions related to such projects, the inventions of the '153 patent allowed for an exchange of data between the client systems performing a distributed project, and the server system that was managing the client systems and that enabled the dynamic coordination and control of processing activities across the client systems. Through poll communications between the client and server systems, for example, the number of client systems participating in the project could be automatically increased or decreased.

78. The invention represented a technological solution to an unsolved technological problem. The written description of the '153 patent describes, in technical detail, each of the limitations in the claims, allowing a person of skill in the art to understand what those limitations cover, and therefore what was claimed, and also to understand how the non-conventional and non-generic ordered combination of the elements of the claims differ markedly from what had been done in the industry prior to the inventions of the '153 patent. More specifically, the asserted claims of the '153 patent each recite poll communications between server and client systems through a network in a distributed computing platform, and the dynamic analysis and coordination of distributed project activities during project operations that lead to, for example, increases or

decreases in the number of client systems participating in the project as the project is still being executed.

79. The system covered by the asserted claims therefore differs markedly from the systems in use prior to this invention, which, *inter alia*, lacked the claimed combination of dynamic interaction between the server and client systems during ongoing project operations. Furthermore, the claimed inventions differ from prior art systems by using dynamic status snapshots about a project generated in part from poll communications to dynamically control and coordinate project activities across the client systems as those activities are occurring, rather than using such snapshots and communications for use in future resource planning relating to a project.

80. As described above, the ’153 patent is drawn to solving a specific, technical problem arising in the context of distributed processing or computing systems. Consistent with the problem addressed being rooted in such complex, distributed approaches to computing, the ’153 patent’s solutions are also rooted in that same technology and cannot be performed with pen and paper or in the human mind.

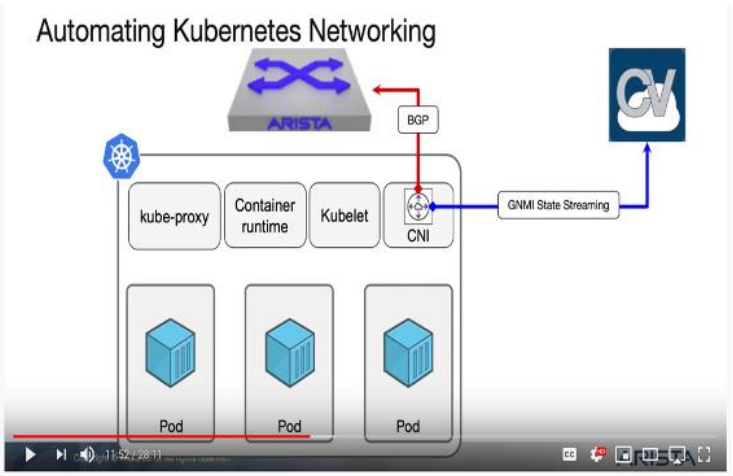
81. Arista has directly infringed, and continues to directly infringe, literally and/or by the doctrine of equivalents, individually and/or jointly, at least claim 18 of the ’153 patent by making, using, testing, selling, offering for sale, and/or importing into the United States products and/or services covered by one or more claims of the ’153 patent. Arista’s products and/or services that infringe the ’153 patent include, but are not limited to, Arista CloudEOS Router for Kubernetes which is sold with:

- cEOS instances, which are container-based instances of Arista EOS software that are each deployed as a Kubernetes Container Network Interface (CNI) or a standalone Kubernetes

container, and that each extend Arista EOS into Kubernetes clusters for auto-provisioning; and

- Arista Cloud Vision (CV), which is implemented as part of a Kubernetes cluster and includes Terraform integrations that enable automated declarative provisioning of the cEOS instances using Kubernetes.

The components of Arista CloudEOS Router for Kubernetes collectively use Kubernetes to orchestrate the dynamic provisioning of Arista networking "nodes" (a node is formed by installing cEOS on a hardware system capable of supporting switching or routing functions). cEOS instances each running on a node and each deployed within a virtual private cloud (VPC) collectively form part of a Kubernetes cluster (a set of nodes that run containerized applications such as cEOS and other containerized applications under control of a Kubernetes controller). cEOS instances provide networking functions on each node to one or more Kubernetes pods (a set of running containers in a Kubernetes cluster). CV, as integrated with a Kubernetes controller, manages a Kubernetes cluster comprising nodes running cEOS instances. Arista CloudEOS Router for Kubernetes, and any other Arista product and/or service that either alone or in combination operates in substantially the same manner, are collectively referenced in this Count as the Accused Products.



11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS

CloudEOS for Kubernetes is available as a software subscription via the following offerings:

Product Number	Product Description
SS-CLOUDEOS-CR-CV-B-1M	CloudEOS Router for Container/Kubernetes Software Subscription License for 1-Month. Includes dynamic routing features (BGP, OSPF), A-care SW support, and 1-Month CloudVision for a single CloudEOS instance (node)

Overview

Arista CloudEOS™ Router for Kubernetes provides an open and scalable solution for customers that are deploying cloud-native infrastructure. Leveraging a containerized version of Arista EOS software (CloudEOS-CR), including the same software binary utilized by all Arista networking platforms, each Kubernetes node now has access to the full power of Arista EOS and CloudVision. This combination provides consistent cloud-grade routing and real-time state streaming telemetry to Kubernetes clusters for the first time.

82. Claim 18 of the ’153 patent is reproduced below:

18. A distributed computing platform having dynamic coordination capabilities for distributed client systems processing project workloads, comprising:

a plurality of network-connected distributed client systems, the client systems having under-utilized capabilities;

a client agent program configured to run on the client systems and to provide workload processing for at least one project of a distributed computing platform; and

at least one server system configured to communicate with the plurality of client systems through a network to provide the client agent program to the client systems, to send initial project and poll parameters to the client systems, to receive poll communications from the client systems during

> *processing of the project workloads, wherein a dynamic snapshot information of current project status is provided based at least in part upon the poll communications from the client systems, to analyze the poll communications utilizing the dynamic snapshot information to determine whether to change how many client systems are active in the at least one project, wherein if a fewer number is desired, including within a poll response communications a reduction in the number of actively participating clients, and if a greater number is desired, adding client systems to active participation in the at least one project within a poll response communications, the server system repeatedly utilizing the poll communications and the poll response communications to coordinate project activities of the client systems during project operations.*

