Future Generation Computer Systems 89 (2018) 563–574

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Why wait? Let us start computing while the data is still on the wire[☆]

Shilpi Bhattacharyya [a,*], Dimitrios Katramatos [b], Shinjae Yoo [b]

[a] *Stony Brook University, Stony Brook, NY 11790, USA*
[b] *Brookhaven National Laboratory, Upton, NY 11973, USA*

HIGHLIGHTS

- A framework namely Analysis on Wire (AoW) capable of computing on the wire.
- AoW can help save resources in the data centers and/or cloud.
- AoW can help in making earlier decisions in relevant scenarios as trading.
- Computations include analysis, visualization, pattern recognition, and forecasting.
- We present three examples - forex trading, media publishing, and monitoring sensors.
- AoW can be deployed anywhere in the network between the source and the destination.

ARTICLE INFO

Article history:
Received 1 February 2018
Received in revised form 9 June 2018
Accepted 14 July 2018
Available online 17 July 2018

Keywords:
Software defined networking
Service function chaining
Big Data
Streaming data
Machine learning in networks
Pattern recognition
Analysis on wire
Intelligent networks
Internet of Things (IoT)

ABSTRACT

In this era of Big Data, computing useful and timely 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. This is a strong motivation to explore how to perform computing and analysis of data "on the wire", i.e., while the data is still in transit. The nature of these computations include analysis, visualization, pattern recognition, and prediction on the streaming data. In this paper we present the idea of a framework capable of analyzing data in transit based on the principles of a service function chaining architecture. This framework can be deployed at any practical location within the network where computation on data flows is desirable. We further describe an all-virtual implementation of the framework as a worst-case scenario and present an early investigation of its capabilities with three examples — pattern recognition and data visualization on streaming Forex data, targeted advertising from clickstream data, and processing of monitoring data from solar sensors for maintenance decisions. Our results indicate that performing computations on live data flows to provide immediate perspective on the data is possible and attractive, but also that performance heavily depends on the amount and capabilities of the dedicated resources.

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 e.g., an institute like CERN produced 73 PB of data in the year 2016 running their Large Hadron Collider [1]. To make all this colossal amounts of data useful, we perform custom computations on them. Conventionally, when data sent from one end is received at the other end — any kind of computation is performed on it at specific data centers or by utilizing newer computing paradigms as cloud, edge and mist computing to quench the computational needs. In cloud computing, such as in Amazon AWS [2], computation and storage is performed in virtual data centers that are put together dynamically. It is a popular computing paradigm which works well on streaming data and performs all kinds of computations under the flavors of SaaS (Software as a service), PaaS (Platform as a service) and IaaS (Infrastructure as a service). Nevertheless, computation (and storage) still takes place in data centers which can be far away from the data sources and data is subjected to significant latencies, overheads, and delays before any meaningful computation can be performed on it. Edge computing [3–5] facilitates the operation of compute, storage, and networking services between end devices and data centers; computation is performed at the physical "edge" of the network. This paradigm is also called "fog computing". Mist computing [6,7] is the latest paradigm involving computing in the

[☆] This project has been funded by Brookhaven National Laboratory, USA (contract number: DE-SC0012704).
[*] Corresponding author.
E-mail addresses: shbhattachar@cs.stonybrook.edu (S. Bhattacharyya), dkat@bnl.gov (D. Katramatos), sjyoo@bnl.gov (S. Yoo).

https://doi.org/10.1016/j.future.2018.07.024

very end devices found at the edge of the network to assist in the motion of data towards the fog and the cloud. All these paradigms are intimately linked to the Internet of Things (IoT) [8] with the huge numbers of data sources and 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. 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 by deploying the AoW framework there and this point can be at the edge of the network, close to the data center, or anywhere applicable in between depending on problem scope and data availability. We believe this framework can enable researchers to go beyond the IoT horizon because of its inherent flexibility to be deployed anywhere between the source and the destination.

We follow the Service Function Chaining (SFC) architecture [9] to build the AoW framework. SFC emphasizes that the functionality of some legacy hardware devices can be implemented with the concept of Software Defined Networking (SDN). To the best of our knowledge, throughout the literature the emphasis of SDNs has always been towards a better network management [10,11]. But, we present a high performance computing perspective for SDNs here to enable "on the wire" computation. We present our idea with examples to show how we can compute on streaming data to inspect, analyze, forecast and recognize specific patterns in the data.

We started the AoW framework as an uncomplicated setup [12], where we send a 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 [13] environment, which made the framework quite slow for Big Data.

The current implementation is based on the Docker [14] environment, which is relatively lighter and faster. We design a SDN 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 data centers or alternatively any kind of cloud storage, but additionally 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 on it. With the help of AoW, we can even prevent or be prepared for impending disasters such as device breakup; we demonstrate this for solar sensors.

We present the functionality and "on the wire" compute potential of the AoW framework with three examples. We run a pattern recognition algorithm on Forex data to plan a better trade investment. We analyze clickstream data from multiple streaming websites to identify user buying patterns. And, finally, we process a streaming data from twenty three solar sensors in the AoW framework to detect which of them is down and possibly schedule maintenance, requisite repairs or replacement for them.

The paper is organized as follows. We discuss the conceptual design of the AoW framework in Section 2.1. The implementation details of the preliminary framework is discussed in Section 2.2, describing the functionality and capabilities of all of its components. The operation of the framework is described in Section 2.3. We start discussing various algorithms which we are executing on the AoW framework under three subsections of Section 3. The observations and the results of the experiments are presented in Section 4. We highlight the related research work in this area and some potential use cases in Sections 5 and 6 respectively. We analyze the trade-off and limitations of the current implementation in Section 7. Finally, we conclude in Section 8, reflecting the future scope of the work as well as our current progress therein.

