Why wait? Let's start computing while data is still on the wire [⋆]

Shilpi Bhattacharyya [*] **Dimitrios Katramatos** [**] **Shinjae Yoo** [***]

[*] *Stony Brook University, Stony Brook, NY 11790, USA (e-mail: shbhattachar@cs.stonybrook.edu).*
[**] *Brookhaven National Laboratory, Upton, NY 11973, USA (e-mail: dkat@bnl.gov)*
[***] *Brookhaven National Laboratory, Upton, NY 11973, USA (email: sjyoo@bnl.gov)*

Abstract: In this era of Big Data, computing useful information from data is becoming increasingly complicated, particularly due to the ever increasing volumes of data that need to travel over the network to data centers to be stored and processed, all highly expensive operations in the long haul. In this paper we suggest that we can do computing and analysis of data "on the wire," i.e., while data is still in transit. The nature of these computations include analysis, visualization, pattern recognition, and prediction, or forecasting, on the streaming data. We follow a service function chaining architecture to implement this, assuming that the data packets arrive within a single network administrative domain. As a demonstration of this new computing paradigm, we present three examples. Firstly, we demonstrate pattern recognition and data visualization on streaming forex data, which can be used for lucrative trading in the forex market. In our second example, we analyze and learn user buying patterns from clickstream data streaming from multiple websites. Finally, we monitor solar sensors for a zero reading while the packets are still on their way to the data center, to schedule any maintenance and requisite repairs with no time delay.

Keywords: Software Defined Networking, Service Chaining Architecture, Big Data, Streaming Data, Algorithms, Analysis on Wire, Intelligent Networks

1. INTRODUCTION

With the Internet revolution, globalization, and digitization of everything and anything possible, there is a tremendous increase in the data volumes moving through the network. To make all this data useful in a larger way, we need to perform custom computations on them. Conventionally, after data sent from one end is received at the other end, any kind of computation is performed on them at either a specific data center or cloud computing is leveraged to quench the computational needs. Here, however, we suggest that it is feasible to start computations on the data as soon as it arrives at a specific point on the wire, which we refer to as our computing unit, and can be anywhere between the data entry point in the corresponding network and the destination data center, depending on problem scope and data availability. Such a point can be at the edge of the network, a little before the data center, or anywhere suitable in between.

We follow the Service Function Chaining(SFC) architecture[Halpern and Pignataro (2015)], which emphasizes how some legacy hardware devices functionality can be implemented with a SDN (Software Defined Networking) framework. But we present the idea with examples to show how can we compute on streaming data to inspect, analyze, forecast, or recognize specific patterns in the data. Our framework is called "Analysis on Wire" (AoW).

We started the AoW framework as a simple setup [Katramatos et al. (2016)], where we send a very simple string as "hello world" from one end to the other, with and without the SFC architecture, and compute the overhead of sending it through the chain, which becomes almost negligible with increasingly more data being sent. That implementation deployed seven virtual machines, configured with the Vagrant[HashiCorp (2017)] environment, which made the framework quite slow for Big Data.

The current implementation is based on the Docker [Docker (2017)]environment, which is relatively lighter and faster. We design a framework to do computations on the streaming data on the network. We run algorithms on this framework to visualize, forecast and analyze the incoming data into the network to infer useful information. This could help us not only in saving resources in the datacenters or alternatively any kind of cloud storage, but also in early decision making since data is processed in flight instead of having to wait for its arrival and accumulation at a data center before processing can begin. Sometimes we can even prevent or be prepared for impending disasters

[⋆] Sponsor and financial support acknowledgment goes here. Paper titles should be written in uppercase and lowercase letters, not all uppercase.

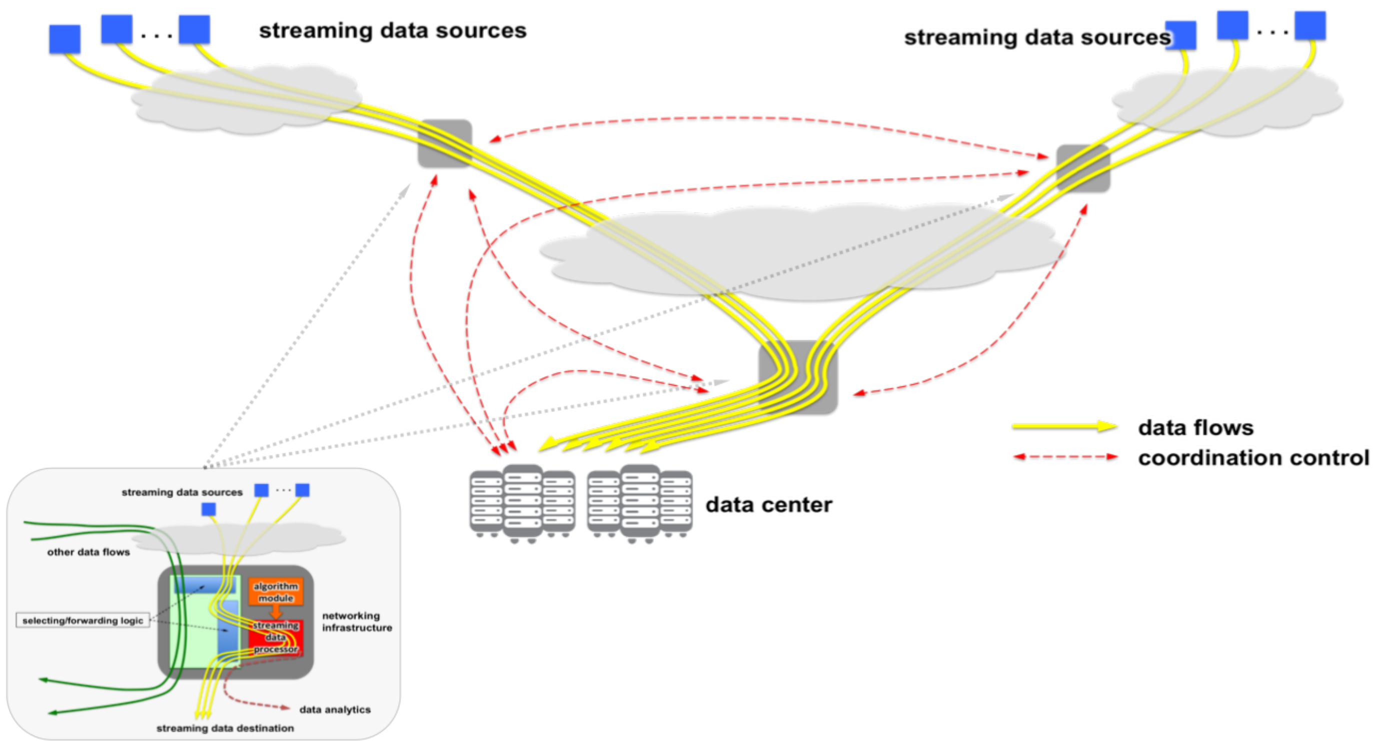


Fig. 1. General Overview of the Analysis on Wire(AoW) Framework with Compute-capable Network Nodes

such as device breakup; we demonstrate this for solar sensors.