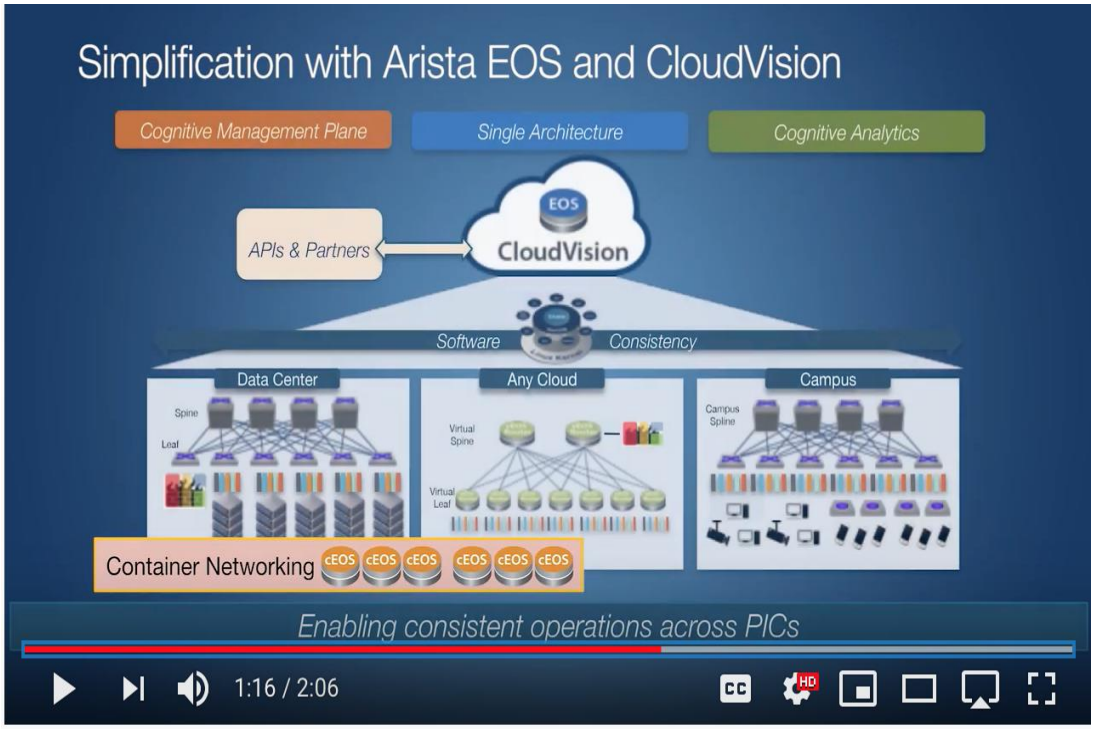
83. The Accused Products, when deployed on Arista's own nodes and servers or those of its customers, constitute a distributed computing platform with dynamic coordination capabilities for distributed client systems that are processing project workloads. As one non-limiting example, the Arista CloudEOS Router for Kubernetes is sold with cEOS instances and CV. A set of cEOS instances each running on a node comprise a plurality of network-connected distributed client systems, and CV comprises a server system configured to communicate with the cEOS instances. Arista Cloud EOS Router for Kubernetes performs projects such as maintaining the requisite set of border gateway protocol (BGP) peering sessions for the servers/switches comprising a VPC based on variables such as real-time application demand being handled by the VPC. It does so by automatically deploying and provisioning cEOS instances by, for example, adding cEOS instances to a VPC, based on poll information exchanges between the cEOS instances and CV that enable the generation of dynamic project status snapshots.

Arista CloudEOS™ Router for Kubernetes provides an open and scalable solution for customers that are deploying cloud-native infrastructure. Leveraging a containerized version of Arista EOS software (CloudEOS-CR), including the same software binary utilized by all Arista networking platforms, each Kubernetes node now has access to the full power of Arista EOS and CloudVision. This combination provides consistent cloud-grade routing and real-time state streaming telemetry to Kubernetes clusters for the first time.

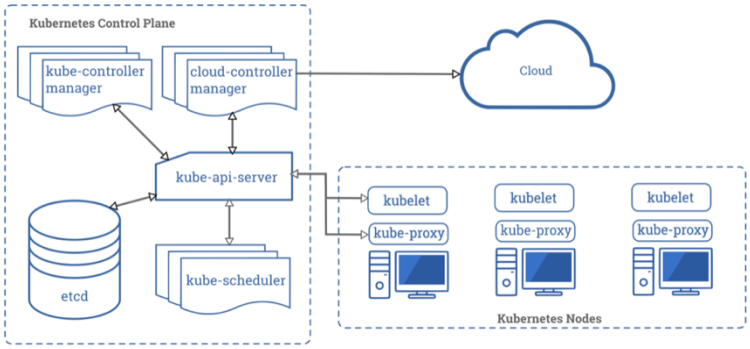
The recommended BGP peering configuration is an iBGP session between CloudEOS on the Kubernetes nodes and an adjacent top-of-rack (ToR) switch, which is acting as a route reflector. Once BGP peering has been established, Kubernetes POD networks are fully routed through the enterprise network without the need for any proprietary encapsulation or out-of-band management.

Arista CloudEOS is then deployed as a Kubernetes daemonset, creating a CloudEOS pod on every node in the cluster to serve as the routing engine.

CloudEOS supports fully elastic pay-as-you-go consumption and scaling in the public cloud, with automatic deployment and provisioning based on real-time application demand. In addition it provides the key features required to achieve integration of network operations with the dynamic cloud operational models required to sustain the benefits of a multi cloud infrastructure, while minimizing operating costs.



Arista Any Cloud for Kubernetes



CloudEOS provides two main capabilities:

1. CloudEOS Multi Cloud, a high-performance virtual machine that normalizes the network connectivity to and between public clouds simplifying the networking operating model for cloud and network operations while enabling declarative software-based provisioning through popular DevOps tools
2. CloudEOS Cloud Native, an instance of EOS deployed as a Kubernetes Container Network Interface or stand-alone Kubernetes container to provide a fully supported, enterprise-class networking stack within Cloud Native environments

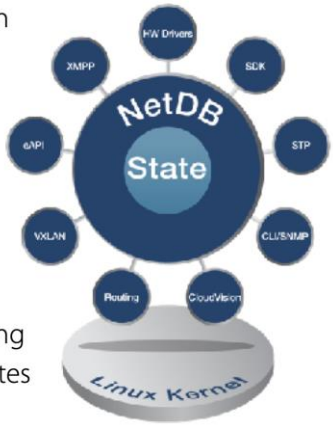
Arista CloudEOS Router - Summary of Key Capabilities

Declarative Cloud Provisioning - using the popular Terraform application a platform engineering team can stand up virtual private cloud instances in multiple public cloud providers and with a single additional line of code per VPC. Terraform will automatically deploy an Arista CloudEOS leaf router, or a high availability pair of routers, and establish secure connectivity to a CNPS for each connected VPC.

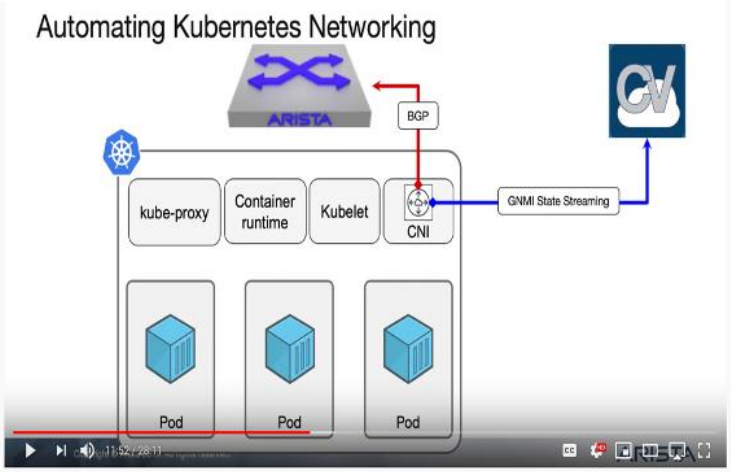
84. The Accused Products, when deployed on distributed processing systems of, for example, Arista or its customers, comprise a plurality of network-connected distributed client systems, the client systems having under-utilized capabilities. As one non-limiting example, in the Accused Products, cEOS instances each running on a node and each deployed within a VPC as part of a Kubernetes cluster, collectively form a plurality of network-connected distributed client systems having under-utilized capabilities.