2. Analysis on Wire (AoW) Framework

2.1. Conceptual design

The AoW framework is designed on the principles of Software Defined Networking [10] where we separate the intelligence (control plane) of the network from its forwarding capability (data plane). The conceptual design of the AoW framework can be seen in Fig. 1 and as depicted, we envisage deployment of AoW-enhanced nodes to multiple locations in a network, spanning multiple domains. These nodes can be coordinated through a layer of middleware to form a distributed computer that can be used to process streaming data originating at multiple geographically distributed sources. The distributed aspect of AoW is beyond the scope of this paper as here we focus on the concept of a single node.

2.2. Preliminary implementation

The preliminary implementation as depicted in Fig. 2, is essentially a Docker-based service function chaining architecture with an OpenDaylight (ODL) [15] controller. We started developing this framework on top of the OpenDaylight Service Function Chaining demo [16]. The controller is a Vagrant virtual machine (VM), which has other nodes as docker containers. All the docker containers are Open vSwitches [17]. We feed the controller with the necessary path information for our framework to perform a desired computation on the streaming data by passing json payloads through a Representational State Transfer (REST) Application Programming Interface (API). The controller, in turn, prepares all other nodes for computation. The controller is also responsible for monitoring the good operation of all other nodes in the framework. The modules in the framework are described as follows.

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 created in its flow table as shown in Fig. 4. The rules have been created in the traffic checker by the controller when we feed json payloads through REST API for the required configuration of the framework, once the controller on Vagrant VM starts. Rules are open ended and can be configured based on the demands 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 into the AoW framework at the traffic checker, the data packet needs to move further towards its destination. The traffic checker encapsulates the incoming data packets with Network Service Headers (NSH) [18] and transports them over User Datagram Protocol (UDP) [19] for further movement in the network. We discuss more about NSH towards the end of this section. Traffic Checker is used interchangeably with Checker hereafter.

Forwarder - Once, a packet has been classified to enter the AoW framework, it means it has been encapsulated with NSH, which guides it further in the network. It reaches a Forwarder now, with a specific Computing Unit (CU) attached to it. The Forwarder redirects the packet to the CU, which parses the packets to extract the payload and executes the desired algorithm on the data payload to do any kind of computation and/or analysis.

Computing unit (CU) - This is our data computing unit in the chain. It has two modules: (a) **(Streaming) Data processor module** - which extracts the payload from the incoming data packets, and converts them in a format which the corresponding Algorithm module can accept as an input, (b) **Algorithm module** - which is the custom algorithm module, wherein a plethora of algorithms such as pattern recognition, forecasting, and in general any form of streaming computation can be executed.

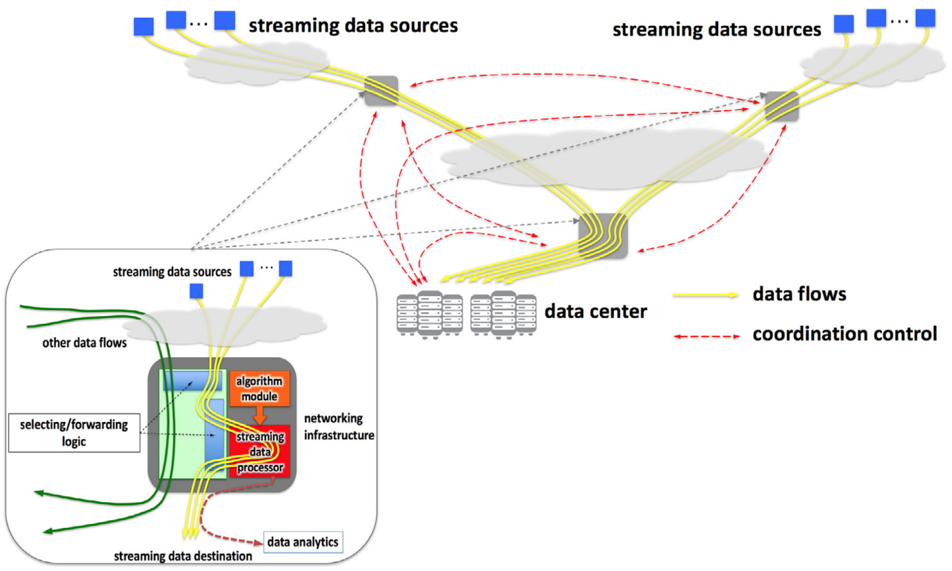


Fig. 1. General overview/conceptual picture of the Analysis on Wire (AoW) framework with compute-capable network nodes. It shows the general layout of a network where there are multiple distributed sources streaming data towards the data centers. The AoW framework can be deployed at multiple locations to perform heterogeneous computations on the data streaming through the network before it reaches the data centers.