In this paper, we present the functionality of the AoW framework with three examples. We run a pattern recognition algorithm on forex data to plan a better trade investment. We also analyse clickstream data from multiple streaming websites to identify user buying patterns. And, finally, we process in our computing unit streaming data from twenty three solar sensors to detect which of them is down and possibly schedule maintenance, requisite repairs or replacement for them.

2. ANALYSIS ON WIRE(AOW) FRAMEWORK DESIGN

The framework is a Docker-based service function chaining architecture with an OpenDaylight[Opendaylight (2017)]controller. We started developing this framework on top of the OpenDaylight Service Function Chaining demo[SFC (2017)]. The controller is a vagrant virtual machine(VM), which has other nodes as docker containers. All the docker containers, including the controller VM are Open vSwitches[OpenvSwitch (2017)]. The controller is responsible for monitoring the good operation of all other nodes in the framework. We feed the controller with the kind of path information our framework needs in order to perform a desired computation on the streaming data by passing json payloads through a REST API. The controller, in turn, prepares all other nodes for computation. A general overview of the framework we implemented can be seen in Fig.1 and it's working model is depicted in Fig. 2.

Following are the modules in the framework:

1. **Traffic Checker** - An Open vSwitch, which decides whether the incoming data packet is destined to enter the chain framework. It makes decisions according to the rules in it's flow table. The rules have been created in the traffic checker by the controller when we feed json payloads through REST APIs for the required configuration of the framework, once the controller vagrant VM starts. Rules are open ended and can be configured based on the needs of the data computing design framework. For our experiments, we have rules based on acceptable ip addresses and packet type. Once a data packet is accepted in the network, the data packet needs to move further towards it's destination. The traffic checker encapsulates the incoming data packets with Network Service Headers[P. Quinn and Pignataro (2017)] and transports them over a UDP protocol[S. Kumar and Melman (2017)] for further movement in the network. We discuss more about network service headers towards the end of this section.

2. **Forwarder** - Once, a packet has been classified to enter the AoW framework, it means it has been encapsulated with Network Service Headers, which guides it further in the network. It reaches a forwarder now, with a specific computing unit attached to it. The forwarder redirects the packet to the computing unit, which parses the packets to extract the payload and executes the desired algorithm on the data payload to do any kind of computation or analysis.

3. **Computing unit(CU)** - This is our data computing unit in the chain. It has two modules:

Data processing module: Extracts the payload from the incoming data packets, and converts them in a format which the corresponding algorithm module can accept as input.

Algorithm module: The custom algorithm module, wherein a plethora of algorithms such as pattern recognition, fore-

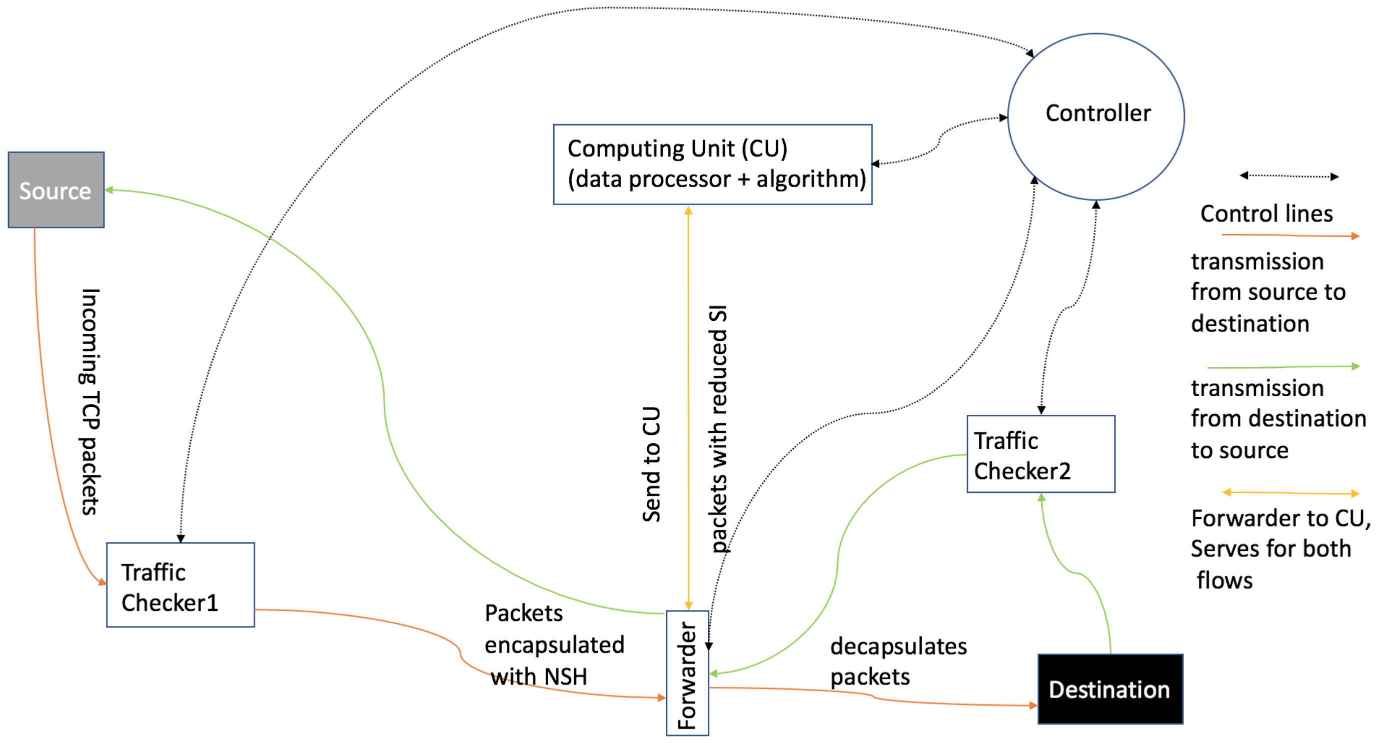

Controller
Computing Unit (CU)
(data processor + algorithm)
Source
Incoming TCP packets
Send to CU
packets with reduced SI
Control lines
transmission from source to destination
transmission from destination to source
Forwarder to CU, Serves for both flows
Traffic Checker2
Traffic Checker1
Packets encapsulated with NSH
Forwarder
decapsulates packets
Destination

Fig. 2. Working model of the Analysis on Wire(AoW) Framework

casting, and in general any form of streaming computation can be executed.

