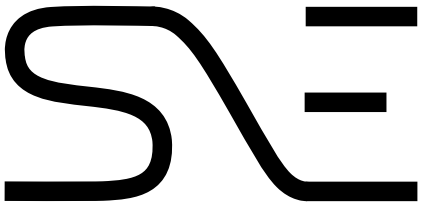

# Node-to-node secure multiparty computation on Vantage6

Renee Witsenburg
rwitsenburg@os3.nl

**Abstract**—This study investigated the possibilities to create node to node communication between Vantage6 algorithm containers without the interposition of the Vantage6 server. Three infrastructures are created: an overlay network in swarm mode, an overlay network with an external key-value store, and a VPN network. The performance of these setups are tested on throughput with iperf3 and on the duration to perform a secure sum algorithm script between three nodes. Throughput is higher on the overlay networks and running a secure sum algorithm on the overlay network is faster than running this algorithm in the VPN setup. A swarm and an overlay network are manually created on the Vantage6 server virtual machine. The docker-manager Python script of the Vantage6 node is changed to join the swarm and connect the algorithm containers to the overlay network. To achieve this new feature a workaround is needed, since the Docker API does not support overlay networks.

**Index Terms**—Vantage6, Personal Health Train, Federated Learning, Differential Privacy, Secure Multiparty Computation, Overlay networking, Docker VPN networking

## 1 Introduction

Researchers use patient data to develop and improve treatments for diseases. Nowadays, people collect a lot of personal health data with apps and smart devices. This data can be shared with someone's personal doctor and with a research institute. With patient data it is possible to create detailed treatment plans. Because patient data is very privacy-sensitive the 'Personal Health Train´ principle has been developed [4]. Patient data, enriched with personal health data, is stored in so-called data-stations. Trains or algorithms are sent to the data-stations. The data owners have access to their own data and can set access permissions that determine who may see their data and how it may be used. Larger data stations can be organised with these sets of rules. Each station runs the algorithm on its own data set to generate aggregated statistics. This makes it possible to collaborate by exchanging aggregated data/statistics without sharing the actual data.

Sites working together are great for federated learning and improving algorithms or treatments automatically. There are a few frameworks and protocols available which make federated learning on privacy-sensitive data possible. WebDISCO [14] is a web service for distributed Cox model learning. A user can create an account and assign tasks to a global server. If all the requirements are met, the task will be executed. These tasks are running on light-weight clients. The OpenMined community has created PySyft [19] as a library for secure and private Deep Learning. This library decouples private data from model training, which makes it possible to let one's code run on a remote site. However, it is only possible to use PySyft in a collaboration between multiple sites when one is using the PyGrid [7] infrastructure, which is in development.

The Vantage6 framework (priVAcy preserviNg federaTed leArninG infrastructurE) [6] is designed with a modular architecture in mind. Vantage6 is an open-source Python framework for federated learning based on the 'Personal Health Train´ principle. With the Vantage6 implementation a researcher sends a task to a central server, which directs the task to the appropriate nodes. This central server coordinates communication between the researchers and the nodes. The nodes contain the algorithm and perform the heavy lifting of the framework. The nodes have access to the data storage on site at the organisation.

This study is about the possibilities to improve the performance within the Vantage6 infrastructure. Collaboration between the nodes is managed by the central server. In some cases this is not the most efficient way to build the infrastructure, because in some cases this generates unnecessary network traffic. During this study multiple proof of concept infrastructures will be set up and their performance will be

measured.

Section two of this paper contains the research questions of the study. Section three presents related work on topics like federated learning, differential privacy and secure multiparty computation and the architectures that can be used. Section four outlines the methods used during this study, such as the infrastructure used and the way the measurements are performed will be described in more detail here. In section five the results of the measurements are presented. This study will end with a discussion and conclusion.

## 2 Research questions

Carrier is a study to detect and prevent coronary artery disease (CAD) early. Clinicians, legal experts, data scientists, and other stakeholders collaborate on research on big data, such as socio-economic big data and personal health data. The study combines big data and artificial intelligence to build models to do detection and intervention to prevent CAD on an individual level [2]. Carrier uses Vantage6 to embrace the core concepts of the personal health train. Regional alliances can set up Vantage6 nodes to use their data, without sharing the data. However, current experiences show the limitations of the Vantage6 framework for the Carrier project.

In this study different infrastructures for the Vantage6 framework are tested on performance. To come up with efficient infrastructures to do secure multiparty computation, the theories and implementations of PyGrid will be investigated in order to use PyGrid inside the Vantage6 framework.

The research question in this project will be:

*How can Vantage6 edge nodes work together efficiently, without the interposition of the central server?*

In order to answer the main research question, the following sub-questions are defined:

- What are the issues in the current Vantage6 project that create a bottleneck between working nodes?
- Which infrastructures could be implemented to make the nodes work together?
- Is it possible to tamper with, read or intercept data or the model?
- How are malicious attempts to corrupt the data or the model detected?

## 3 Related work

Previously, when a data scientist wanted to answer research questions about multiple datasets, copies of the datasets where generated from the source and transported to a trusted party. This party then became responsible for the storing, merging and analyzing of the combined data. In figure 1 this traditional central approach is visualized. Nowadays, loads and loads of data is generated in databases, (mobile) systems, IoT devices, etc. Therefore, other methods of combining and processing data needs to be designed.

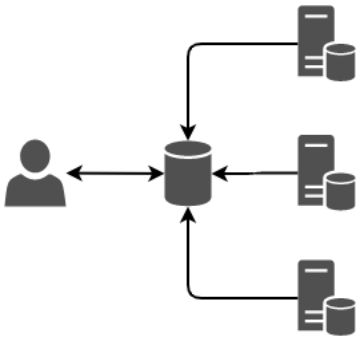

Fig. 1: Traditional Central Approach on data research

One such method is federated learning, which has become widely adopted. For example, keyboard autocompletion models are trained locally on your mobile device. Data that cannot be seen is used to train an artificial intelligence. To use these models and infrastructures in healthcare, developers need to comply with the law to protect this data. Patient data needs to comply with strict rules before it can be used for research. The Health Insurance Portability and Accountability Act (HIPAA) Safe Harbor [3] in America defines explicit rules for the de-indentification of healthcare data. The Data Protection Act in the Netherlands requires that clinical patient data is physically located in member states of the European Union.