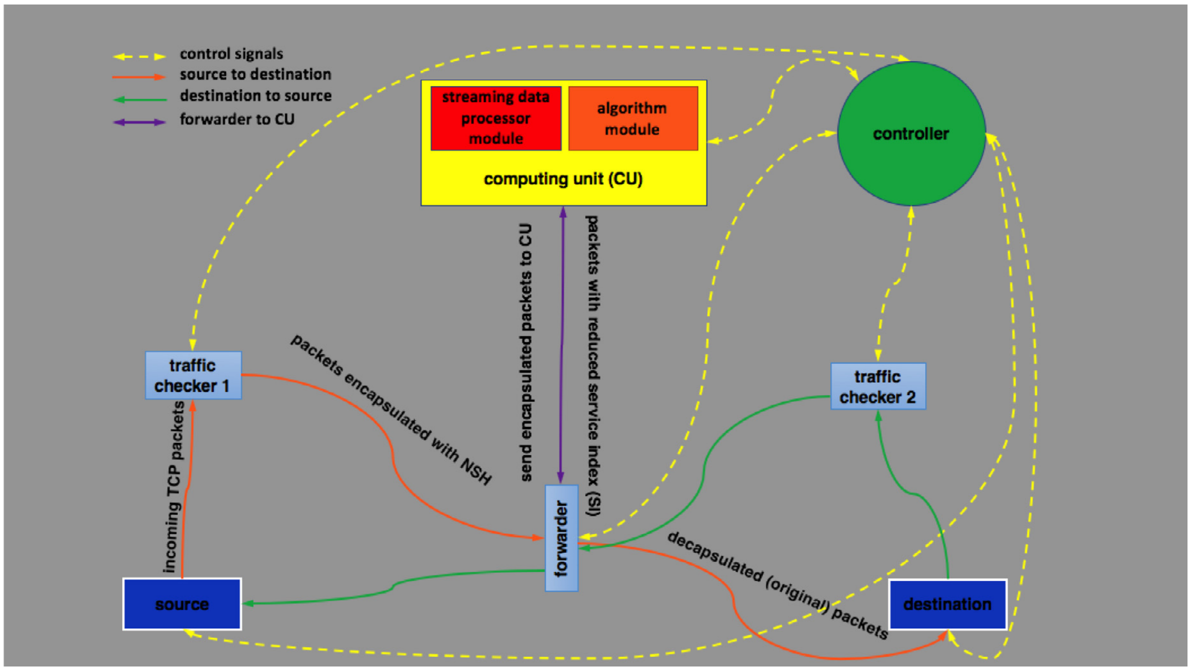


Fig. 2. Preliminary implementation of the Analysis on Wire (AoW) Framework. The incoming TCP data packets enter the AoW framework at the traffic checker from the source, where the packets are encapsulated with NSH headers. With the help of these headers, the data packets traverses their way through the AoW framework forwarded by the intended forwarder to the computing unit, where desired algorithm is executed on the processed payload from the packets. Post this, the packets are sent back to the forwarder, which forwards them to the destination.

2.3. Description of operation

The CU extracts the payload from the encapsulated UDP packets it receives from the Forwarder. The UDP packets contain the original packet, which in our case are Transmission Control Protocol (TCP) packets. From the enclosed TCP packets, the CU extracts the payload and transforms it into a Comma Separated Values (CSV) 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 1) congestion controlled rate; in a 2) single administrative network, which is majorly true for streaming data. We can briefly explain these two terms as follows. Our setup cannot work extremely well without any reasonable acceleration as we believe through Field-programmable gate arrays (FPGAs) [20] and/or Graphics processing units (GPUs) [21] if congestion happens to occur at any of the components in the framework. Congestion in this context implies the rate of data packets entering the framework should be within the maximum bandwidth the components in the framework can successfully receive and forward. If the rate of traffic exceeds the bandwidth of any of the components in the framework, the packets would start dropping. 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.

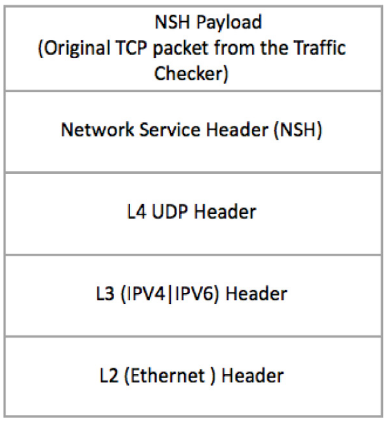

NSH Payload
(Original TCP packet from the Traffic Checker)
Network Service Header (NSH)
L4 UDP Header
L3 (IPV4|IPV6) Header
L2 (Ethernet) Header

Fig. 3. Network Service Header (NSH) UDP Stack. When data packets enter the AoW framework, the traffic checker encapsulates the incoming packets with NSH headers. These headers enforce movement of the data packets creating a computing chain in the AoW framework.

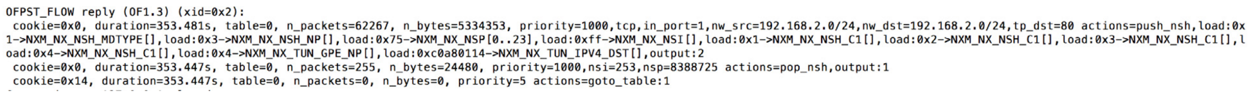
As the AoW framework forces the traffic to go through a constrained chain of modules to be able to do "computing on the wire", it definitely adds overhead. We are looking into ways how FPGAs and/or GPUs can minimize this overhead. The mention of FPGAs and GPUs in any kind of computing environment echoes the idea of acceleration in computation. Although, this is obviously possible with both the devices — we plan on using FPGAs and/or GPUs [22] for network acceleration. FPGA Network Processing Cards [23] are capable of addressing a range of latency-critical applications including: Signals Intelligence, Encryption/Decryption, Network monitoring and security, Deep Packet Inspection, Real-Time image processing and High Frequency Trading. We are already investigating whether the use of such FPGAs can help boost the performance of the AoW framework, by assisting in quick NSH encapsulation and decapsulation in our current work on a hybrid SDN test bed. GPUs and FPGAs, among other new hardwares, are usually employed as co-processors to accelerate query execution. The advantage of these architectures lie in their massive inherent parallelism as well as a different programming model compared to the classical von Neumann CPUs. These hardware architectures can offer the processing capability to distribute the workload among the CPU and other processors, and enable systems to process bigger workloads [24]. It goes without saying that the upcoming age of 100 Gbps+ data rate could not be handled by even powerful multi-core CPU machines alone [24] for big data analytics. We must have accelerated computing devices to deal with the computation and analysis of data streaming at such a high rate into the AoW framework. FPGAs could give us a 57X faster result [25–27] and a 40X speedup could be achieved using GPUs [28]. We expect to be able to substantiate these figures or come up with better numbers soon in our test bed. We also need to look into how to combine both these devices in the AoW framework to optimize performance further.

AoW supports a bidirectional framework as illustrated in Fig. 2. Bidirectional framework implies that CU chain is applicable in both directions and all ports send and receive packets at the same time. For example, at one point, the initial source sends traffic to the destination with computation happening in the CU while packets are still in flight. At the same instance, the destination can become source and start sending traffic towards the current source, which becomes the destination now. Accordingly, we have two Traffic Checkers, one at the source and other at the destination. The Forwarders with their CUs are applicable in both directions.