The CU extracts the payload from the encapsulated UDP packets it receives from the forwarder. The UDP packets contain the original packets, which in our case are TCP packets. From the enclosed TCP packets, the CU extracts the payload and transforms it into a CSV (comma separated values) format, which is fed to the algorithm running on the CU. We can execute any algorithm on the incoming traffic given it enters at a congestion controlled rate in a single administrative network, which is majorly true for streaming data. We can briefly explain these two terms as follows. Since, NSH encapsulated packets are transmitted over the UDP protocol, our setup cannot work extremely well (without any possible acceleration as through FPGAs[XILINX (2017)] or GPUs[NVIDIA (1999)]) if congestion occurs at any node in the network. A single administrative domain implies that we have full control over everything happening in the network at hand. This gives us the liberty to do any kind of computation on the packets entering the AoW framework.

Since, we are sending TCP packets, we need a bidirectional framework as illustrated in Fig. 2. Accordingly, we have two traffic checkers, one at source and other at destination. The forwarders with their CU are applicable for both directions.

Network Service Headers(NSH) ensure the successful implementation of our framework. The traffic checker encapsulates NSH headers based on the Computing Unit Paths and Computing Unit Chains, we feed to the OpenDaylight (ODL) controller. Our Computing unit chain consists of only a single Computing Unit. This chain is passed to the

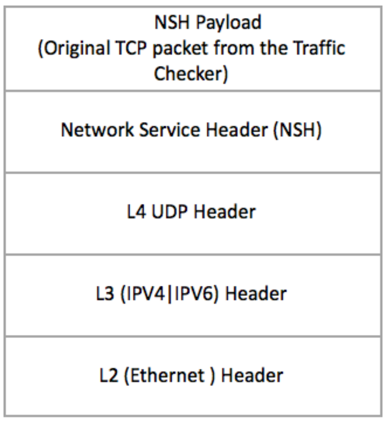

NSH Payload
(Original TCP packet from the Traffic Checker)
Network Service Header (NSH)
L4 UDP Header
L3 (IPV4|IPV6) Header
L2 (Ethernet) Header

Fig. 3. Network Service Header (NSH) UDP Stack

Computing Unit Path. In this framework, NSH headers are transported through the UDP protocol which works quite well in a congestion controlled environment as this. And at the end of the chain, the SI(service index) is reduced to zero at the computing unit and sent back to the forwarder. The forwarder decapsulates the packets and sends the original enclosed TCP packets intact to the destination. The NSH UDP stack is shown in Fig. 3.

In our setup, we run pattern recognition, detection, and behavior analysis algorithm on the incoming data. Our primary focus in this paper is to present the idea of using an SFC environment for generic computation and the design of such a network framework capable to do computations on streaming data. However, we are certain that there exists many meaningful computations that can be performed on our framework and it is in our future plans to explore a wide range of such computations.

The computing unit chain consists of the traffic checker, forwarder and it's attached computing unit. so it passes through as depicted in Fig. 2.

2.1 Implementation

We inject traffic into the network through a TCP packet generator implemented with a python script. This traffic tries to enter the framework at the traffic checker. So, once a data packet is at the checker, the checker tries to match the destination and packet type of the data packet with it's flow rules. Our traffic checker checks for TCP packets for a specific source network and a specific destination network. If there happens to be a match, the checker inserts appropriate NSH headers into it to move it forward through the framework. If there is no match, the data packet does not need to enter this framework, but it can go directly to it's destination based on custom routing policies. This is very useful in scenarios where we want data packets with specific attributes to enter our framework for possibly analyzing them or perform custom computations on them. As per the values in the inserted NSH headers on the top of original packet, the checker transmits it to the intended forwarder. The forwarder forwards the packets to it's attached computing unit(Fig. 2). The attached CU parses the packets through a data parser python script, and transforms the extracted payload to a CSV format within the data processing module. The output from data module is fed to the algorithm module, where we run the pattern recognition and detection algorithm as python scripts. More specifically, in the data processing module, we run tcpdump to capture the incoming traffic, where we parse and extract the payload from each packet. We run our algorithms on the parsed data within the algorithm module. This helps us to have a very useful insight of the moving data which is still in flight and we might not even need to store this data in some cases. Conventionally, we would wait at the destination for all data to be gathered, which might take hours and sometimes days to receive or we redirect all data towards cloud or edge and post that only we would run relevant algorithms on them. But, with this framework, we can keep getting information about incoming data soon after they enter the network. The only latency we get is the time taken by the checker to insert NSH headers onto the incoming data packets.

In the following subsection, we demonstrate in a detailed manner the three example algorithms mentioned earlier with computations on streaming data through the framework. Our focus is on the network design and these examples serve as evidence of what is possible in such a framework. We believe this to be a new computing paradigm, "computing in the network fabric" - which has the potential to be largely beneficial for this age of Big Data and certainly Bigger Data in the near future.

3. ALGORITHM EXAMPLES ON THE COMPUTING UNIT

3.1 Visualization and Pattern recognition algorithm on Forex data

The Forex market is one of the most liquid markets of the world. And this is a perfect example of streaming data, where ask and bid prices are streamed from servers throughout the world continuously. We demonstrate online computation here by analyzing the streaming data from one of such servers in the AoW framework. This server streams the timestamp, bid and ask prices for GBPUSD (British Pound to USD). An example of the data is shown in Fig. 4 in an intermediate processing state at CU. We can see the timestamp, bid and ask price at the tail of each packet.

The Pattern Recognition algorithm[Harrison (2013)] visualizes the Forex tick datasets for one day as seen in Fig.5, which is further used in pattern recognition. The green line indicates the ask prices and the blue line represents the bid prices.This is helpful in stock and forex trading as it gives an idea on whether to invest or not based on prior similar patterns calculated from the incoming data. We also store these patterns to predict future similar patterns by training our algorithm on these previous patterns[Harrison (2013)]. The forex data enters the network framework at the traffic checker. The data sent over the network is bid and ask prices for each second over a day in CSV format.

From the current ask and bid prices entering the network, the traders might want to predict in advance how this data is going to vary and whether it would be profitable to invest in forex at that certain point. As the data keeps entering the network framework, algorithm module at the computing unit keeps predicting the pattern of this bidding based on previously stored similar patterns. Accordingly, the involved traders can make a decision on investing or selling short based on the bid-ask spread.