Research has previously been done on how to de-indentify data and share it between parties. In the subsections WebDISCO, PySyft and PyGrid, and Vantage6, different secure multiparty computation infrastructures will be looked into and discussed.

### 3.1 WebDISCO

In 2015 Lu et al. published a paper about their developed product, WebDISCO, a web service for distributed cox model learning without patient-level data sharing [14]. They developed the product to overcome the disadvantages of the traditional central approach of doing research on multiple datasets, such as the concerns of individual privacy, institutional policies, practical considerations on data transportation and so on. In their research they focused on the differences between computation of the Cox Model in a traditional central approach versus the Cox Model in a distributed approach.

The implementation of the infrastructure to use WebDISCO can be seen in figure 2. Users can register at and log in into the web portal. The user initializes a task and sets the parameters, such as the number of iterations and the location of the input data. The task creator and the task participants are notified when the task is created. When the task is triggered, the global server starts interacting with the clients on the participants' workstations. HTML5 and a signed communication between Java Servlets and Applets is used to create this infrastructure. The computations are done locally, since the data must not leave the site. The heavy lifting of the computations is performed by lightweight clients. This conflicts with the heavy-server light-client architecture and limits the duration of the iteration. The authors of the WebDISCO study conclude that the

proof-of-concept implementation has met technical barriers, but that these can be overcome.

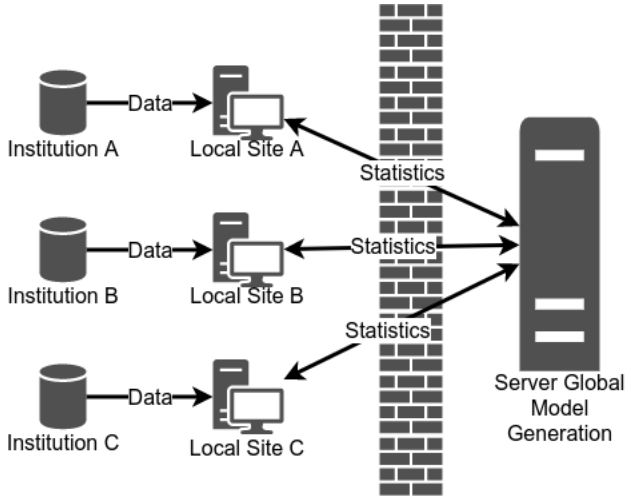


Fig. 2: WebDISCO infrastructure

### 3.2 PySyft and PyGrid

Ryffel et al. wrote a paper in 2018 about their new library for privacy preserving deep learning [19], which they later called PySyft. PySyft combines federated learning, secure multiparty computation and differential privacy. When using multiple datasets, federated learning is vulnerable to reverse engineering attacks. It makes it possible to extract data from the model. In PySyft, the researcher can add differential privacy to protect the data before it enters the model, so the reverse engineering attack doesn't harm the data.

PySyft makes it possible "to work with data you cannot see". The library or ecosystem focuses on consistent object serialization and deserialization, core abstractions, and algorithm execution. The connection between the data scientist and data owner are peer-to-peer, as seen in figure 3, based on WebRTC. The library alone will not connect to external data.

The developers of PySyft admit that the framework had some issues. In the future, they want to improve and add some features, such as decreasing training time and making the multiparty computation function detect malicious attempts to corrupt the data or the model.


Fig. 3: PySyft Duet

OpenMined is the community working on PySyft. Their developers created PyGrid to create an infrastructure. PyGrid aims to be a peer-to-peer platform that uses PySyft. This architecture is composed of gateways and nodes. Nodes contain private data clusters provided by the data owner. Data owners connect the node to the infrastructure to make it available to the public. Access to the nodes can also be managed by the data owner, as well as a brief description, tags and labels to help the data scientist find the dataset. The gateway performs the queries over the network. It works like an interface between the data scientist and the grid network, without having access to the data. A data scientist can connect to the network and search for the right datasets. It creates pointers to the data and does the computations [7].

### 3.3 Vantage6

Vantage6 is another new infrastructure, which is described in an article in 2020 by Moncada-Torres et al. [17]. This framework was designed to overcome the limitations of other open source frameworks. DataSHIELD only lets researchers use R and the pre-defined library of functions and algorithms [1]. Facebook's CrypTen depends on PyTorch and does not work on all operating systems [12]. PySyft only works with horizontal-partitioned data, where parties have the same features of different patients.

This framework is no longer dependent on an operating system or programming language, since its flexible setup on the Docker platform can be used with horizontal and vertical partitioned data, and it interacts with the researcher via HTTP requests. It is based on a client-server model with a general architecture that can be found in figure 4. Collaborations between organisations and data scientists are agreed on beforehand. The requested algorithm is delivered as a Docker image from the registry on the Vantage server to the node(s). The nodes run the algorithm container to do the computation and the solution is transmitted via the server to the researcher.

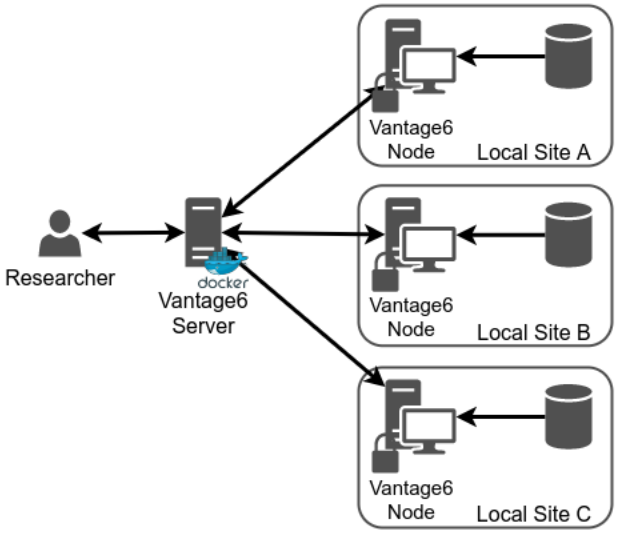


Fig. 4: General Infrastructure of Vantage6

The framework has already been successfully used for cancer research, but the developers are still working on expanding the set of tools for federated learning analysis.