NSH [18] ensure the successful implementation of our framework. The Traffic Checker encapsulates packets with NSH based on the CU paths and CU chains, we feed to the ODL controller. A CU chain consists of the Traffic Checker, Forwarder and its attached CU. NSH contains CU Path information and a service index (SI) - a numerical value varying from 0 to 255 indicating the number of hops for the packets to go through in the CU path. The Traffic Checker forwards the packets to the first Forwarder after encapsulating the packets with NSH. Based on the CU path, the SI value is decremented at each CU attached to a Forwarder. When the SI value is zero at a Forwarder, this implies that the path traversal by the packet is complete and the Forwarder sends the packet to its destination. In our experiments, the CU chain consists of only a single CU. This chain is passed to the CU Path. In this framework, NSH headers are transported through the UDP protocol which is suitable for stream processing as it is connectionless communication and at the same time works quite well in a congestion controlled environment as this but NSH is essentially protocol agnostic. At the end of the chain, SI is reduced to zero at the CU and sent back to the Forwarder. The Forwarder removes the NSH and sends the originally enclosed TCP packets intact to the destination. The NSH UDP stack is shown in Fig. 3.

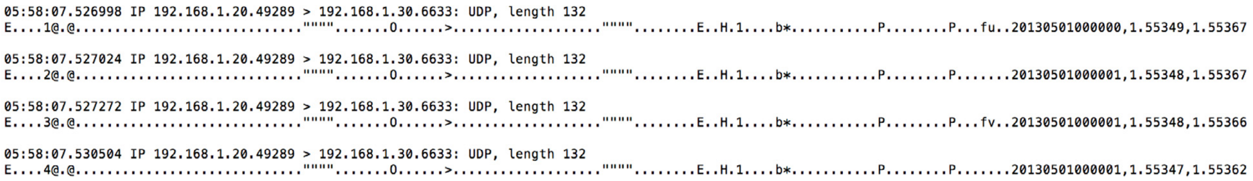
We inject traffic into the network through a TCP packet generator implemented with a Python script. This traffic tries to enter the AoW framework at the Traffic Checker. So, once a data packet is at the Checker, the Checker tries to match the destination and packet type of the data packet with its flow rules. Here, the 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 as shown in Fig. 4. If there is no match, the data packet does not need to enter this framework, but it can go directly to its destination based on custom routing policies. This can be further useful in scenarios where we want data packets with specific attributes to enter our framework for possibly analyzing them or to perform custom computations on them. As per the values in the inserted NSH headers on top of original packet for the desired computation path, the Traffic Checker transmits it to the intended Forwarder. The Forwarder forwards the packets to its attached CU (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 Streaming Data processor module. The output from the data processor module is fed to the Algorithm module, where we run the pattern recognition, detection and prediction algorithms coded as Python scripts. More specifically, in the Data processor 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 to have a 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 have to wait for the data to arrive at the destination data center to be gathered and processed, which would take much longer to give us any necessary insight into the data. Stream processing in the data center, if possible, would yield results much faster, but still not as fast as on the wire. On the other hand, a large number of streams may overwhelm a data center, whereas processing in the network can distribute the load and reduce the data volume as seen in Fig. 1. With the AoW 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 Traffic Checker to insert NSH headers onto the incoming data packets.

In the following section, we demonstrate in a detailed manner the three example algorithms we spoke about earlier to compute on the data streaming into the framework. Our primary focus is on the network design and these examples serve as evidence of what is possible in such a framework. However, we are certain that there exist many meaningful computations in addition to the



```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=353.481s, table=0, n_packets=62267, n_bytes=5334353, priority=1000,tcp,in_port=1,nw_src=192.168.2.0/24,nw_dst=192.168.2.0/24,tp_dst=80 actions=push_nsh,load:0x
1->NXM_NX_NSH_MDTYPE[],load:0x3->NXM_NX_NSH_NP[],load:0x75->NXM_NX_NSP[0..23],load:0xff->NXM_NX_NSI[],load:0x1->NXM_NX_NSH_C1[],load:0x2->NXM_NX_NSH_C1[],load:0x3->NXM_NX_NSH_C1[],l
oad:0x4->NXM_NX_NSH_C1[],load:0x4->NXM_NX_TUN_GPE_NP[],load:0xc0a80114->NXM_NX_TUN_IPV4_DST[],output:2
 cookie=0x0, duration=353.447s, table=0, n_packets=255, n_bytes=24480, priority=1000,nsi=253,nsp=8388725 actions=pop_nsh,output:1
 cookie=0x14, duration=353.447s, table=0, n_packets=0, n_bytes=0, priority=5 actions=goto_table:1
```

Fig. 4. The output for dump-flows at Traffic Checker1. The flow tables for open vSwitches contain rules for driving the traffic through the desired computation path in the AoW framework and is programmed into the Traffic Checker by the Controller. The table use is described as: Table 0 - Transport Ingress; Table 1 - Path Mapper; Table 2 - Next Hop, Table 10 - Transport Egress [17].



```
05:58:07.526998 IP 192.168.1.20.49289 > 192.168.1.30.6633: UDP, length 132
E....1@.@..........................""""".......0......>..................""""".........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@.@..........................""""".......0......>..................""""".........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@.@..........................""""".......0......>..................""""".........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@.@..........................""""".......0......>..................""""".........E..H.1....b*...........P........P.......20130501000001,1.55347,1.55362
```

Fig. 5. Forex data during an intermediate processing at the Computing Unit (CU). At the CU, tcpdump captures the packets and feeds to the Streaming data processor module in the same format as shown here.