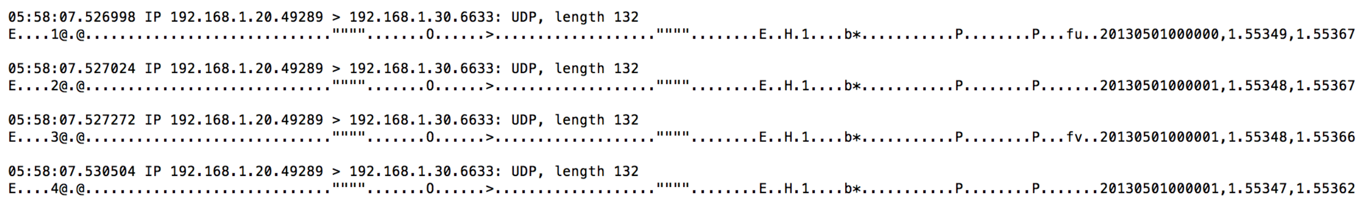
At the computing unit, we normalize our data points in percent change format. We calculate patterns for ten data points together. With each incoming data point, we get a new pattern with the last nine points. This current pattern is compared with previously stored patterns and the similarity percentage is calculated. If there is 70 percent or more similarity as visible in Fig. 6, a possibility of the same kind of bid and ask prices is predicted and the people or the computers involved can snoop and make a decision. This percentage is only for experimental purpose. So, if the incoming pattern is 70 percent or more similar to previous patterns, which had a profit in the past, the traders can consider investing in such scenario and perhaps make some profit.

The blue line is the current pattern in question and the green line is the matching similar pattern.

3.2 Clickstream analysis by media publishers

Media publishers, in order to make more profit and attract more customers, continuously stream user clickstream data from their websites for analyzing user interests and investment patterns and customize websites for each individual user.

This is a huge volume of data which is continuously streaming and we suggest we do the requisite computations on the wire as soon as we receive this data over a time period. This way, the computation is very quick and normally does not need storage post computation, unless storage is desired. In fact, the much smaller analyzed output can be much easier stored. These computations



```
05:58:07.526998 IP 192.168.1.20.49289 > 192.168.1.30.6633: UDP, length 132
E...1@.@..........................""""".......O......>..................""""".......E..H.1...b*..........P.......P...fu..20130501000000,1.55349,1.55367

05:58:07.527024 IP 192.168.1.20.49289 > 192.168.1.30.6633: UDP, length 132
E...2@.@..........................""""".......O......>..................""""".......E..H.1...b*..........P.......P......20130501000001,1.55348,1.55367

05:58:07.527272 IP 192.168.1.20.49289 > 192.168.1.30.6633: UDP, length 132
E...3@.@..........................""""".......O......>..................""""".......E..H.1...b*..........P.......P...fv..20130501000001,1.55348,1.55366

05:58:07.530504 IP 192.168.1.20.49289 > 192.168.1.30.6633: UDP, length 132
E...4@.@..........................""""".......O......>..................""""".......E..H.1...b*..........P.......P......20130501000001,1.55347,1.55362
```

Fig. 4. Forex data during an intermediate processing at CU

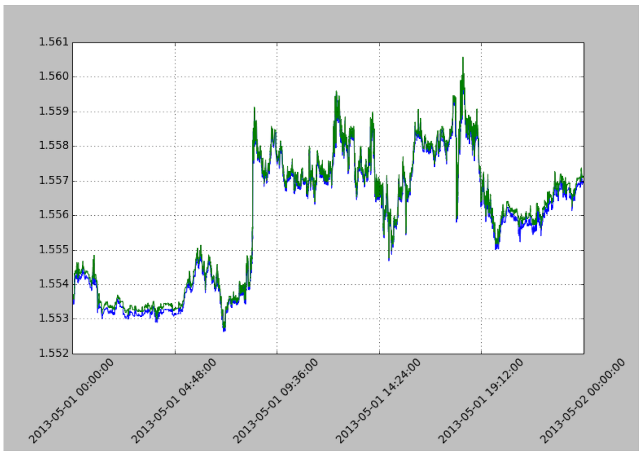


Fig. 5. Forex data visualization

can also let the publishers know earlier about individuals topmost choice of category for investment, with the help of which they can target these users for a sure shot buyer by showing relevant products. This is known as "Targeted Advertising."

We can actually do this by using the AoW framework. As soon as the data starts streaming from the websites for a media publisher, it enters the framework and reaches the computing unit through the forwarder, after being accepted at the traffic checker. Since the computing unit is close to the source, the data is available for computation earlier than it would be at the datacenters or cloud and we can start processing the data right away. Also, the data of any number of websites can be considered together depending on how deep in the network we choose to perform the computation instead of only the website(s) at the edge area, if using an edge computing model.

For our experiment, we assume six websites under a single administrative website are streaming data packets. These data packets contain unique user ids, ip address from where it is clicked and a lot of other information like browser details, location details from where the website is clicked[Opendata (2017)]. The data payload from a single page is as indicated in Fig. 7.

This can give a lot of information to publishers for example, ongoing trend in Sunnyvale or festival related hits in New York. Based on this information, the media publishers can prioritize content placements. This can even give a specific user's buying or browsing patterns. In our experiment, we get streaming data from six sources and find the browsing pattern for six users at each timestamp. The data is streamed every second from these webpages.

The data processing module parses the clickstream data from the different websites and converts it to CSV formatted text. This CSV formatted text is fed to the algorithm module, which calculates the majority category hits for all users(identified by unique user ids) as shown in Fig. 8.

3.3 Solar sensors streaming data analysis

We have twenty three solar sensors from which readings are streamed every second. We analyze this solar sensor data while they are still in transit to datacenters. As soon as the packets from the sensors are available, our framework pushes these data packets to the traffic checker from where it reaches the computing unit through forwarder with the help of encapsulation. Since the computing unit is not far from the traffic checker, the computing unit can analyze the sensor readings much before these readings even reach the datacenters.

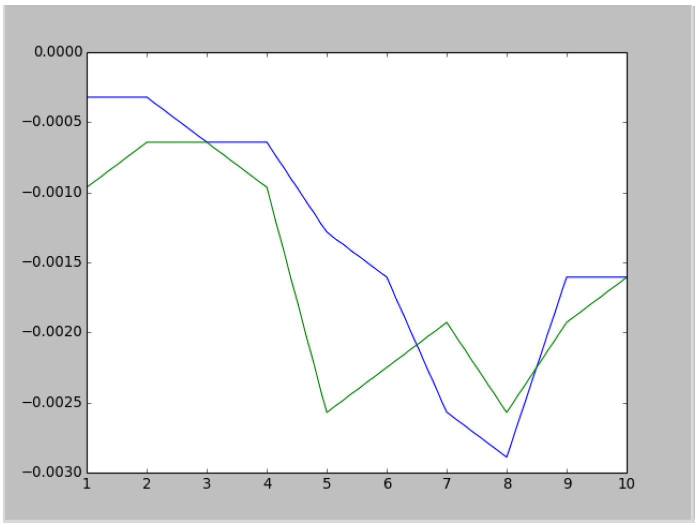
What we do at the computing unit is as follows: The data processing module at the computing unit casts the payload in the incoming data packets in a CSV format to feed to the algorithm module. The attached algorithm module parses the processed payloads to determine which of the sensor readings are zero. The zero reading of a solar sensor indicates that it is down or broken. This can be very useful as it can help to fix the sensor beforehand and if needed may be replace it with another sensor earlier than if we would have waited for the readings to reach the datacenter and take any action past that. This is an idea, which can be leveraged in the Internet of things (IoT)[IoT (2017)] arena too.