Overview

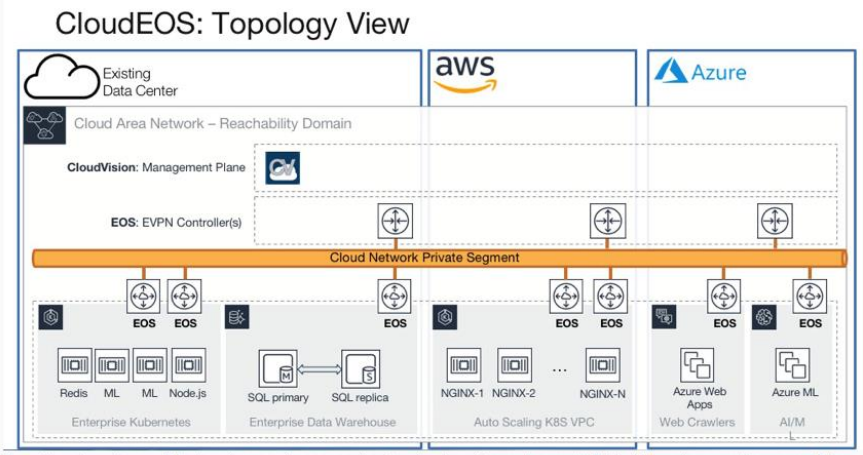
Arista CloudEOS™ Router for Kubernetes provides an open and scalable solution for customers that are deploying cloud-native infrastructure. Leveraging a containerized version of Arista EOS software (CloudEOS-CR), including the same software binary utilized by all Arista networking platforms, each Kubernetes node now has access to the full power of Arista EOS and CloudVision. This combination provides consistent cloud-grade routing and real-time state streaming telemetry to Kubernetes clusters for the first time.



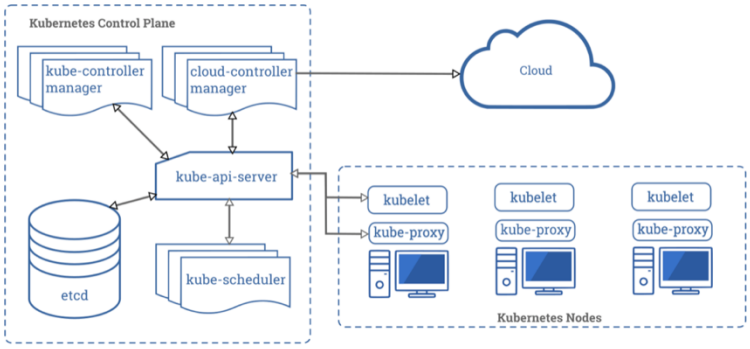
Arista CloudEOS is then deployed as a Kubernetes daemonset, creating a CloudEOS pod on every node in the cluster to serve as the routing engine.



11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS



Doug Gourlay, vice president and general manager for cloud networking software at Arista Networks, says the overarching



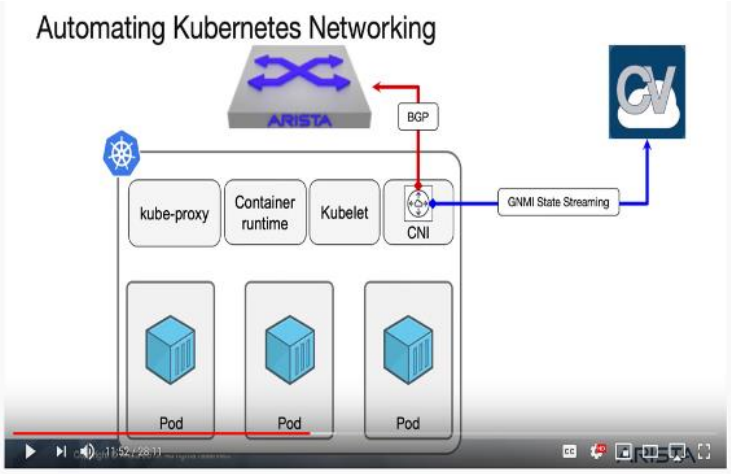
Arista Networks, much like every provider of a network switch, has been making a case for standardizing networking software across on-premises and cloud computing environments. Rather than rely on networking services from cloud service providers that need to be managed in isolation, Arista Networks is making it possible to deploy a common layer of networking software across multiple cloud and on-premises IT environments. Achieving that goal becomes even easier when all the platforms running that networking software expose the same Kubernetes application programming interfaces (APIs), notes Gourlay.

To make that possible, Arista Networks created CloudEOS Cloud Native, an instance of the Arista NOS that can be deployed on either a standalone Kubernetes cluster or on a Kubernetes cluster that supports the Container Network Interface (CNI) defined by the Cloud Native Computing Foundation (CNCF).

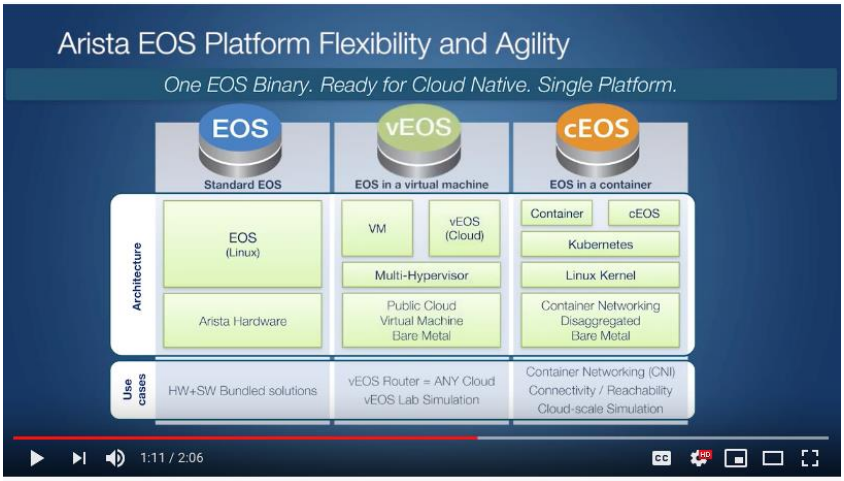
CloudEOS supports fully elastic pay-as-you-go consumption and scaling in the public cloud, with automatic deployment and provisioning based on real-time application demand. In addition it provides the key features required to achieve integration of network operations with the dynamic cloud operational models required to sustain the benefits of a multi cloud infrastructure, while minimizing operating costs.

85. The Accused Products comprise a client agent program configured to run on the client systems and to provide workload processing for at least one project of a distributed computing platform. As one non-limiting example, as depicted in the below screenshots of an

Arista product demonstration, Arista illustrates a client agent program comprised of a kube-proxy, container runtime, and kubelet Kubernetes components, and of cEOS instances, which all are parts of a client system and provide workload processing for a project running on the distributed computing platform (*e.g.*, the Arista CloudEOS Router for Kubernetes platform).