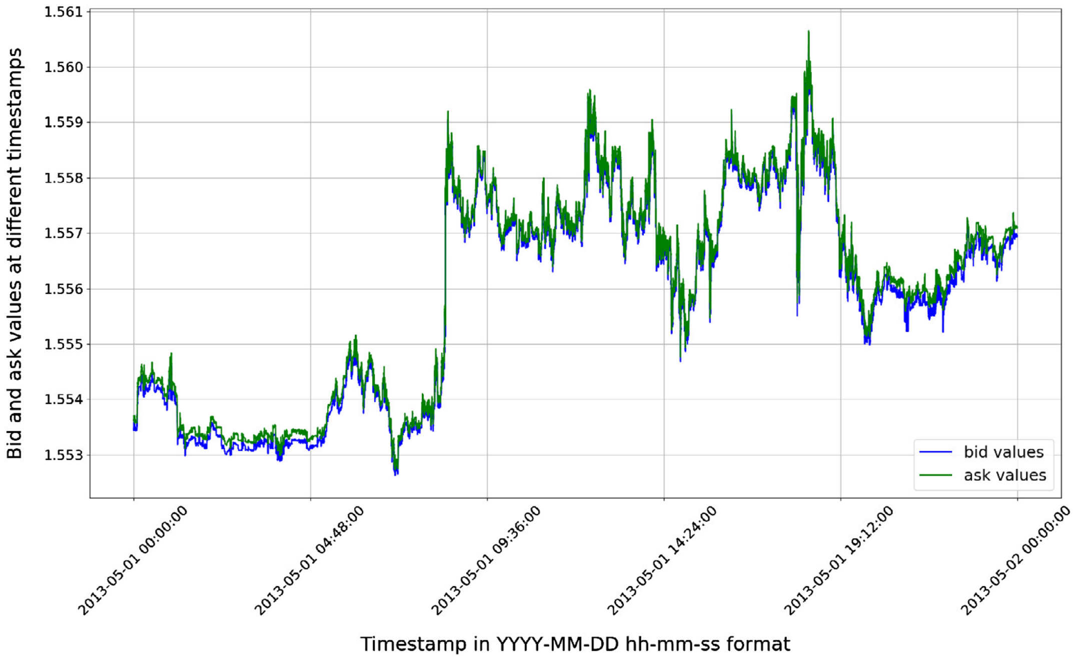


Fig. 6. Forex data visualization. As the payload from the incoming packets is processed at the Data processor module, the visualization algorithm in the Algorithm module renders the trend for the bid and the ask prices for the Great British Pound to US Dollars (GBPUSD) Forex for each second of a day. The *y*-axis show the bid and ask values across the timestamps on the *x*-axis.

examples discussed in this paper, that could be performed in the AoW framework and it is in our future plans to explore a wide range of such computations and come up with a generalization. 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 "on the wire" 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 the Great British Pound to USD (GBPUSD) Forex. An example of the data is shown in Fig. 5 in an intermediate processing state at CU. We can see the timestamp, bid and ask price at the tail of each packet.

We use the Pattern Recognition algorithm [29] to visualize the Forex tick datasets for one day as seen in Fig. 6, 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 [29]. 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, the Algorithm module at the CU keeps predicting the pattern of this bidding based on previously stored similar patterns. Accordingly,