The CSV data format which is fed to the algorithm is shown as indicated in Fig 9. The sensor readings at two different timestamps passing over the network is depicted as in Fig. 10.

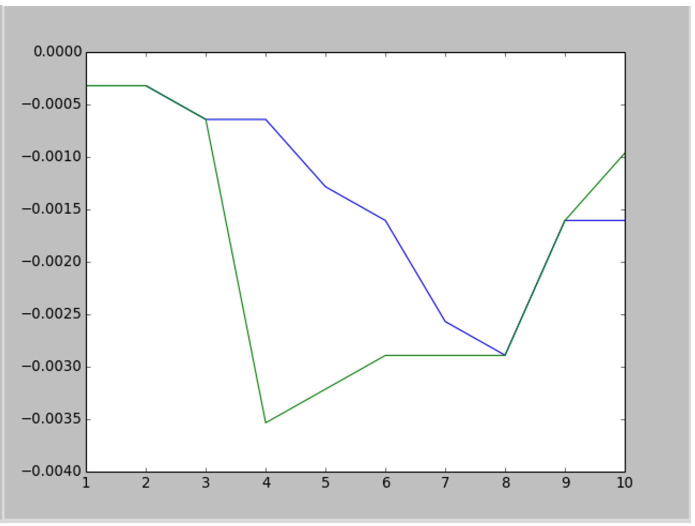
4. OBSERVATIONS AND RESULTS

We present a comparison between data packets going through our framework versus going straight to the destination in all three above cases. We depict graphs in two categories. The first category compares the time taken to send data packets from one end to another through the framework and directly without the framework as in Fig. 11, 12, 13. The second category compares the total time taken for data transfer plus processing of data packets plus performing the computation algorithm on them, again through the framework vs directly in which we process and compute on the packets at the destination as shown in Fig. 14, 15, 16.

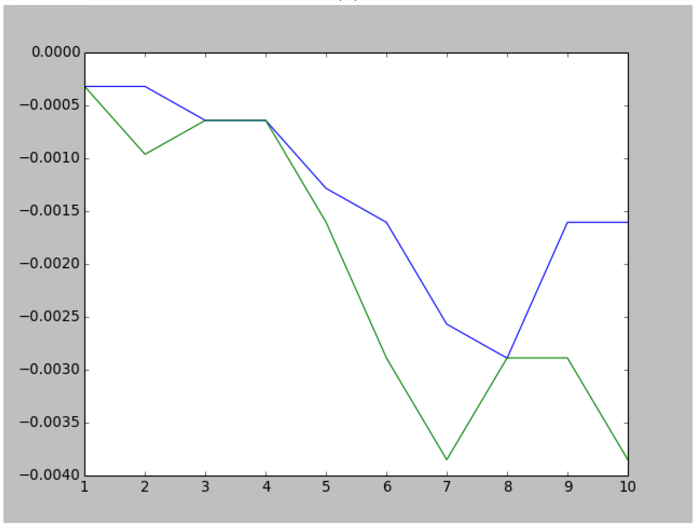
We observe that up to 1000 packets, the data packet transfer time through the framework vs. directly are almost



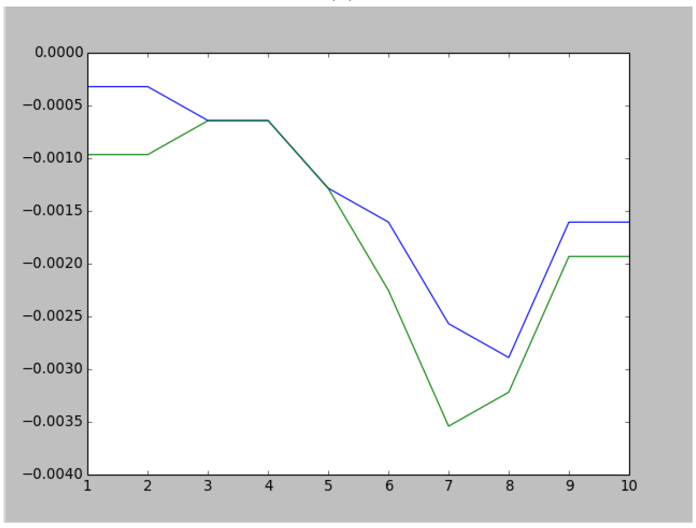
(a)



(b)



(c)



(d)

Fig. 6. Pattern with greater than 70 percent similarity

```
1331800486      2012-03-15 01:34:46     2859997896193943381     6917530184062522013     FAS-2.8-AS3     N       0       69.76.12.213    1       0               10      http://www.ac
me.com/SH55126545/VD55177927        {8D0E437E-9249-4DDA-BC4F-C1E5409E3A3B}                                                                                                 U        en
-us,en;q=0.5          591     0       0       U       U       Y       0       0       300     rr.com  15/2/2012 1:7:2 4 420   45      41              Mozilla/5.0 (Windows NT 6.1; WO
W64; rv:10.0.2) Gecko/20100101 Firefox/10.0.2     48      0       2       11      0       coeur d alene   usa     881     id      0       0       0       0
                                 0                                                                                                                                      KXLY
                            0                                                                                       120
                                                                                                                                KXLY

                                                                                                                                                                         0
```

Fig. 7. Clickstream payload from a single page

the same. Beyond this rate, there is quite a difference as we understand our framework works best in a congestion controlled environment which we dont have here; sending packets at a higher rate leads to congestion in the network and hence the delay. But, one important thing which we want to mention is that, since the original communication protocol is TCP, there is a guarantee on packet delivery. The additional operation of encapsulating every packet is bound to impose some limitations on the useful rate of packets through the system before congestion appears. Since our environment is meant mostly for proof of concept it is not particularly geared for speed. However, there are several ways to achieve higher rates which we plan to investigate in the near future.

Again, the comparison of data processing time plus algorithm execution varies in the three algorithms from that in their direct path and computation thereafter. This depends on the kind of computation we do in our computing unit. For example, in the forex data, we do much more computing than the clickstream data and the solar sensors as we visualize, store patterns and forecast future patterns. In this kind of compute intensive scenarios in our framework, we can think of using possible acceleration through GPUs and FPGAs, which is one of our ongoing projects. For the other two cases of clickstream data and solar sensors, the computation is of low overhead, the only difference comes from the NSH encapsulation and decapsulation at the checker and forwarder respectively. The NSH encapsulation is something that can be accelerated, and

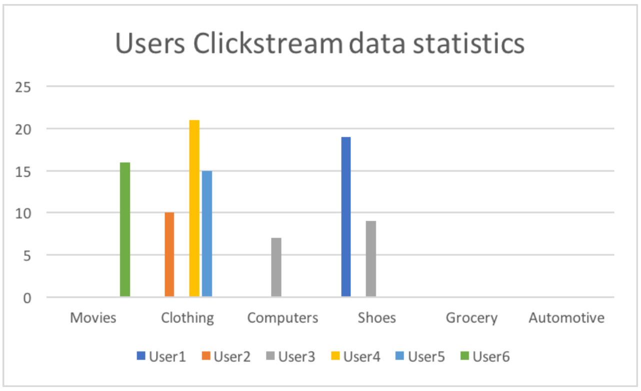


Fig. 8. User Clickstream data statistics

we also plan to examine alternative approaches that may eliminate the need for such headers in certain cases.