### 3.4 Node to node communication

In some cases it is necessary to have node to node interaction for processing algorithms. Containers on the same Docker host machine can talk to each other by using the docker0 bridge. In the Vantage6 framework it is more likely that algorithm containers are located on different hosts. When

Vantage6 nodes need to work together, the traffic always goes through the Vantage6 server, because only the server knows which nodes are connected in the infrastructure. But this could be undesirable. The Vantage6 server can trace back the data to the node. Even when encryption is turned on, the Vantage6 server needs to decrypt and encrypt the data between nodes [15]. To create a node to node connection inside the Vantage6 framework without the interposition of the Vantage6 server a solution must be found. To make it possible to let the nodes know that they have neighbors in the collaboration, PySyft makes use of peer to peer communication. Peer to peer connections inside Docker can be created by creating an overlay network. This is a VXLAN network that can make Docker containers on multiple hosts connect directly to each other.

Hermans and de Niet did performance research on Docker overlay networks in high-latency environments in 2016 [13]. They did a performance analysis on the default Docker overlay network and three third party overlay drivers, Weave, Flanned and Calico, in the GEANT Testbeds Service. These experiments are specific to GTS, but give an impression of the performance of an overlay network inside Docker. Herman and de Niet hypotized that overlay solutions would perform worse than underlay solutions, since overlay solutions introduce performance overhead by adding a layer to the network. However, since these solutions are running in-kernel, this decrease would not be very large. In their results they found much irregular behaviour while measuring UDP and TCP throughput with iperf, without an identifiable cause. Nevertheless, they claim that the performance decrease in overlay networks are negligible.

Zismer repeated the research of Hermans and de Niet to discover why the UDP throughput of the Docker overlay network is much lower that the TCP throughput [21]. He focused on the correlation between throughput and CPU usage. Previous work of Claassen [9] and Hermans and de Niet hypotized that the strange results of the throughput had to do with segmentation and fragmentation of the packets. Iperf3 creates such a high number of CPU cycles that this affects the throughput of both UDP and TCP.

In 2017 Brouwers looked into the security of Docker swarm networking [8], which by default is the mode to create overlay networks inside the Docker infrastructure. He discusses the default security measures in the Docker infrastructure and performs multiple attacks to see what happens when a container becomes compromised. In certain cases it was possible to inject packets in the VXLAN tunnel, but the real application is questionable. Furthermore, Brouwers managed to do replay attacks on non-encrypted and encrypted packets. Attacking the gossip traffic of the swarm managers was not investigated. With the security measures on the swarm traffic, the impact of these attacks could be low.

## 4 Methods

To get a better understanding of the Vantage6 federated learning framework, a proof of concept has been set up, which represent a collaboration between three organisations. For this study a Dell PowerEdge R230 server is used, with Ubuntu 18.04 LTS installed. Four Xen virtual machines are created to act as a global Vantage6 infrastructure. One of the virtual machines is assigned the role of Vantage6 server and the other three are assigned the roles of the Vantage6 nodes. In figure 5 this setup is visualized.

The virtual machines are created with Ubuntu 18.04 LTS, running with 1024 MB memory and 15 G of storage. They comply with the Vantage6 requirements: Python 3.6+, Docker Community Edition, a stable internet connection and access to the data. The Vantage6 server could also be used as a private registry for the Docker images created. The nodes have access to the database stored on the virtual machine. For the first test the publicly available Diabetes Data Set [18] is used in combination with test algorithms from the Carrier project.

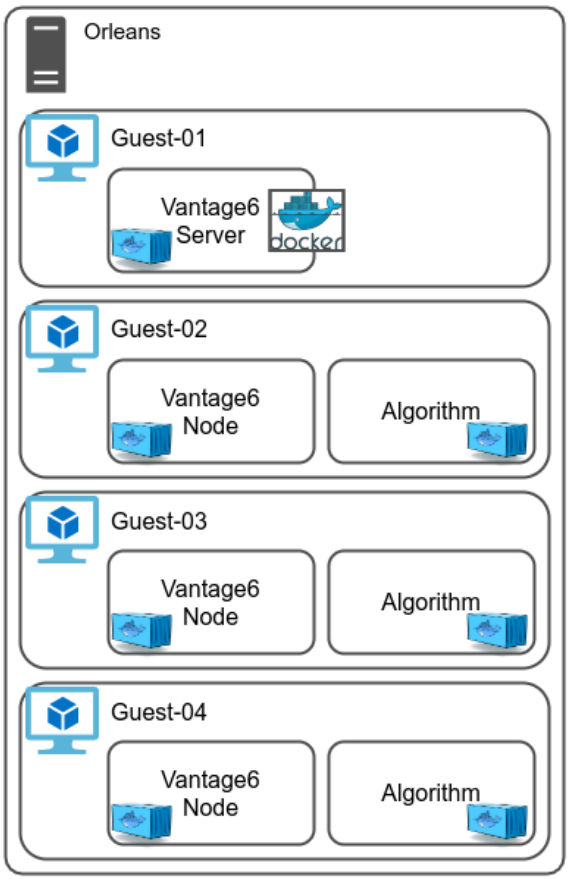


Fig. 5: Vantage6 Infrastructure Setup in the Proof of Concept

First, the setup is tested on whether each node can perform a task individually, outside of the collaboration with the other nodes. Next, a collaboration is registered. The same vantage6-carrier-algorithm [20] is ran within the collaboration. With tcpdump the traffic between the nodes and the server can be analysed. Analysing the traffic must determine what the bottleneck is in the node to node communication.

### 4.1 Node to node communication

To create the node to node communication feature in Vantage6, three infrastructure designs are tested. Since the Vantage6 framework is running inside docker hosts and docker containers, these network architectures must connect the

algorithm containers. The choice was made to test the built in overlay network in docker and to create n VPN network to connect the nodes directly to each other.

One way to make containers on different hosts talk to each other is to use an overlay network. Normally, these containers are hidden from each other, but the overlay network will help to create a subnet at the top of these hosts. Each container connected to this overlay network will be able to communicate with the other containers. There will be two variants of the overlay network, as you can see in figure 6. The first variant will be created using Docker swarm mode. The Vantage6 server virtual machine will initiate the swarm and will be the swarm manager. On this machine the overlay network will be created with encryption enabled. The node machines join the swarm. The algorithm containers on the node machines attach to the overlay network. The second variant works a little bit different. In this case, the swarm mode would not be used, but a separate key value store will be used. On the Vantage6 server machine the progrium/consul docker image will be used to create the key value store container. This key value store makes it possible to keep track of the participation hosts on the overlay network, as well as the general network configuration.