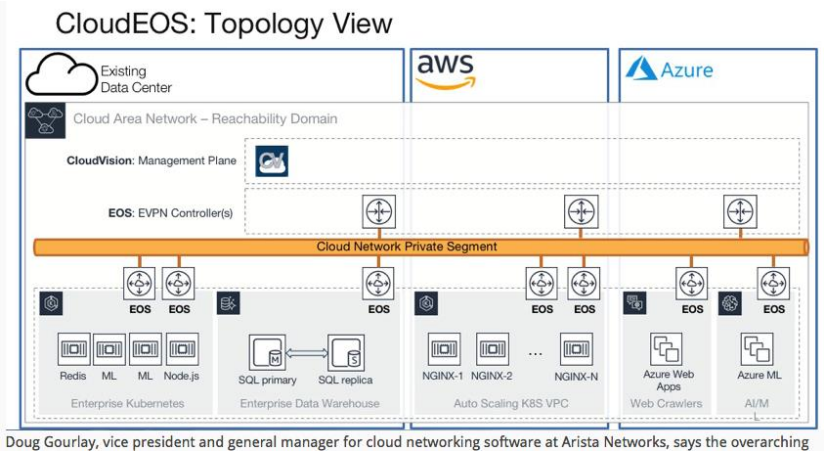
11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS



Arista Any Cloud for Kubernetes

CloudEOS provides two main capabilities:

1. CloudEOS Multi Cloud, a high-performance virtual machine that normalizes the network connectivity to and between public clouds simplifying the networking operating model for cloud and network operations while enabling declarative software-based provisioning through popular DevOps tools
2. CloudEOS Cloud Native, an instance of EOS deployed as a Kubernetes Container Network Interface or stand-alone Kubernetes container to provide a fully supported, enterprise-class networking stack within Cloud Native environments

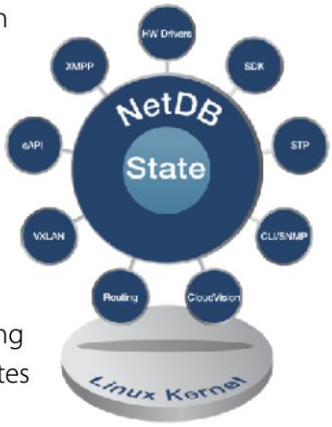


Doug Gourlay, vice president and general manager for cloud networking software at Arista Networks, says the overarching

86. The Accused Products comprise at least one server system configured to communicate with the plurality of client systems through a network. As one non-limiting example, CV as integrated with a Kubernetes controller manages a Kubernetes cluster comprising nodes running cEOS instances and communicates with them through a network.

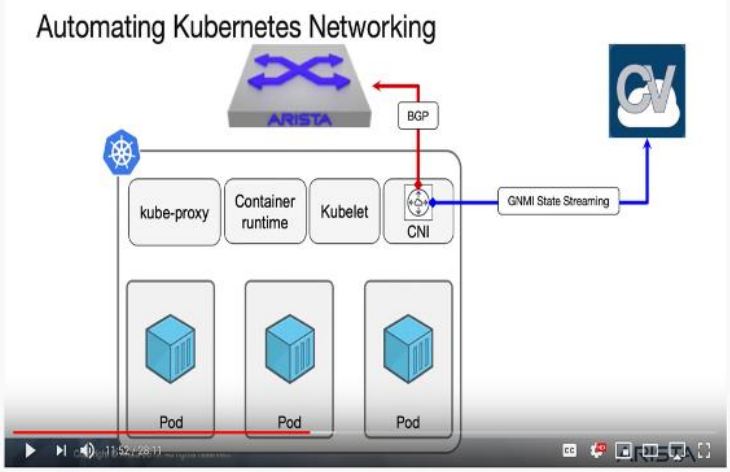
Overview

Arista CloudEOS™ Router for Kubernetes provides an open and scalable solution for customers that are deploying cloud-native infrastructure. Leveraging a containerized version of Arista EOS software (CloudEOS-CR), including the same software binary utilized by all Arista networking platforms, each Kubernetes node now has access to the full power of Arista EOS and CloudVision. This combination provides consistent cloud-grade routing and real-time state streaming telemetry to Kubernetes clusters for the first time.



Provisioning & Monitoring

- CloudVision turnkey automation
- Template-based deployment
- Event monitoring & management
- Onboard packet capture analysis
- Cloud Tracer™ Telemetry



11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS

Arista CloudEOS delivers an open networking platform, with best-in-class networking, enhanced security, and cluster management that preserves architectural choices for customers as they embark on their Kubernetes journey.

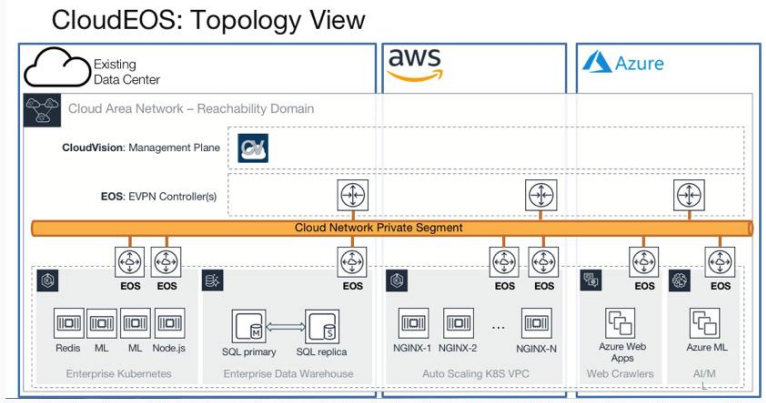
CloudEOS is available on Amazon Web Services and Microsoft Azure immediately and on Google Cloud within the next quarter. CloudVision 2019 is available and shipping today and provides a clustered management plane for the entire network.

Waiting for the Primary Node Installation to Finish

These examples show the system output shown as CVP completes the installation for the primary node.