ΔT between computation time of the AoW framework and Direct path :

We define ΔT here as the total time difference experienced by computation on streaming data packets through the framework vs directly to the destination and computation thereafter. ΔT is the difference introduced by the AoW framework in this particular implementation. It is the sum of the delays introduced by the NSH encapsulation at the traffic checker, the forwarding time by the forwarder, computation time by the computing unit(data processing module plus algorithm module), NSH header processing at each unit in the framework like decreasing service index by the computing unit and removing headers by the forwarders.

As we see in the Fig. 17, minimum overhead is 44 seconds in case of Forex data at around 60 packets every second. So, the setup does not seem ideal for this kind of computation unless some acceleration is applied at the computing unit or for NSH encapsulation and associated operations. With acceleration, this framework is expected to be able to handle flows of much higher bandwidth and we are already working towards this direction.

From Fig. 18 and 19, we see that we get results at almost no delay through our framework than sending the data directly and then doing any kind of computation on it. This is possible because computation has negligible overhead for these cases and since the computing unit is close to the sender, we can get results with almost no delay through our AoW framework in these cases. These graphs also show a minor negative difference between the computation through the framework and direct path for some values. We account for this as a measurement tolerance of the setup.

4.1 Abbreviations and Acronyms

SFC: Service Function Chaining

UDP: User Datagram Protocol

SDN: Software Defined Networking

SI: Service Index

NSH: Network service header

VM: Virtual machine

REST: Representational state transfer

API: Application programming interface

TCP: Transmission control protocol

CU: Computing Unit

UDP: User Datagram Protocol

MTU: Maximum Transmission Unit

FPGA: Field-programmable gate array

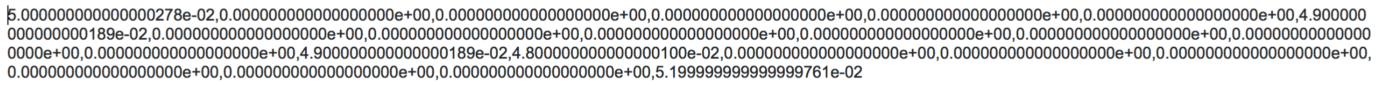
GPU: Graphics processing unit

IoT: Internet of things

5. RELATED WORK

The SFC architecture is being developed and standardized as a means to interconnect virtualized network functions and create a modern and effective SDN solution for today's datacenters. By replacing specialized hardware with virtual components running in VMs on powerful compute nodes, a lot is gained in terms of performance, scalability, renewability, and reduced cost. Clearly, the work performed by a virtual firewall or an intrusion detection system is very complicated and also has to be executed quickly and on streaming data. As another example, consider work done by SURFnet to transcode a 4K video stream [van der Pol (2016)] with service functions performing video transcoding placed in clouds. Our philosophy, and approach, is that a virtual SFC environment sounds like a prime candidate for the distributed computing in the network fabric that we envisage as a new computing paradigm. In our pursuit, however, we are not bound by any particular technique or problem restriction; we simply want to go far beyond the bounds of networking and virtual network functions and devise a framework that can execute any reasonable algorithm on streaming data, while also investigating the behavior and performance of such algorithms to determine the feasibility of solving certain problems on the wire. In this respect, we not only utilize the SFC architecture as it is being standardized, but also seek to simplify, enhance, and combine it with other software and hardware technologies to create our AoW framework. In this paper we essentially present our first attempt at a fully virtual approach.

We should also point out the fundamental differences our approach has when compared with cloud computing[Azure (2017)], edge computing[Jake Jones (2017)], fog computing[S. Chen and Shi (2017)], and mist[Martin (2015)] computing. In cloud computing, such as in Amazon AWS[Amazon (2017)], computation and storage is performed in virtual data centers that are put together dynamically. It is a very popular computing paradigm which works very well on streaming data and perform all kinds of computations under the flavors of SaaS (Software as a service), PaaS (Platform as a service) and IaaS (Infrastructure as a service). However, computation (and storage) still takes place in datacenters which can be far away from the data sources and data is subjected to significant latencies, overheads, and delays before any useful computation can be performed on it. Edge computing[Jake Jones



```
5.000000000000000278e-02,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,4.900000
000000000189e-02,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,0.00000000000000
0000e+00,0.000000000000000000e+00,4.900000000000000189e-02,4.800000000000000100e-02,0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,
0.000000000000000000e+00,0.000000000000000000e+00,0.000000000000000000e+00,5.199999999999999761e-02
```