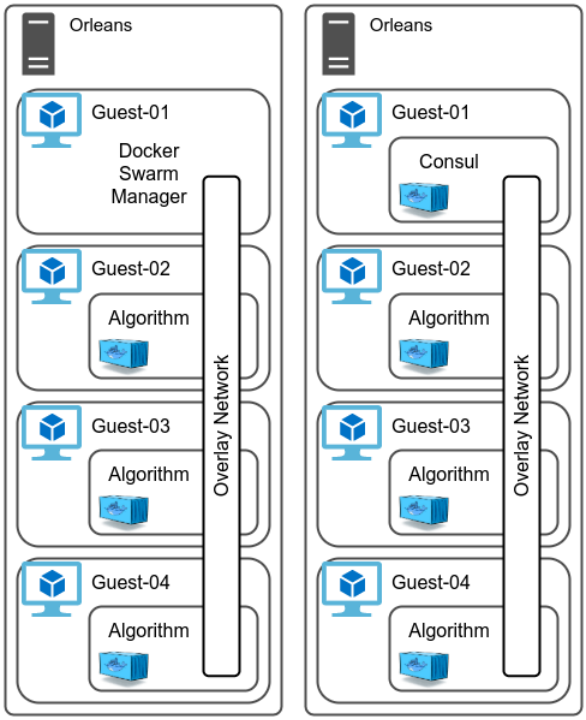


Fig. 6: Algorithm containers connected by an overlay network in swarm mode or with an external key-value store

In the VPN scenario, an OpenVPN server is installed on the Vantage6 server virtual machine. On the other VMs, OpenVPN client containers are created with the dperson/openvpn-client docker image. The algorithm containers are directly connected to the OpenVPN client containers to make their traffic flow through the VPN network, as displayed in figure 7. To test the performance of this setup, iperf3 is used. A container created with the networkstatic/iperf3 docker image replaces the algorithm container but is also directly connected with OpenVPN client container.

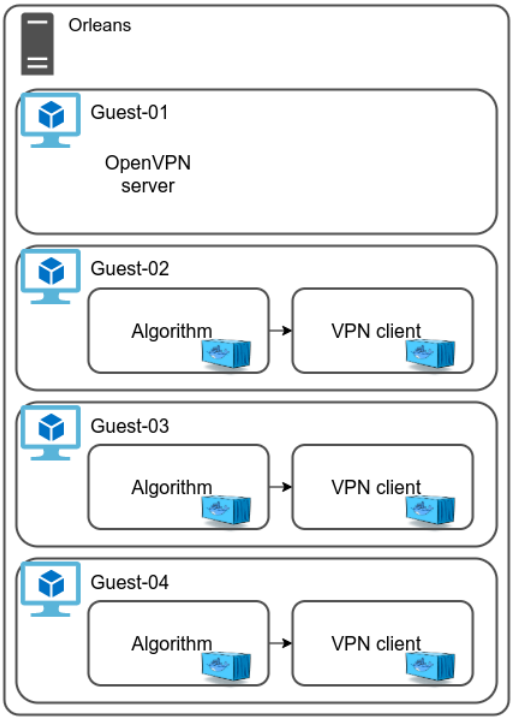


Fig. 7: Algorithm containers connected by VPN-client containers

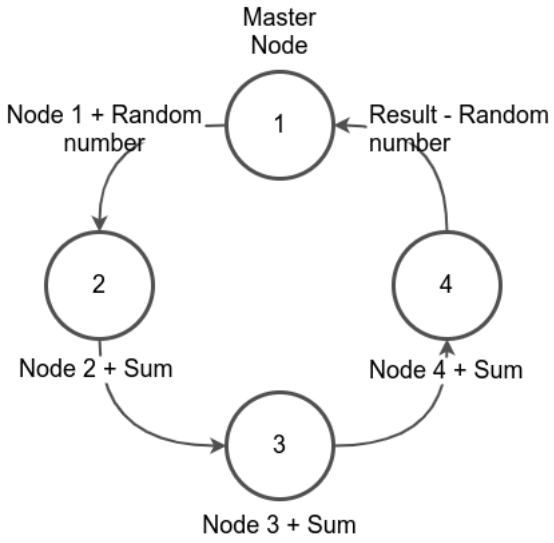


Fig. 8: Secure sum

As mentioned earlier, the performance of the different setups is tested on throughput with iperf3 in the networkstatic/iperf3 containers. The setups are tested with a real world example where the nodes need to collaborate on a simple algorithm too. The algorithm used is called secure sum [10]. This algorithm is chosen because it shows what is wrong with the current Vantage6 architecture. On the

Vantage6 website, a blog post has been published about the problem [15]. Secure sum is one of the basic building blocks of secure multi-party computation. The first node generates a random number, R, between 0 and n. Node one adds this to local value v and sends this result to the second node. Node two does not learn anything about the actual value of node one. Node two adds its own local value and sends it to node three. The formula looks as follows:

$$V = R + \sum_{j=1}^{1-l} v_j$$

In the last step, when the result comes back to the first node, this node extracts the random value R to obtain the actual result. The whole process is visible in figure 8. None of the nodes could have learned each other's local values.

To find answers on the security based research questions the documentation will be consulted.

## 5 Results

Vantage6 traffic from the server's perspective could be divided into two flows. The nodes and the server communicate their status updates through a websocket. The interaction between the data scientist and the server finds place in HTTP API requests. First, the data scientist authenticates and creates one or multiple tasks. Next the Vantage6 server uses the Vantage6 REST API to make the task(s) run on the right nodes. After performing a Carrier example task, in the pcap file it was found that the traffic keeps running through the Vantage6 server. The nodes do not know about each other's existence, even if they are in the same collaboration. When encryption is turned on, the Vantage6 server needs to decrypt and encrypt the traffic to pass the results to the next node. This makes the Vantage6 server the bottleneck inside the collaboration.

Knowing this bottleneck, one could think about recreating the Vantage6 network setup to one where the nodes can directly communicate to each other. In the method section it was determined an overlay network would be used with and without the Docker swarm mode on and a VPN network would be created with OpenVPN. In the following subsection, the performance results of these experimental setups will be discussed.