• Waiting for primary node installation to pause until other nodes send files

```
Running : /bin/sudo /bin/systemctl start cvpi-watchdog.timer
Running : /bin/sudo /bin/systemctl enable docker
Running : /bin/sudo /bin/systemctl start docker
Running : /bin/sudo /bin/systemctl enable kube-cluster.path
Running : /bin/sudo /bin/systemctl start kube-cluster.path
Waiting for all components to start. This may take few minutes.
```



Doug Gourlay, vice president and general manager for cloud networking software at Arista Networks, says the overarching

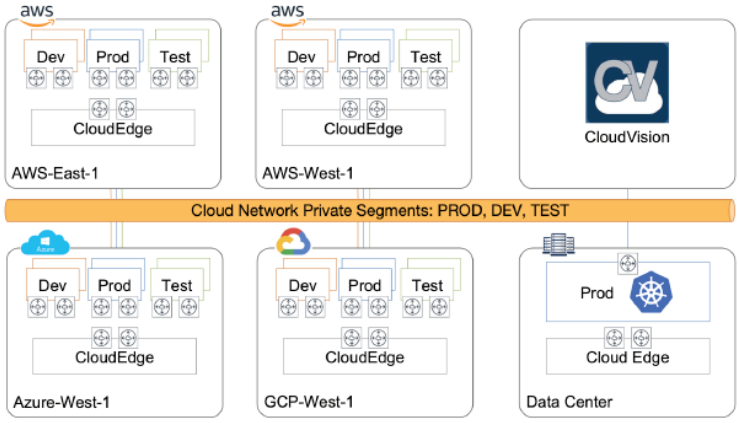
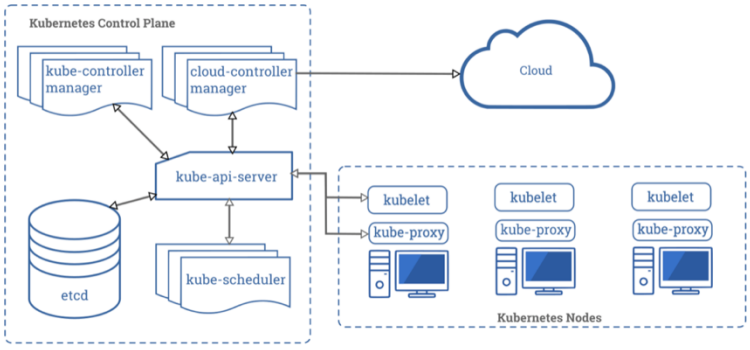


Figure 2: Multi Cloud Networking with Cloud Network Private Segments



kube-controller-manager

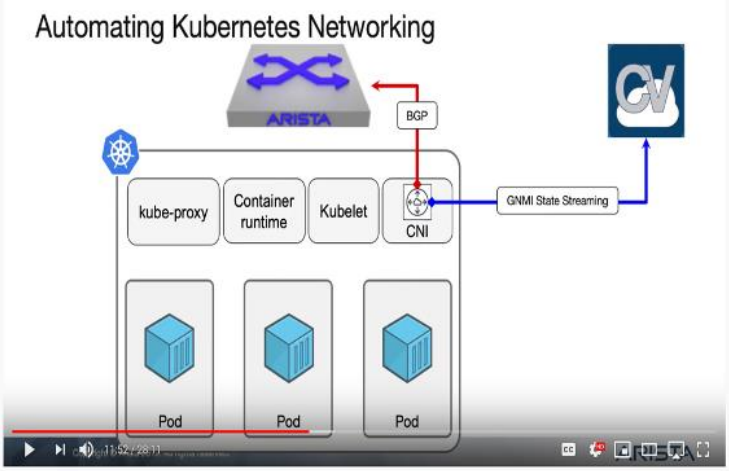
Component on the master that runs controllers.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

These controllers include:

- Node Controller: Responsible for noticing and responding when nodes go down.
- Replication Controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.

87. The Accused Products comprise at least one server system to provide the client agent program to the client systems and to send initial project and poll parameters to the client systems. As one non-limiting example, CV, as integrated with a Kubernetes controller, provides cEOS instances (including kube-proxy, container runtime, and kubelet Kubernetes components) to nodes of a VPC, as well as initial parameters relating to a project (*e.g.*, such as maintaining a requisite number of BGP peering sessions for the VPC) and initial poll parameters such as those relating to the telemetry streaming that occurs between CV and the cEOS instances.



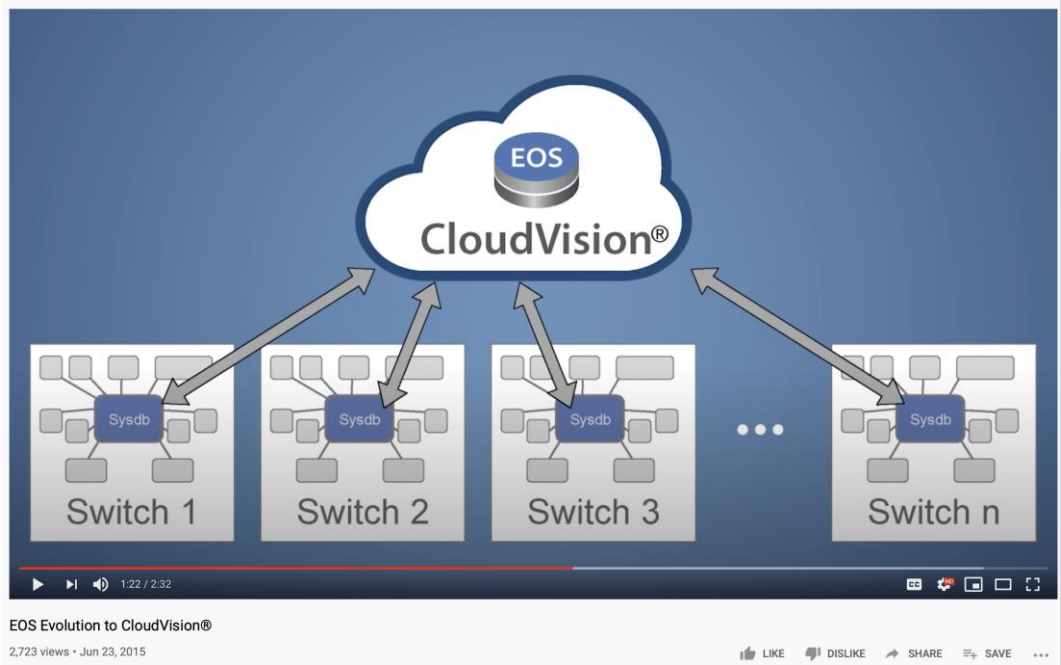
11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS

infrastructures, and bare metal environments. As applications move to cloud-native computing and micro-services, CloudEOS enables hyper-scale customers, service providers, and enterprises to deploy workloads into any location, and support them in any place in the cloud – hence enabling rapid time-to-market, massive scalability and unprecedented agility for any workload. Some of the

Provisioning & Monitoring

- CloudVision turnkey automation
- Template-based deployment
- Event monitoring & management
- Onboard packet capture analysis
- Cloud Tracer™ Telemetry

Declarative Cloud Provisioning - using the popular Terraform application a platform engineering team can stand up virtual private cloud instances in multiple public cloud providers and with a single additional line of code per VPC. Terraform will automatically deploy an Arista CloudEOS Leaf Router, or a high availability pair of routers, and establish secure connectivity to a CNPS for each connected VPC.