Fig. 9. Solar sensor payload at the data processing unit of CU

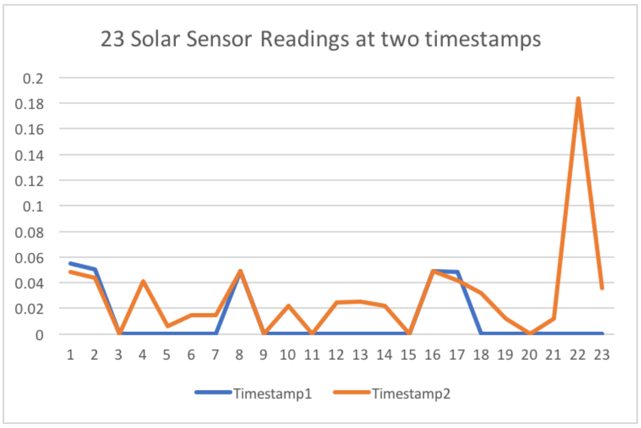


Fig. 10. 23 Solar sensor readings at two timestamps

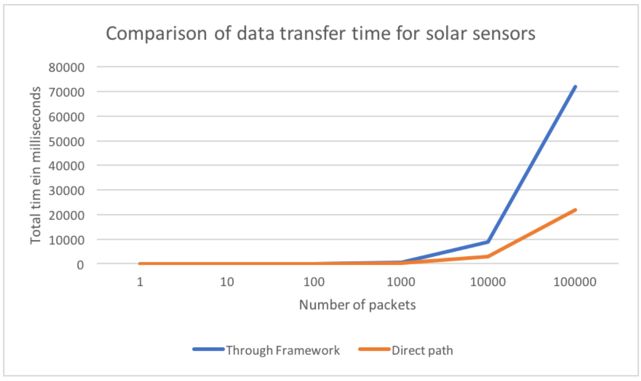


Fig. 13. Comparison of data transfer time for solar sensors

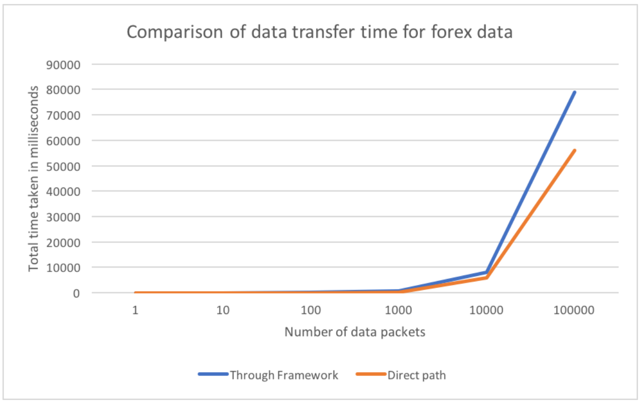


Fig. 11. Comparison of data transfer time for forex data

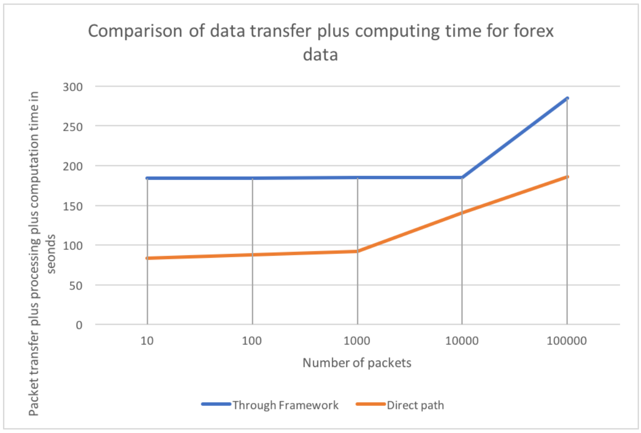


Fig. 14. Comparison of data transfer plus computing time for forex data

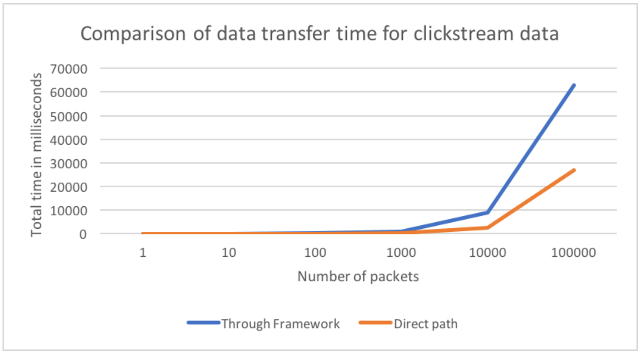


Fig. 12. Comparison of data transfer time for clickstream data

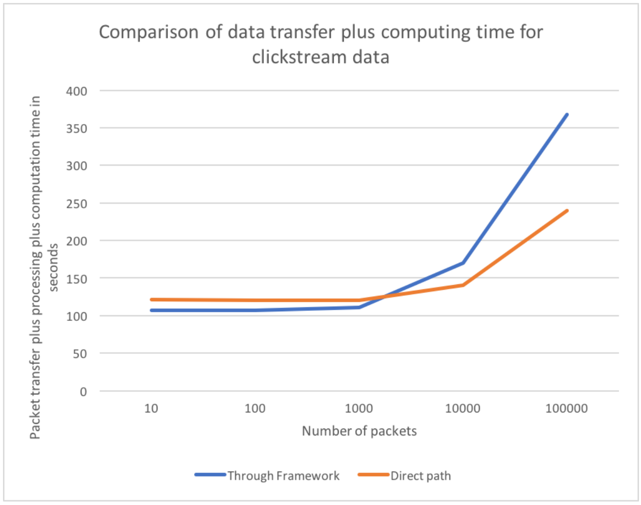


Fig. 15. Comparison of data transfer plus computing time for clickstream data

(2017)] facilitates the operation of compute, storage, and networking services between end devices and datacenters; computation is performed at the physical edge of the network. This paradigm is also called fog computing. Mist computing[Martin (2015)] is the latest paradigm involving computing in the very end devices found at the edge of the network to assist in the motion of data towards the fog and the cloud. All paradigms are intimately linked with the Internet of Things (IoT) with the huge numbers of data

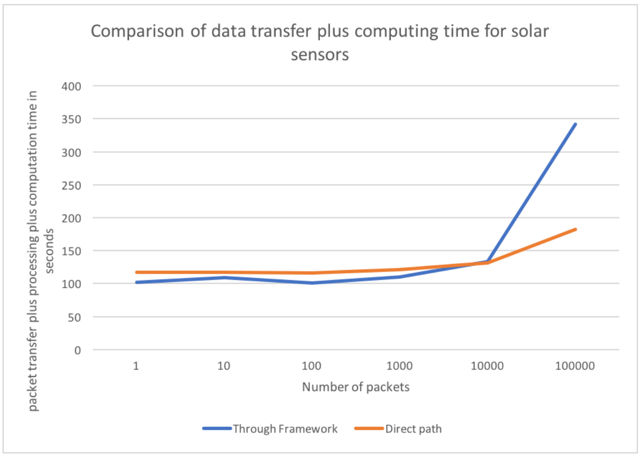


Fig. 16. Comparison of data transfer plus computing time for solar sensors

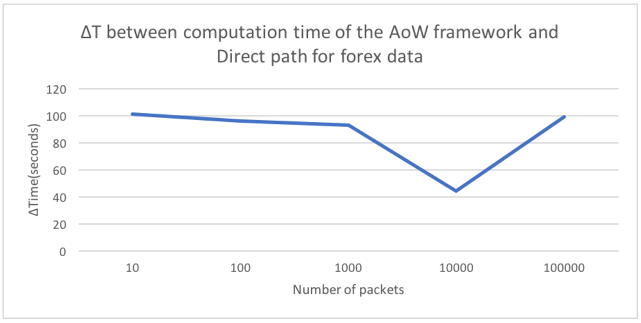


Fig. 17. ΔT between computation time of the AoW framework and Direct path for forex data