### 5.1 Performance

The throughput performance was tested using iperf3. In figure 9 the results are displayed. The bit rate on the VPN setup is the lowest, 66.5 Mbits/sec. VPN could slow traffic down, usually around five of six percent, but in these results this is a lot more. Since multiple containers are running on the same host, it is possible that resources must be shared and this influences the speed of the containers and their throughput. In comparison to a VM setup, the overlay network performs six times slower, 1.84 Gbits/sec on a overlay network created in swarm mode and 1.14 Gbits/sec with a separate consul key-value store. It is known overlay networks throughput is influenced by CPU capacity, as shown by Zismer [21].

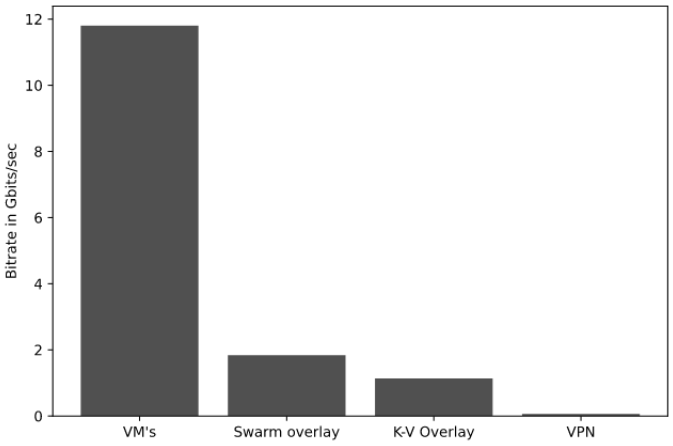


Fig. 9: Results: Iperf3 bitrates

The secure sum script simulated the real life scenario. This secure sum script, as can be found in Appendix A, is placed on the virtual machines to create algorithm containers. The results in table 1 show that the difference in the two variants of the overlay network are negligible. On an average of ten runs of the secure sum algorithm on three nodes on the overlay network with swarm mode the duration was 3.2397002 seconds. The same experiment on the overlay network with the key value store had an average duration of 3.292436 seconds. On average, the differences are milliseconds. Aside from this, it can be seen that the performance between the overlay network and a network between the virtual machines differs little. The average duration with three virtual machines was 3.089627 seconds. These results can be found in figure 10.

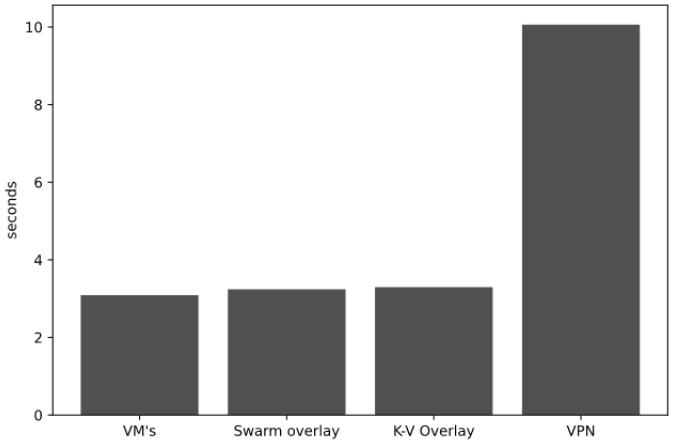


Fig. 10: Results: Average duration of calculation secure sum

### 5.2 Security

In swarm mode, swarm service management traffic is encrypted by default. It uses the Advanced Encryption Standard algorithm in Galois/Counter Mode where the symmetric keys are rotated every twelve hours by the swarm manager nodes. Application data is not encrypted by default, but encryption can be enabled by using the `--opt`

TABLE 1: Results of the Secure Sum algorithm

| Experiment | VM's | Overlay Swarm mode | Overlay Key-Value store | VPN |
|---|---|---|---|---|
| 1 | 0:00:03.105545 | 0:00:03.582516 | 0:00:03.390814 | 0:00:10.281000 |
| 2 | 0:00:03.081636 | 0:00:03.191026 | 0:00:03.195391 | 0:00:10.036331 |
| 3 | 0:00:03.084494 | 0:00:03.218464 | 0:00:03.203445 | 0:00:10.034723 |
| 4 | 0:00:03.089529 | 0:00:03.172414 | 0:00:03.217327 | 0:00:10.033763 |
| 5 | 0:00:03.078190 | 0:00:03.229757 | 0:00:03.194397 | 0:00:10.034258 |
| 6 | 0:00:03.080338 | 0:00:03.179556 | 0:00:03.245142 | 0:00:10.033010 |
| 7 | 0:00:03.101750 | 0:00:03.231941 | 0:00:03.204726 | 0:00:10.034662 |
| 8 | 0:00:03.092386 | 0:00:03.185545 | 0:00:03.196428 | 0:00:10.034415 |
| 9 | 0:00:03.082591 | 0:00:03.237920 | 0:00:03.269443 | 0:00:10.033298 |
| 10 | 0:00:03.099807 | 0:00:03.167863 | 0:00:03.250965 | 0:00:10.032882 |

`encrypted` option when creating an overlay network. This enables IPSEC encryption on application data. The IPSEC tunnels between the nodes also use the AES algorithm in GCM mode and manager nodes automatically rotate the keys every 12 hours.

Consul is used as a key-value store to make overlay networks without the swarm mode. When using this, multiple warnings are given about node discovery, and overlay networks with an external k/v store are deprecated and will be removed in a future release of Docker. Support to make use of this option is no longer available. The consul image has options to secure the setup, but is not secure by default. The gossip protocol that manages memberships and broadcast messages to the cluster is powered by serf. All gossip messages are encrypted using the AES-256 in GCM mode. Consul provides the option to use Access Control List to control access. Furthermore, there is an option to use RPC Encryption with TLS, but for that, Consul requires that all clients and servers have key pairs that are generated by a single Certificate Authority [5].

For the VPN node to node setup OpenVPN was used in the implementation. OpenVPN combines Secure Sockets Layer and Transport Layer Security to provide encryption. These two protocols are known as they utilize public-key cryptography to implement a secure connection over HTTP. In the setup used, only public-keys are utilized. SSL/TLS is designed to be tamper proof. As found in the TLS specification, RFC 4346, "no attacker can modify the negotiation communication without being detected by the parties to the communication" and "[t]he primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications." [11]. It is important to keep OpenVPN up to date, since the application is known for multiple security vulnerabilities as CRIME, BREACH and VORACLE [16].