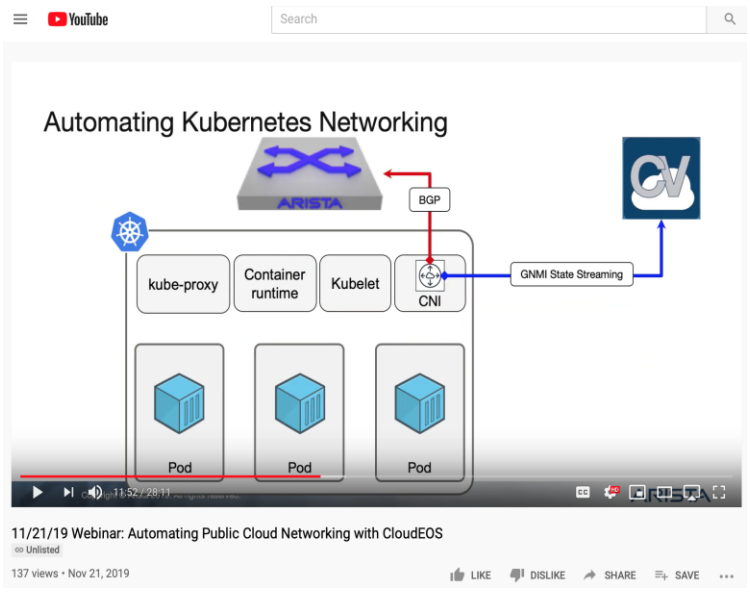
Pod Templates

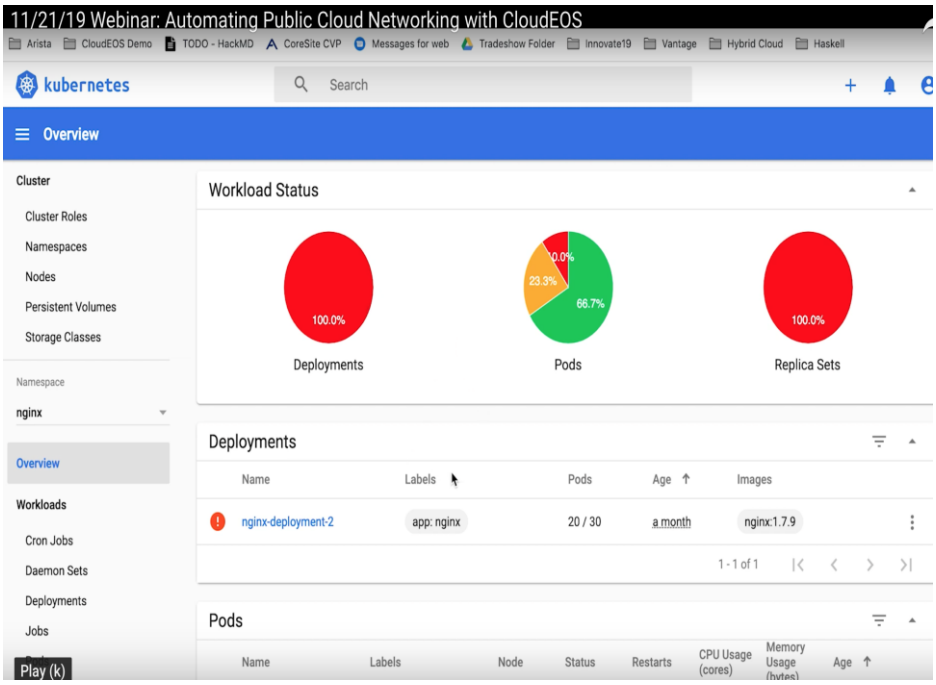
Pod templates are pod specifications which are included in other objects, such as Replication Controllers, Jobs, and DaemonSets. Controllers use Pod Templates to make actual pods. The sample below is a simple manifest for a Pod which contains a container that prints a message.

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

88. The Accused Products comprise at least one server system to receive poll communications from the client systems during processing of the project workloads, wherein a dynamic snapshot information of current project status is provided based at least in part upon the poll communications from the client systems. As one non-limiting example, CV receives state-

based telemetry streaming from cEOS instances, thus collectively providing dynamic snapshot information of project status.

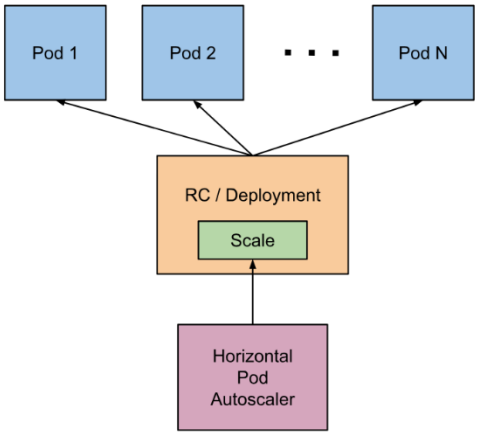




Arista CloudEOS along with CloudVision allows customers to create and manage secure global network topologies spanning multiple regions across multiple clouds and their Kubernetes clusters on-premises. In this solution, Kubernetes provides the underlying cluster lifecycle management, as well as user access control.

After initial deployment, CloudEOS can be configured and monitored using standard EOS CLI commands, Arista's JSON eAPI, OpenConfig, and real-time state streaming telemetry APIs used by Arista CloudVision.

Enhanced Visibility and Analytics - each networking node within the CNPS streams telemetry data back to Arista CloudVision which can be deployed in the public cloud in a VPC or on-premises. This enables a digital replica of the network state to be available for analytics or as training data for supervised machine learning models.



The Horizontal Pod Autoscaler is implemented as a control loop, with a period controlled by the controller manager's `--horizontal-pod-autoscaler-sync-period` flag (with a default value of 15 seconds).

During each period, the controller manager queries the resource utilization against the metrics specified in each HorizontalPodAutoscaler definition. The controller manager obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API (for all other metrics).

- For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each pod targeted by the HorizontalPodAutoscaler. Then, if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. If a target raw value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value (depending on the type of target specified) across all targeted pods, and produces a ratio used to scale the number of desired replicas.

89. The Accused Products comprise at least one server system to analyze the poll communications utilizing the dynamic snapshot information to determine whether to change how many client systems are active in the at least one project. As one non-limiting example, CV, as integrated with a Kubernetes controller, analyzes telemetry information from the cEOS instances utilizing dynamic snapshot information about project status to determine the number of cEOS instances that are active in the project.

After initial deployment, CloudEOS can be configured and monitored using standard EOS CLI commands, Arista's JSON eAPI, OpenConfig, and real-time state streaming telemetry APIs used by Arista CloudVision.

Enhanced Visibility and Analytics - each networking node within the CNPS streams telemetry data back to Arista CloudVision which can be deployed in the public cloud in a VPC or on-premises. This enables a digital replica of the network state to be available for analytics or as training data for supervised machine learning models.

CloudEOS supports fully elastic pay-as-you-go consumption and scaling in the public cloud, with automatic deployment and provisioning based on real-time application demand. In addition it provides the key features required to achieve integration of network operations with the dynamic cloud operational models required to sustain the benefits of a multi cloud infrastructure, while minimizing operating costs.

Arista CloudEOS along with CloudVision allows customers to create and manage secure global network topologies spanning multiple regions across multiple clouds and their Kubernetes clusters on-premises. In this solution, Kubernetes provides the underlying cluster lifecycle management, as well as user access control.

After initial deployment, CloudEOS can be configured and monitored using standard EOS CLI commands, Arista's JSON eAPI, OpenConfig, and real-time state streaming telemetry APIs used by Arista CloudVision.

For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each pod targeted by the HorizontalPodAutoscaler. Then, if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each pod. If a target raw value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value (depending on the type of target specified) across all targeted pods, and produces a ratio used to scale the number of desired replicas.