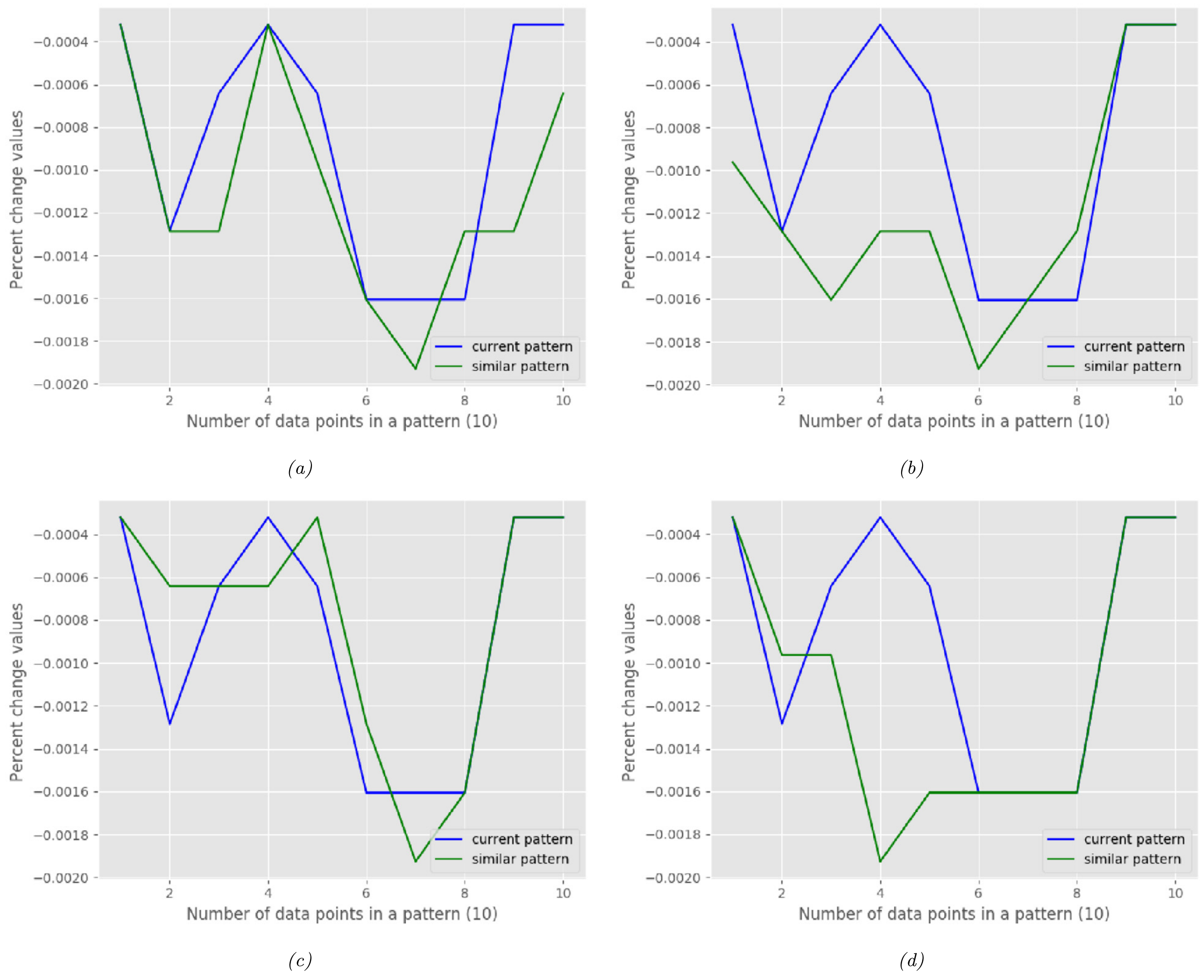


Fig. 7. Pattern with greater than 70 percent similarity. The Algorithm module at the Computing Unit runs a prediction algorithm for every incoming data packet with the last nine data packets against all stored historical patterns. If the similarity calculated is more than 70 percent with the current input pattern, a match is found and the matched patterns are returned. These are some of the matched patterns from the history with the current input pattern. The *y*-axis shows the percent change values for ten continuous input data points on the *x*-axis.

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 a percent change format as per Eq. (1). 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 a 70 percent or more similarity as visible in Fig. 7, 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.

In Fig. 7, the blue line is the current pattern in question and the green line is the matching similar pattern.

$$percentChange = [(currentPoint - startPoint) / startPoint] \times 100.00 \quad (1)$$

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 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 achieve this by using the AoW framework. As soon as the data starts streaming from the websites for a media publisher, it enters the AoW framework and reaches the CU through the Forwarder, after being accepted at the Traffic checker. Since we deploy AoW close to the source, the data is available for computation earlier than it would be at the data centers or cloud and we can start processing the data right away. Also, the data from 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 restricted in an edge computing model.

```
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. 8. Clickstream payload example from one of the web pages. Out of all the web pages streaming data, this is one data packet generated from clicking a clothing (product id: VD55177927) item for one of the users (user id: 8D0.437E-9249-4DDA-BC4F-C1E5409E3A3B).

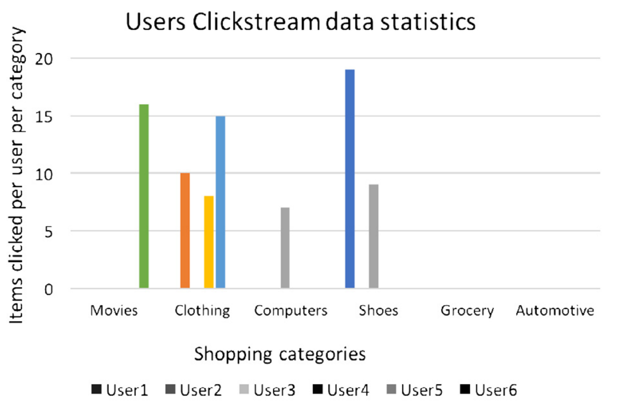


Fig. 9. User Clickstream data statistics. The Clickstream data consists of clicks from multiple users with unique ids on different products as shown in category on the *x*-axis. As this stream of data passes through the Computing Unit, the Algorithm module runs an algorithm on the CSV format data from the Data processor module to calculate the statistics of different user clicks in six different categories as shown on the *y*-axis.

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 [30]. The data payload from a packet of one of the web pages is as indicated in Fig. 8.

This can give a lot of information to publishers, for example — ongoing trend in Sunnyvale or festival related hits in New York according to the number of hits on a web page. 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 web pages.

The Data processor 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. 9.

3.3. *Solar sensors streaming data analysis*

In this section, we analyze multiple solar sensors data to detect any anomaly in their readings to be able to do any feasible repair or maintenance activity on them faster than if we would have waited for the sensor readings to reach the data centers and processing thereafter.

We have twenty three solar sensor readings for a day from the Long Island Solar Farm (LISF) dataset from which we stream readings every second. We analyze this solar sensor data while they are still in transit to the data centers. As soon as the packets from the sensors are available to be sent towards the data centers, the AoW framework pushes these data packets to the Traffic Checker from where it reaches the CU through Forwarder with the help of NSH encapsulation. Since the AoW framework is not deployed far from the streaming source, the CU can analyze the sensor readings much earlier than these readings ever reach the data centers.

What we do at the computing unit is as follows: the Data Processor module at the computing unit casts the payload in the incoming data packets furnished through tcpdump in a CSV format to feed to the Algorithm module. The attached Algorithm module parses the processed payloads to determine which 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 replace it with another sensor earlier than if we would have waited for the readings to reach the data center and take any action past that. This is an idea, which can possibly surpass the Internet of things (IoT) [31] processing being done at the edge and start being done "on the wire".

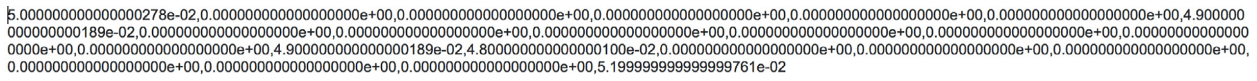
The CSV data format which is fed to the algorithm is shown in Fig. 10. The sensor readings at two different timestamps passing over the network is depicted in Fig. 11.

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 total time taken to send data packets from one end to another through the framework and directly without the framework as in Fig. 12. The second category compares the total time taken for data transfer plus processing of data packets plus performing the computation algorithm on them, through the framework versus directly towards the data center (destination) in which we process and compute on the packets at the destination as shown in Fig. 13.

We observe that up to 1000 packets, the data packet transfer time through the framework versus directly are almost the same. Beyond this rate, there is quite a difference as we understand our framework works best in a congestion-controlled environment which we do not have here; sending packets at a higher rate leads to congestion in the network and hence the delay. 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. We are already investing to handle this congestion by accelerating the network through specialized FPGAs in the AoW framework. Since the current environment is meant mostly for proof of concept, it is not particularly geared for speed. However, there are several ways to achieve higher rates including FPGAs and GPUs for accelerating computation, which we plan to investigate in the near future.

The comparison of data processing time plus algorithm execution varies in the three algorithms from that in their direct path plus 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 data 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/or FPGAs, which is one of our ongoing projects over a hybrid distributed setup. For the other two cases of clickstream data and solar sensors, the computation is of low overhead, the



```
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. 10. Solar sensor payload at the Data Processing unit of the Computing Unit. The readings from the 23 solar sensors stream into the AoW framework in a single data packet and the above figure shows the payload from one of the data packets at a time.