### 5.3 Implementation inside Vantage6

As the results show, to create a network that connects the algorithm containers on multiple hosts, the overlay network is the best choice. Using Docker overlay networks without swarm mode is deprecated and this feature will be removed in a future release of Docker. It can be concluded that attaching nodes to the swarm and connecting the algorithm container to the overlay network could be a good implementation.

In this study the docker manager script of the Vantage6 node is changed to add this functionality. During the initialization of the node, the node joins the swarm. When calling the run function, the algorithm container is created. At this point, the algorithm container needs to connect to the overlay network. When implementing this feature, it was noticed that this was not an option. The Vantage6 framework is written in Python and makes use of the Docker-Py API to create the containers, but the Docker-Py API does not support overlay networking. Therefore, a workaround was needed. This challenge was bypassed by using the subprocess function.

In the proof of concept implementation, the swarm was initialized on the Vantage6 server virtual machine, and the overlay network was created on this machine as well. This was not added in the Vantage6 server scripts, but added manually. This was done for practical reasons, but the swarm and the overlay network could have been created on a separate machine to limit the power of the server machine.

## 6 Discussion

During the Carrier project, the data will be located on different locations. This means that when the nodes in a collaboration need to work together there are multiple setups that can achieve this. In the original setup, the Vantage6 server coordinates the tasks running on the nodes, even when the nodes need to collaborate directly with each other. In this study, the scope was to create a node to node setup without the interposition of the central Vantage6 server. Therefore, the focus was to create a Docker network between multiple hosts. However, there are other ways to create node to node communication, for example on Application level. In the Vantage6 infrastructure, a websocket is used to keep track of the status of the nodes by the Vantage6 server. The knowledge of the state of the node could be shared with the nodes in the collaboration if needed. This mechanism is also seen in the PyGrid framework. In this framework, gateways are used to find the right data and route the requests.

The Vantage6 framework is currently in development and features are added every few months. This results in outdated documentation. As a developer or data scientist,

you are dependent on the community and the development team of Integraal Kankercentrum Nederland to develop functionalities and algorithms. The question is whether the Vantage6 framework is production ready. However, currently, the availability of frameworks for multiparty computation for medical cases are limited, so there are not many choices.

This study did not test the actual performance of the overlay network inside the Vantage6 framework. During the experiments, only one algorithm container was running on the Docker host. The Vantage6 node container was turned off and removed from the host. When running a Vantage6 task, a minimum of two containers are running on the same host. Zismer's report warns about this. The CPU capacity influences the throughput and performance of the network. Aside from this, a workaround was needed to implement the overlay network in the Vantage6 framework. It is not known how this influences the performance of the network.

## 7 Conclusion

Analyzing the traffic on the Vantage6 infrastructure made it clear how the Vantage6 server handles the tasks on multiple nodes. This server divides the tasks over the nodes in the collaboration. The server is the only part in the infrastructure that knows about the existence and the availability of the nodes. Asymmetric keys are only used between an organization and the server, which means that the server could decrypt the traffic that should be private between nodes. This is a potential risk in case the Vantage6 server is compromised. Sending the traffic directly to the next node is a more efficient than sending it via the server.

Three infrastructures have been selected to implement direct node to node communication without the interposition of the Vantage6 server: an overlay network in Docker swarm mode, an overlay network with a separate key-value store and a VPN network. This makes the nodes direct accessible to each other by creating a new network layer with a new subnet. It can be concluded that an overlay network in swarm mode is the best solution. Node discovery and overlay networks with an external key-value store are deprecated and will be removed in the future. The use of VPN containers slows down the performance of tasks together. The cause of this could be that the CPU capacity is heavily burdened by the VPN client containers that route the traffic.

No attempt was made to tamper with, read or intercept the data or the models. But from the documentation of Docker, Consul and OpenVPN it can be concluded that, with the right options enabled, this is not possible. It would be nice if these options were enabled by default, so infrastructures were secure by design. Vantage6 has a configuration file that could be used to exclude algorithm images for use inside the infrastructure.

The three implementations of the node to node communication use different methods to encrypt the data. Docker swarm uses AES encryption and rotates the key every twelve hours. It is not mentioned if attacks are detected, but it is assumed by rotating the key that often, it is not possible to crack the key in such limited time. Consul uses AES too, but does not mention rotation of the key. It is possible to use TLS on encryption data. OpenVPN uses SSL/TLS to encrypt data. TLS is designed to make it impossible to modify the communications without being detected. The sender and the receiver will be notified when this happens.

The main question "How can Vantage6 edge nodes work together efficiently, without the interposition of the central server?" can be answered as follows: the nodes can work together efficiently by using an overlay network to connect containers on multiple Docker hosts. Docker supports this by connecting multiple host on a swarm. The central nodes can detect their peers by themselves by scanning the overlay network.

## 8 Future work

The scope of this study was limited to the implementation of a network to connect Vantage6 algorithm containers on multiple hosts. After gathering the results of the performance of the different infrastructures, an attempt was made to implement a connection of the algorithm containers to the overlay network. It was assumed that the Docker-Py library, which is used in Vantage6, supported this functionality. However, overlay networks are not supported in the Docker-Py library and a workaround needed to be created. In this proof of concept the subprocess module is used to create algorithm containers to connect the algorithm containers to the network. After this drawback, the decision was made not to measure the performance inside Vantage6 due to time limitations. However, it would be interesting to measure the difference between the secure sum algorithm without Vantage6 on an overlay infrastructure and the secure sum algorithm inside the Vantage6 infrastructure.

Currently, eScience Center is exploring the possibilities of PySyft and PyGrid for the Carrier project too. This framework had a different setup and node to node communication that is organized via PyGrid gateways. The roadmap of Vantage6 mentions direct communication between nodes for August 2021. This update will allow nodes to communicate with other nodes in their collaboration through a socket channel. It could be researched what the differences of these two frameworks are and how they perform relative to each other.

Lastly, in this research the security measures of Vantage6 and the proof of concept of the node to node infrastructure inside Vantage6 are not investigated further. The practical experiments that Brouwers did in 2017 could be repeated for this infrastructure to find weaknesses inside this framework.

## REFERENCES


[1] About datashield. https://www.datashield.ac.uk/about/. [Online; accessed 7-02-2021].