90. If a fewer number of client systems are desired by the distributed computing platform, the server will include within a poll response communications a reduction in the number of client systems actively participating in the project, and if a greater number is desired, the server will add client systems to active participation in the at least one project within a poll response communications. As one non-limiting example, CV, as integrated with a Kubernetes controller, supports automatic deployment and provisioning of cEOS instances based on information received through poll communication that impacts project status, such as data indicative of changes in real-time application demand being handled by the cluster / VPC serviced by the Accused Products.

CloudEOS supports fully elastic pay-as-you-go consumption and scaling in the public cloud, with automatic deployment and provisioning based on real-time application demand. In addition it provides the key features required to achieve integration of network operations with the dynamic cloud operational models required to sustain the benefits of a multi cloud infrastructure, while minimizing operating costs.

Declarative Cloud Provisioning - using the popular Terraform application a platform engineering team can stand up virtual private cloud instances in multiple public cloud providers and with a single additional line of code per VPC. Terraform will automatically deploy an Arista CloudEOS Leaf Router, or a high availability pair of routers, and establish secure connectivity to a CNPS for each connected VPC.

By using standard network protocols such as BGP, and best in class data center networking switches, Arista and Tigera provide a high performance networking solution for Kubernetes clusters. This solution provides operational simplicity and visibility to networking teams deploying Kubernetes in the data center, while supporting features such as Kubernetes network policy. With Arista EOS and Tigera Calico, customers can deploy scalable Kubernetes clusters without sacrificing features or performance.

Provisioning & Monitoring

- CloudVision turnkey automation
- Template-based deployment
- Event monitoring & management
- Onboard packet capture analysis
- Cloud Tracer™ Telemetry

infrastructures, and bare metal environments. As applications move to cloud-native computing and micro-services, CloudEOS enables hyper-scale customers, service providers, and enterprises to deploy workloads into any location, and support them in any place in the cloud – hence enabling rapid time-to-market, massive scalability and unprecedented agility for any workload. Some of the

> If there were any missing metrics, we recompute the average more conservatively, assuming those pods were consuming 100% of the desired value in case of a scale down, and 0% in case of a scale up. This dampens the magnitude of any potential scale.
>
> Furthermore, if any not-yet-ready pods were present, and we would have scaled up without factoring in missing metrics or not-yet-ready pods, we conservatively assume the not-yet-ready pods are consuming 0% of the desired metric, further dampening the magnitude of a scale up.

Finally, just before HPA scales the target, the scale recommendation is recorded. The controller considers all recommendations within a configurable window choosing the highest recommendation from within that window. This value can be configured using the `--horizontal-pod-autoscaler-downscale-stabilization` flag, which defaults to 5 minutes. This means that scaledowns will occur gradually, smoothing out the impact of rapidly fluctuating metric values.

From the most basic perspective, the Horizontal Pod Autoscaler controller operates on the ratio between desired metric value and current metric value:

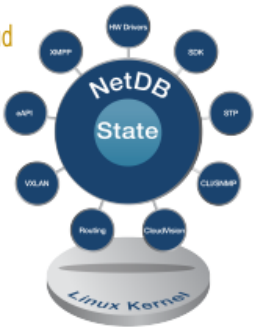
```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue
```

For example, if the current metric value is `200m`, and the desired value is `100m`, the number of replicas will be doubled, since `200.0 / 100.0 == 2.0` If the current value is instead `50m`, we'll halve the number of replicas, since `50.0 / 100.0 == 0.5`. We'll skip scaling if the ratio is sufficiently close to 1.0 (within a globally-configurable tolerance, from the `--horizontal-pod-autoscaler-tolerance` flag, which defaults to 0.1).

91. The Accused Products comprise at least one server system to repeatedly utilize the poll communication and the poll response communications to coordinate project activities of the

client systems during project operations. As one non-limiting example, CV receives real-time state-based telemetry data streamed from the cEOS instances to coordinate project activity of the cEOS instances as those instances are operating on projects, such as maintaining the requisite set of border gateway protocol BGP peering sessions for the VPC they are serving.

CloudEOS™ is Arista's Multi Cloud and Cloud Native networking solution, supporting autonomic operation, and delivering an enterprise-class, highly secure and reliable networking experience for the cloud.



Autonomic Cloud Operations

Elastic: Pay-As-You-Go (PAYG) consumption through Public Cloud Marketplaces and Service Catalogs

All Inclusive: includes CloudVision with Terraform integrations, all EOS routing and telemetry features, and A-care software support

Consistent operational model: Arista EOS® and CloudVision® complemented with DevOps and CloudOps toolchain integrations like Terraform

To provide a scalable and automated network experience, CloudEOS fully integrates its automated operational features with Arista CloudVision® to simplify the operators experience of interconnecting and managing multi-cloud, cloud-native and on-premises enterprise networks. Leveraging a network-wide approach for workload orchestration and workflow automation, together with advanced network telemetry and standards-based programmability, CloudEOS and CloudVision provide a seamless and universal approach to multi-cloud networking.

Arista CloudEOS Router - Summary of Key Capabilities

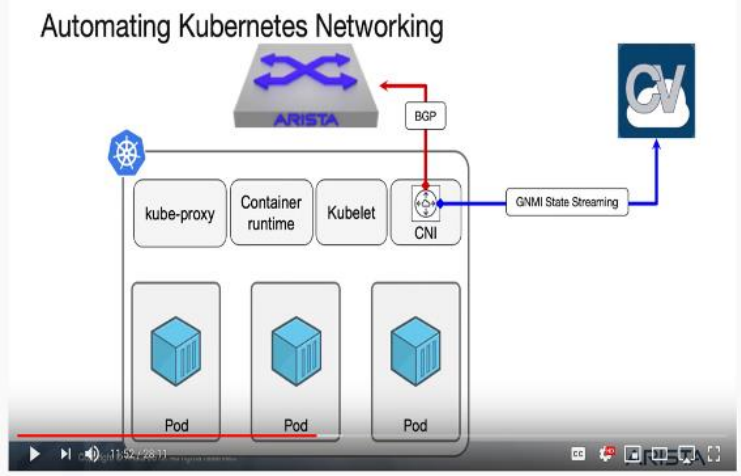
Declarative Cloud Provisioning - using the popular Terraform application a platform engineering team can stand up virtual private cloud instances in multiple public cloud providers and with a single additional line of code per VPC. Terraform will automatically deploy an Arista CloudEOS leaf router, or a high availability pair of routers, and establish secure connectivity to a CNPS for each connected VPC.

The recommended BGP peering configuration is an iBGP session between CloudEOS on the Kubernetes nodes and an adjacent top-of-rack (ToR) switch, which is acting as a route reflector. Once BGP peering has been established, Kubernetes POD networks are fully routed through the enterprise network without the need for any proprietary encapsulation or out-of-band management.

Arista CloudEOS is then deployed as a Kubernetes daemonset, creating a CloudEOS pod on every node in the cluster to serve as the routing engine.

CloudEOS supports fully elastic pay-as-you-go consumption and scaling in the public cloud, with automatic deployment and provisioning based on real-time application demand. In addition it provides the key features required to achieve integration of network operations with the dynamic cloud operational models required to sustain the benefits of a multi cloud infrastructure, while minimizing operating costs.