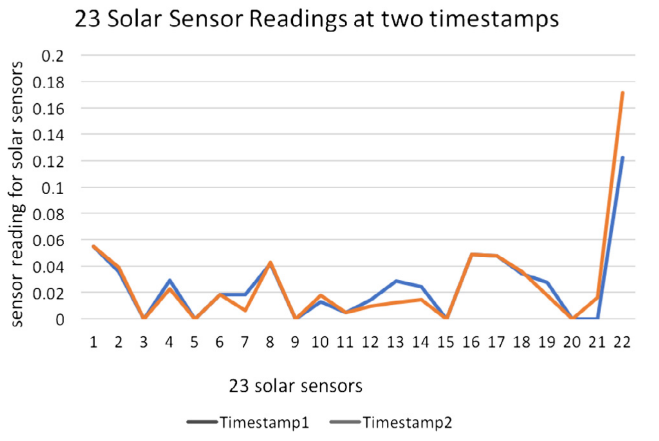


Fig. 11. 23 Solar sensor readings at two timestamps. The data packets (assuming a single data packet contains all 23 solar sensor readings) get processed at the streaming Data processor module into a CSV format which is then used by the Algorithm module to determine the sensors with zero readings. Lower the reading, lesser the sun is encountered at the sensor location. If the sensor reading is zero, this implies, either the sensor is broken or needs maintenance. The *y*-axis shows readings for the 23 solar sensors on the *x*-axis.

only difference comes from the NSH encapsulation and decapsulation at the Traffic Checker and the Forwarder respectively. The NSH encapsulation is something that can be accelerated, and we also plan to examine alternative approaches that may eliminate the need for such headers entirely 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 CU (Data processor module plus Algorithm module), NSH header processing at each unit in the framework like decreasing SI by the CU and removing headers by the Forwarders.

As we see in Fig. 14(a), minimum overhead is 44 s 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.

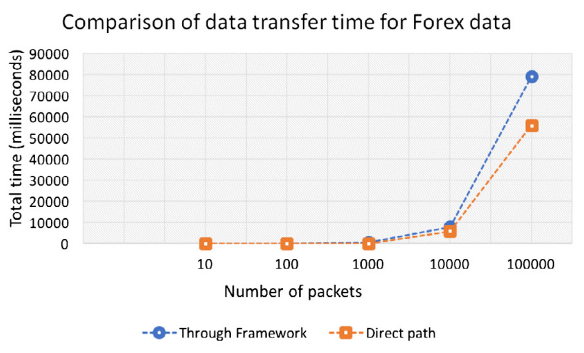
From Fig. 14(b) and (c), we see that we get results at a little delay through our framework than sending the data directly and then doing any kind of computation on it. This is possible because computation has negligible overhead for these cases and since the AoW framework is positioned close to the sending host, we can get results with a very small delay in these cases.

5. Related work

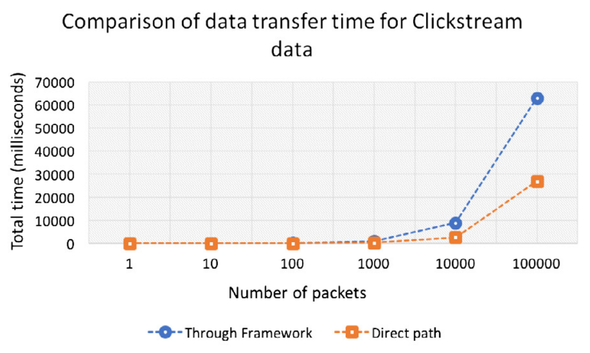
The concept of Active networks (AN) [32,33] was considered in the 90‘s for providing programmable network APIs, even before IETF presented the concept of SDNs in 2004. The idea could not take off that time due to expensive computing power available in network nodes and standardized hardware. AN were proposed to assist with the growing problem of network management [34]. The underlying idea was to embed code in packets which would then program the network nodes. AN architecture was proposed to be a combination of a Node OS and an execution environment. The Active Network Encapsulation Protocol provides this capability analogous to NSH in the SFC architecture where we focus on generic computations on data streams. Unlike AoW, the AN approach was focused at computations for better network management and was envisioned with a different range of applications focusing on smooth functioning of networks such as multicast, QoS, caching like CDNs [34] rather than doing big data analytics or processing on packets flowing through the network. SURFnet used SFC to transcode a 4k video stream [35] with service functions performing video transcoding placed in clouds. This demo used SFC to force the traffic go through a predefined constrained path through clouds located all over the globe (SURFnet OpenStack test bed at Amsterdam, SURFsara HPC cloud at Amsterdam, Okeanos at Greece, Cloud Sigma at Switzerland and Microsoft Azure at Amsterdam). Thus, this is a distributed cloud computing setup. In AoW we use SFC to perform generic computations within the network fabric, as we aim at minimizing latency and an earlier insight into the data still in flight, while at the same time saving bandwidth at the data centers. Vasiliadis et al. [22] describes a GPU-Accelerated Stateful Packet Processing Framework (GASPP) to perform network traffic processing in GPUs. GASPP provides mechanisms for sharing memory context between network interfaces and the GPU to avoid redundant data movement, and for scheduling packets in an efficient way that increases the utilization of the GPU and the shared PCIe bus. The authors suggest using GASPP for computationally intensive and complex network operations such as stateful traffic classification, intrusion detection, and packet encryption. This is a high volume network traffic processing framework which offloads the load from CPU to multi-core GPUs. A recent paper [36] presents the sPIN framework for stream processing in Network Interface Cards (NICs) with the help of specialized data processors. sPIN offers Remote Direct Memory Access (RDMA) functions and thereby enabling tens of gigabytes per second transmission rates at sub-microsecond latencies in modern NICs. In sPIN, kernels do not offload compute-heavy tasks, but only tasks that can be performed on incoming messages and require limited local state. The AoW framework has a different, wider scope from GASPP and sPIN as we aim at performing generic computations on the data flows in the network fabric. However, we will be investigating the applicability of these technologies to accelerate processing in AoW.