[2] Carrier. https://www.esciencecenter.nl/projects/carrier/. [Online; accessed 4-01-2021].

[3] Health insurance portability and accountability act of 1996 (hipaa). https://www.cdc.gov/phlp/publications/topic/hipaa.html. [Online; accessed 15-12-2020].

[4] The personal health train network. https://pht.health-ri.nl/. [Online; accessed 18-11-2020].

[5] Security model. https://www.consul.io/docs/security. [Online; accessed 1-03-2021].

[6] Vantage6.ai. https://vantage6.ai/. [Online; accessed 18-11-2020].

[7] Emma Bluemke. Pygrid: A peer-to-peer platform for private data science and federated learning. https://blog.openmined.org/what-is-pygrid-demo/. [Online; accessed 12-01-2021].

[8] Marcel Brouwers. Security considerations in docker swarm networking. *System and Network Engineering, University of Amsterdam*, 2017.

[9] Joris Claassen. Container network solutions research project 2. *System and Network Engineering, University of Amsterdam*, 2015.

[10] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Zhu. Tools for privacy preserving distributed data mining. *SIGKDD explorations*, 4(2):28–34, 2002.

[11] T. Dierks. The transport layer security (tls) protocol, April 2006. RFC 4346.

[12] David Gunning. Crypten: A new research tool for secure machine learning with pytorch. https://ai.facebook.com/blog/crypten-a-new-research-tool-for-secure-machine-learning-with-pytorch/. [Online; accessed 7-02-2021].

[13] Siem Hermans and Patrick de Niet. Docker overlay networks, performance analysis in high-latency environments. *University of Amsterdam*, 2016.

[14] Chia-Lun Lu, Shuang Wang, Zhanglong Ji, Yuan Wu, Li Xiong, Xiaoqian Jiang, and Lucila Ohno-Machado. Webdisco: a web service for distributed cox model learning without patient-level data sharing. *Journal of the American Medical Informatics Association*, 22(6):1212–1219, 2015.

[15] Frank Martin. About secure multi-party computation. https://distributedlearning.ai/blog-index/about-secure-multi-party-computation/. [Online; accessed 12-01-2021].

[16] Alex Mitchell. A deeper look into openvpn: Security vulnerabilities. https://sdtimes.com/softwaredev/a-deeper-look-into-openvpn-security-vulnerabilities/. [Online; accessed 28-02-2021].

[17] Arturo Moncada-Torres, Frank Martin, Melle Sieswerda, Johan van Soest, and Gijs Geleijnse. Vantage6: an open source privacy preserving federated learning infrastructure for secure insight exchange. In *AMIA Annual Symposium Proceedings*, pages 870–877, 2020.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[20] D Smits. vantage6-test-algorithms. https://github.com/CARRIER-project/vantage6-test-algorithms, 2021.

[21] Arne Zismer. Performance of docker overlay networks. *University of Amsterdam*, 2016.

# APPENDICES

## APPENDIX A
## SECURE SUM PYTHON SCRIPT

```
import socket
import selectors
import threading
import os
import random
import time
import datetime

event = threading.Event()

class Server:
   def __init__(self, host, port):
      self.server_addr = (host, port)
      self.sel = selectors.DefaultSelector()

   def listen(self):
      sock = socket.socket()
      sock.bind(self.server_addr)
      sock.listen(5)
      sock.setblocking(False)
      self.sel.register(sock, selectors.EVENT_READ, self.accept)

      while True:
         events = self.sel.select()
         for key, mask in events:
            callback = key.data
            callback(key.fileobj, mask)

   def accept(self, sock, mask):
      conn, addr = sock.accept() # Should be ready
      sock.setblocking(False)
      self.sel.register(conn, selectors.EVENT_READ, self.read)

   def read(self, conn, mask):
      data = conn.recv(1028) # Should be ready
      if data:
         global sum_value
         sum_value = data.decode()
         event.set()
      else:
         self.sel.unregister(conn)
         conn.close()


class Client:
   def __init__(self, node, ip):
      self.sel = selectors.DefaultSelector()
      self.node = os.environ.get('NODE', node)
      self.node_value = os.environ.get('NODE_VALUE', 82)
      self.ip = ip

   def secure_sum(self):
      result = 0
      next_number = int(self.ip[-1]) + 1
      next_ip = self.ip[:-1] + str(next_number)
      if int(self.node) == 1:
```

```
            # Start of the secure sum at node 1
            start = datetime.datetime.now()
            r = random.randrange(1000)
            result = r + int(self.node_value)
            self.sock = self.start_connection(next_ip, 4321)
            self.send_value(self.sock, result)
            # Secure sum comes back at node 1
            event.wait()
            final_result = int(sum_value) - r
            end = datetime.datetime.now()
            print("Final result:", final_result)
            print("Duration:", (end - start))
            f= open("test.txt","a")
            duration = end - start
            duration = str(duration)
            f.write("Duration: "+ duration + "\n")
            f.close()
         else:
            # Wait on sum result from other node
            event.wait()
            result = int(sum_value) + int(self.node_value)
            response = os.system("ping -c 1 " + next_ip)
            if response == 0:
               self.sock = self.start_connection(next_ip, 4321)
            else:
               self.sock = self.start_connection('<IP-address of node 1>', 4321)
            self.send_value(self.sock, result)

      def start_connection(self, host, port):
         server_addr = (host, port)
         sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         sock.connect_ex(server_addr)
         sock.setblocking(False)
         return sock

      def send_value(self, sock, value):
         data = str(value)
         sock.send(data.encode())

   def create_server(host, port):
      server = Server(host, port)
      server.listen()

   def main():
      host_ip = socket.gethostbyname(socket.gethostname())
      server_thread = threading.Thread(target=create_server, args=(host_ip, 4321))
      server_thread.start()
      client = Client(1, host_ip)
      client.secure_sum()

   if __name__ == '__main__':
      main()
```