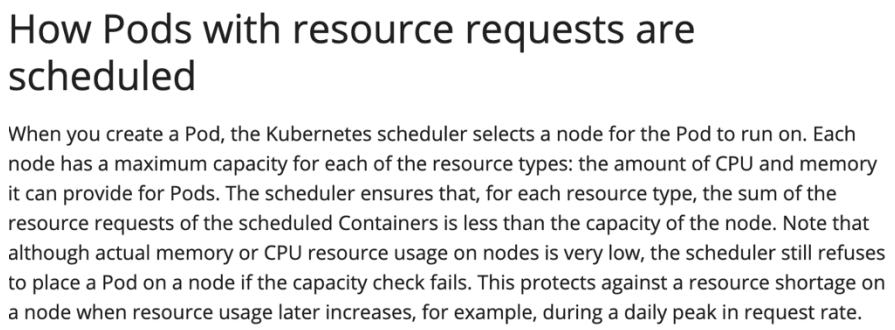
By using standard network protocols such as BGP, and best in class data center networking switches, Arista and Tigera provide a high performance networking solution for Kubernetes clusters. This solution provides operational simplicity and visibility to networking teams deploying Kubernetes in the data center, while supporting features such as Kubernetes network policy. With Arista EOS and Tigera Calico, customers can deploy scalable Kubernetes clusters without sacrificing features or performance.



11/21/19 Webinar: Automating Public Cloud Networking with CloudEOS

The Horizontal Pod Autoscaler is implemented as a control loop, with a period controlled by the controller manager's `--horizontal-pod-autoscaler-sync-period` flag (with a default value of 15 seconds).

During each period, the controller manager queries the resource utilization against the metrics specified in each HorizontalPodAutoscaler definition. The controller manager obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API (for all other metrics).



How Pods with resource requests are scheduled

When you create a Pod, the Kubernetes scheduler selects a node for the Pod to run on. Each node has a maximum capacity for each of the resource types: the amount of CPU and memory it can provide for Pods. The scheduler ensures that, for each resource type, the sum of the resource requests of the scheduled Containers is less than the capacity of the node. Note that although actual memory or CPU resource usage on nodes is very low, the scheduler still refuses to place a Pod on a node if the capacity check fails. This protects against a resource shortage on a node when resource usage later increases, for example, during a daily peak in request rate.

92. Additionally, Arista has been, and currently is, an active inducer of infringement of the ’153 patent under 35 U.S.C. § 271(b) and contributory infringement of the ’153 patent under 35 U.S.C. § 271(c) either literally and/or by the doctrine of equivalents.

93. Arista has actively induced, and continues to actively induce, infringement of the ’153 patent by intending that others use, offer for sale, or sell in the United States, products and/or services covered by one or more claims of the ’153 patent, including but not limited to the Accused Products. Arista provides these products and/or services to others, such as customers, resellers and end-user customers, who, in turn, use, provision for use, offer for sale, or sell in the United States products and/or services that directly infringe one or more claims of the ’153 patent.

94. Arista has contributed to, and continues to contribute to, the infringement of the ’153 patent by others by knowingly providing products and/or services that, when installed and configured result in a system as intended by Arista, directly infringe one or more claims of the ’153 patent.

95. Arista knew of the ’153 patent, or should have known of the ’153 patent, but was willfully blind to its existence. Upon information and belief, Arista has had actual knowledge of the ’153 patent since at least August 17, 2020 and/or as early as the service upon Arista of this Complaint. By the time of trial, Arista will have known and intended (since receiving such notice)

that its continued actions would infringe and actively induce and contribute to the infringement of one or more claims of the ’153 patent.

96. Arista has committed, and continues to commit, affirmative acts that cause infringement of one or more claims of the ’153 patent with knowledge of the ’153 patent and knowledge or willful blindness that the induced acts constitute infringement of one or more claims of the ’153 patent. As an illustrative example only, Arista induces such acts of infringement by its affirmative actions of intentionally providing hardware and/or software components that, when used in their normal and customary way as desired and intended by Arista, infringe one or more claims of the ’153 patent and/or by directly or indirectly providing instructions on how to use its products and services in a manner or configuration that infringes one or more claims of the ’153 patent, including those found at one or more of the following:

- https://youtu.be/fP75osrRj7o
- https://www.youtube.com/watch?v=8cFaevNlz40
- https://www.arista.com/en/company/news/press-release/8760-pr-20191105
- https://www.arista.com/assets/data/pdf/Whitepapers/Arista_CloudEOS_Product_Brief.pdf
- https://www.arista.com/assets/data/pdf/user-manual/um-books/CV_Config_Guide_2020.1.pdf
- https://www.arista.com/assets/data/pdf/Datasheets/CloudEOS-Router-At-a-Glance.pdf
- https://www.arista.com/assets/data/pdf/Datasheets/EOSCloudVision_DataSheet.pdf

97. Arista has also committed, and continues to commit, contributory infringement by, *inter alia*, knowingly selling products and/or services that when used cause the direct infringement of one or more claims of the ’153 patent by a third party, and which have no substantial non-infringing uses, or include a separate and distinct component that is especially made or especially adapted for use in infringement of the ’153 patent and is not a staple article or commodity of commerce suitable for substantial non-infringing use.

98. As a result of Arista's acts of infringement, Plaintiffs have suffered and will continue to suffer damages in an amount to be proved at trial.

PRAYER FOR RELIEF

Plaintiffs request that the Court enter judgment against Arista as follows:

(A) finding that Arista has infringed one or more claims of each of the above patents-in-suit, directly and/or indirectly, literally and/or under the doctrine of equivalents;

(B) awarding damages sufficient to compensate Plaintiffs for Arista's infringement under 35 U.S.C. § 284;

(C) finding this case exceptional under 35 U.S.C. § 285 and awarding Plaintiffs their reasonable attorneys' fees;

(D) awarding Plaintiffs their costs and expenses incurred in this action;

(E) awarding Plaintiffs prejudgment and post-judgment interest; and

(F) granting Plaintiffs such further relief as the Court deems just and appropriate.

DEMAND FOR JURY TRIAL

Plaintiffs demand trial by jury of all claims so triable under Federal Rule of Civil Procedure 38.

Date: August 18, 2020

Respectfully submitted,

/s/ Derek Gilliland
DEREK GILLILAND
STATE BAR NO. 24007239
SOREY, GILLILAND & HULL, LLP
P.O. BOX 4203
109 W. TYLER ST.
LONGVIEW, TEXAS 75601
903.212.7362 (TELEPHONE)
DEREK@SOREYLAW.COM

OF COUNSEL:

Paul J. Hayes
phayes@princelobel.com
Matthew Vella
mvella@princelobel.com
Robert R. Gilman
rgilman@princelobel.com
Jonathan DeBlois
jdeblois@princelobel.com
PRINCE LOBEL TYE LLP
One International Place, Suite 3700
Boston, MA 02110
Tel: (617) 456-8000
Fax: (617) 456-8100

COUNSEL for PLAINTIFFS