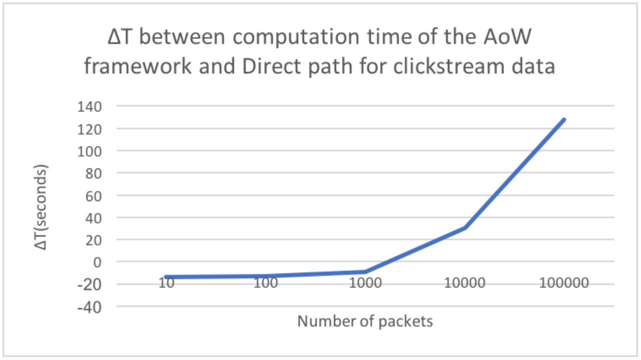


Fig. 18. ΔT between computation time of the AoW framework and Direct path for clickstream data

sources/destinations at the extremes of the network and were conceived to facilitate the operation and reduce the volume of traffic through the network that could eventually drown data centers with data tsunamis. The Analysis on the Wire computing paradigm, as we have presented in the original idea paper[Katramatos et al. (2016)] covers the network area between data centers and edge and can be adjusted to include nodes at or near the edge or intermediate nodes, or ones closer to the data centers, depending on the scope and locality of the problem of interest. At the user edge we consider cases where computing will take place fog-level devices (but not mist-level); at the data center side, we can reach the WAN provider switches just before a sites border router(s); and anywhere in between. Network nodes with native, attached, or even remote computing

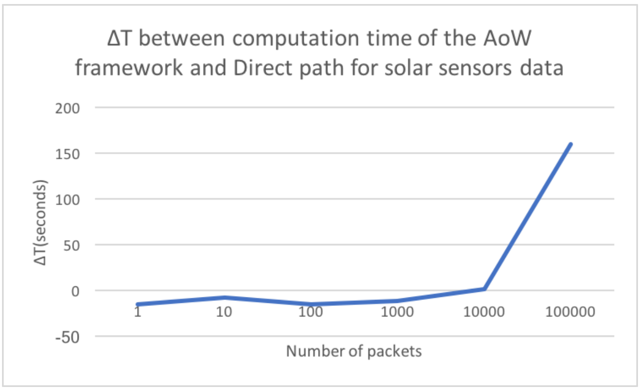


Fig. 19. ΔT between computation time of the AoW framework and Direct path for solar sensors data

capabilities can be coordinated with a layer of suitable middleware to form a flavor of distributed computer with, we believe, great potential for performing unprecedented style computations in the network fabric. For example, one could form a computer with a number of edge nodes to extract specific information from sensor network domains and also include a few upstream nodes in a hierarchy that could apply a cascade of different computations on the selected data before ever arriving at a data center.

6. CONCLUSION AND FUTURE WORK

There is tremendous opportunity to perform computations on wire. Such an approach can help in saving resources at datacenters for the current as well as the impending Big Data age, earlier decision-making, and faster results. We demonstrated the concept for a single administrative network domain in a congestion controlled environment. Some areas for immediate applications are power grid, weather prediction, cybersecurity for detecting anomalous patterns, and elephant (high bandwidth, high volume, long duration) flow analysis.

In situations where the data rate exceeds the capacity of a basic AoW framework, we can utilize multiple parallel computing units with a divide and conquer approach: each unit performs on a fraction of the data, thereby rendering load balancing as well as faster results and efficient usage of framework resources.

We have talked throughout the paper assuming data packets are independent, and MTU is large enough to fit the entire data at a point of time in one packet. There may be scenarios, where payload and header exceeds the MTU size of transmission protocol and these are the situations, where we need to apply defragmentation on the packets to do complex computing. This is something, which we plan to do in our future work. We are already looking into the acceleration of computation on the AoW framework by employing GPUs and FPGAs for compute intensive algorithms and are working towards a full-fledged implementation of the AoW framework.

Finally, in this paper we focused on demonstrating the concept of the AoW framework using a single computing unit. We are already working towards a multiple-node geographically-distributed prototype and our future plans include experimentation with different computing schemes and algorithms in a distributed fashion.

REFERENCES

Amazon (2017). Streaming data. https://aws.amazon.com/streaming-data/.

Azure (2017). Cloud computing. https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

Docker (2017). Docker. https://www.docker.com/what-docker.

Halpern, J. and Pignataro, C. (2015). Service Function Chaining (SFC) Architecture. RFC 7665, Internet Engineering Task Force (IETF) . URL https://tools.ietf.org/html/rfc7665.

Harrison (2013). Pattern recognition algorithm on forex tick datasets. https://pythonprogramming.net/machine-learning-pattern-recognition-algorithmic-forex-stock-trading.

HashiCorp (2017). Vagrant. https://www.vagrantup.com/intro/index.html.

IoT (2017). Internet of things. https://www.forbes.com/sites/bernardmarr/2017/05/05/internet-of-things-and-predictive-maintenance-transform-the-service-industry/.

Jake Jones, S. (2017). Edge computing: The cloud, the fog and the edge. https://www.solid-run.com/edge-computing-cloud-fog-edge/.

Katramatos, D., Yue, M., Yoo, S., van Dam, K.K., Xu, J., and Zhang, J. (2016). Streaming data analysis on the wire. In *2016 New York Scientific Data Summit (NYSDS)*, 1–7. doi:10.1109/NYSDS.2016.7747816.

Martin, M.J. (2015). Cloud, fog, and now, mist computing. https://www.linkedin.com/pulse/cloud-computing-fog-now-mist-martin-ma-mba-med-gdm-scpm-pmp.

NVIDIA (1999). Graphics processing unit(gpu). http://www.nvidia.com/object/gpu.html/.

Opendata (2017). Clickstreamdata. https://opendata.stackexchange.com/questions/1779/clickstream-sample-dataset/.

Opendaylight (2017). Opendaylight. https://www.opendaylight.org/.

OpenvSwitch (2017). Open vswitch. http://openvswitch.org/.

P. Quinn, U.E. and Pignataro, C. (2017). Network Service Header (NSH) draft-ietf-sfc-nsh-19. Internet-Draft draft-ietf-sfc-nsh-19, Internet Engineering Task Force (IETF) . URL https://tools.ietf.org/html/draft-ietf-sfc-nsh-19.

S. Chen, T.Z. and Shi, W. (2017). Fog computing. In *IEEE Internet Computing, vol. 21, no. 2, pp. 4-6, Mar.-Apr. 2017*, 1–3. doi:10.1109/MIC.2017.39.

S. Kumar, L. Kreeger, S.M.W.H.R.M. and Melman, D. (2017). UDP Transport for Network Service Header draft-kumar-sfc-nsh-udp-transport-03. Internet-Draft draft-ietf-sfc-nsh-19, Internet Engineering Task Force (IETF) . URL https://tools.ietf.org/html/draft-kumar-sfc-nsh-udp-transport-03.

SFC, O. (2017). Opendaylight sfc demo. https://github.com/opendaylight/sfc/.

van der Pol, R. (2016). Experiences with OpenDaylight Service Function Chaining (SFC). Presentations, SURFnet . URL https://kirk.rvdp.org/presentations/FOSDEM-2016-rvdp-SFC.pdf.

XILINX (2017). Field-programmable gate array(fpga). https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html/.