## APPENDIX B
## RESULTS IPERF3 WITH VMS

```
Connecting to host 145.100.111.54, port 5201
[ 4] local 145.100.111.56 port 56244 connected to 145.100.111.54 port 5201
[ ID] Interval Transfer Bandwidth Retr Cwnd
[ 4] 0.00-1.00 sec 1.35 GBytes 11.6 Gbits/sec 0 3.04 MBytes
[ 4] 1.00-2.00 sec 1.41 GBytes 12.1 Gbits/sec 0 3.04 MBytes
[ 4] 2.00-3.00 sec 1.39 GBytes 11.9 Gbits/sec 0 3.04 MBytes
[ 4] 3.00-4.00 sec 1.38 GBytes 11.8 Gbits/sec 0 3.04 MBytes
[ 4] 4.00-5.00 sec 1.38 GBytes 11.8 Gbits/sec 0 3.04 MBytes
[ 4] 5.00-6.00 sec 1.40 GBytes 12.0 Gbits/sec 0 3.04 MBytes
[ 4] 6.00-7.00 sec 1.39 GBytes 11.9 Gbits/sec 0 3.04 MBytes
[ 4] 7.00-8.00 sec 1.39 GBytes 11.9 Gbits/sec 0 3.04 MBytes
[ 4] 8.00-9.00 sec 1.33 GBytes 11.4 Gbits/sec 0 3.04 MBytes
[ 4] 9.00-10.00 sec 1.28 GBytes 11.0 Gbits/sec 0 3.04 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bandwidth Retr
[ 4] 0.00-10.00 sec 13.7 GBytes 11.8 Gbits/sec 0 sender
[ 4] 0.00-10.00 sec 13.7 GBytes 11.8 Gbits/sec receiver

iperf Done.
```

## APPENDIX C
## RESULTS IPERF3 WITH SWARM OVERLAY

```
Connecting to host 10.0.1.8, port 5201
[ 5] local 10.0.1.5 port 37510 connected to 10.0.1.8 port 5201
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 5] 0.00-1.00 sec 217 MBytes 1.82 Gbits/sec 29 1.19 MBytes
[ 5] 1.00-2.00 sec 219 MBytes 1.84 Gbits/sec 0 1.31 MBytes
[ 5] 2.00-3.00 sec 216 MBytes 1.81 Gbits/sec 243 1.06 MBytes
[ 5] 3.00-4.00 sec 219 MBytes 1.84 Gbits/sec 0 1.20 MBytes
[ 5] 4.00-5.00 sec 221 MBytes 1.86 Gbits/sec 0 1.32 MBytes
[ 5] 5.00-6.00 sec 221 MBytes 1.86 Gbits/sec 62 1.42 MBytes
[ 5] 6.00-7.00 sec 224 MBytes 1.88 Gbits/sec 0 1.53 MBytes
[ 5] 7.00-8.00 sec 225 MBytes 1.89 Gbits/sec 0 1.54 MBytes
[ 5] 8.00-9.00 sec 218 MBytes 1.82 Gbits/sec 0 1.54 MBytes
[ 5] 9.00-10.00 sec 211 MBytes 1.77 Gbits/sec 0 1.54 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bitrate Retr
[ 5] 0.00-10.00 sec 2.14 GBytes 1.84 Gbits/sec 334 sender
[ 5] 0.00-10.04 sec 2.14 GBytes 1.83 Gbits/sec receiver

iperf Done.
```

## APPENDIX D
## RESULTS IPERF3 WITH KEY-VALUE STORE OVERLAY

```
Connecting to host 192.168.0.2, port 5201
[ 5] local 192.168.0.3 port 37380 connected to 192.168.0.2 port 5201
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 5] 0.00-1.00 sec 134 MBytes 1.13 Gbits/sec 63 834 KBytes
[ 5] 1.00-2.00 sec 141 MBytes 1.18 Gbits/sec 0 953 KBytes
[ 5] 2.00-3.00 sec 131 MBytes 1.10 Gbits/sec 46 758 KBytes
[ 5] 3.00-4.00 sec 135 MBytes 1.13 Gbits/sec 0 879 KBytes
[ 5] 4.00-5.00 sec 136 MBytes 1.14 Gbits/sec 31 988 KBytes
[ 5] 5.00-6.00 sec 135 MBytes 1.13 Gbits/sec 46 773 KBytes
[ 5] 6.00-7.00 sec 135 MBytes 1.13 Gbits/sec 0 896 KBytes
[ 5] 7.00-8.00 sec 139 MBytes 1.16 Gbits/sec 0 1003 KBytes
[ 5] 8.00-9.00 sec 129 MBytes 1.08 Gbits/sec 46 818 KBytes
[ 5] 9.00-10.00 sec 138 MBytes 1.15 Gbits/sec 0 935 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bitrate Retr
[ 5] 0.00-10.00 sec 1.32 GBytes 1.14 Gbits/sec 232 sender
[ 5] 0.00-10.00 sec 1.32 GBytes 1.13 Gbits/sec receiver

iperf Done.
```

## APPENDIX E
## RESULTS IPERF3 WITH VPN

```
Connecting to host 10.8.0.3, port 5201
[ 5] local 10.8.0.4 port 52108 connected to 10.8.0.3 port 5201
[ ID] Interval Transfer Bitrate Retr Cwnd
[ 5] 0.00-1.00 sec 8.04 MBytes 67.4 Mbits/sec 11 129 KBytes
[ 5] 1.00-2.00 sec 7.76 MBytes 65.1 Mbits/sec 19 98.6 KBytes
[ 5] 2.00-3.00 sec 8.13 MBytes 68.2 Mbits/sec 3 124 KBytes
[ 5] 3.00-4.00 sec 7.95 MBytes 66.7 Mbits/sec 12 105 KBytes
[ 5] 4.00-5.00 sec 7.83 MBytes 65.6 Mbits/sec 11 84.1 KBytes
[ 5] 5.00-6.00 sec 7.95 MBytes 66.7 Mbits/sec 0 137 KBytes
[ 5] 6.00-7.00 sec 8.01 MBytes 67.2 Mbits/sec 20 134 KBytes
[ 5] 7.00-8.00 sec 7.95 MBytes 66.7 Mbits/sec 11 131 KBytes
[ 5] 8.00-9.00 sec 7.89 MBytes 66.2 Mbits/sec 11 76.2 KBytes
[ 5] 9.00-10.00 sec 7.83 MBytes 65.6 Mbits/sec 1 117 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval Transfer Bitrate Retr
[ 5] 0.00-10.00 sec 79.3 MBytes 66.5 Mbits/sec 99 sender
[ 5] 0.00-10.05 sec 79.1 MBytes 66.0 Mbits/sec receiver

iperf Done.
```