6. Potential use cases

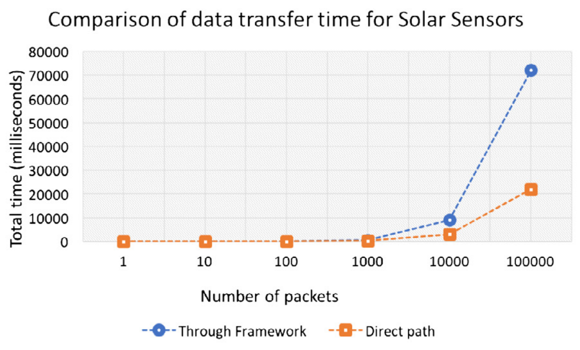
Smart Power grid - The start of the first phase of a sensor deployment program has been announced by the New York Power Authority that will use new technologies to perform on-line monitoring of power plants, substations, and power lines in New York [37]. These sensors aim at proactively predicting potential problems and facilitate reduced unplanned downtime, lower maintenance costs, and minimize potential operational risks. The sensor system will feed data such as temperature, power loads, vibrations, pressure, emissions, and moisture into a hub in near real time which will



(*a*) ***Comparison of data transfer time for Forex data***



(*b*) ***Comparison of data transfer time for clickstream data***



(*c*) ***Comparison of data transfer time for Solar Sensors data***

Fig. 12. Comparison of data transfer time in milliseconds when reaching the destination directly (orange) and while going through the AoW framework (blue). The delay encountered while going through the AoW framework is because of the extra effort needed by the AoW framework to guide the traffic through the desired computation chain by introducing encapsulation on the incoming packets and processing of headers at each hop in the framework. The y-axis shows the time taken in milliseconds for the number of packets on the x-axis.

store it and perform data analytics on it. Currently, the plan is to do the analysis in the compute clouds provided by GE, for which simulation is already being done by GE through their product Predix [38]. We foresee that this problem can be a prime candidate for the AoW framework thereby saving plenty of cloud resources and discounting latency at the same time.

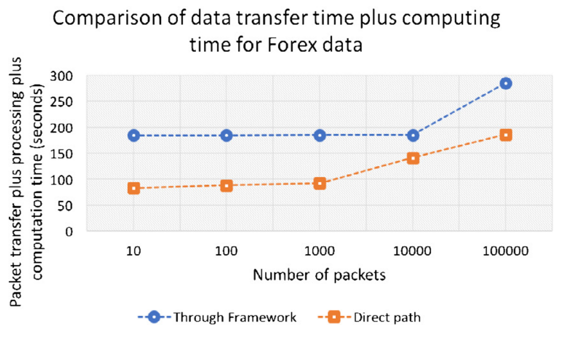
Cybersecurity - There is a major cyber security challenge with today's high-bandwidth networks. Current technologies are already lagging behind in capabilities to analyze the bulk of the network traffic and detect threats in real time [39]. For example, DOE laboratories are currently interconnected through ESnet with multiple 100Gbps links, 400Gbps has been tested, and it is expected that such cutting edge networks will reach terabit speeds within the next few years. Typical firewalls, intrusion detection, and other protection systems cannot handle traffic at such high rates without significant adverse effects on the traffic (bottlenecks) or long delays in detection (e.g., only support for offline analysis). In order to deal with this — scientific "Big Data" streams are typically considered trustworthy and set up to bypass protection systems, offering adversaries unique and unguarded access to both DOE networks and laboratory computing systems, including high performance computing systems. These streams are often unencrypted pathways into government networks. Moreover, packets are often of uniform length and structure giving a sophisticated and determined adversary (e.g., state sponsored) opportunity to deploy dedicated attacks (e.g., imposter packets) to exploit these features. We believe that the AoW framework can offer immense help here dealing with modern networks.

Elephant flows detection (high bandwidth, high volume, long duration) - Research-based networks as ESNet, GEANT generate extremely large datasets which are moved to the local data centers for segregating elephant flows from mice flows. Mice flows are bursty and latency sensitive applications, whereas elephant flows involve big data file transfers that tend to have long-lived flows where transfer throughput is more important than latency. The segregation at the data center is computationally expensive and waits for the entire dataset to be arrived, elephant flows can fill network buffers, causing queuing delays and packet drops. Chhabra et al. [40] describes a machine learning approach to identify elephant flows in data centers. The existing methods [41,40] suggest to do this segregation in data centers. But we suggest this to be done on streaming data flows on the wire while packets are still on their way towards the data centers through the AoW framework. The Traffic Checker can redirect the desired flows into the AoW framework, on which the CU can run machine learning algorithms for separating the flows.

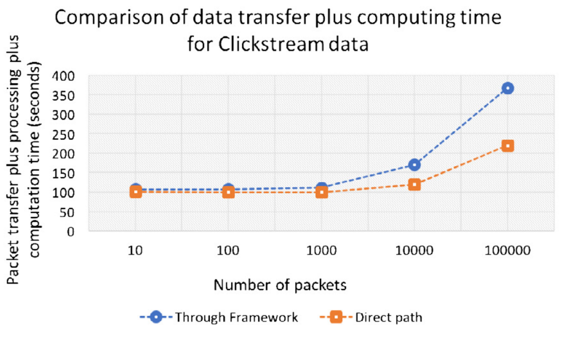
Weather prediction - Insolation or solar irradiance is essential for numerical weather prediction and understanding seasons and climate change [42]. Xu et al. [43] proposes a local vector autoregressive framework with ridge regularization to forecast irradiance with the help of low-cost irradiance sensors without explicitly determining the wind field or cloud movement. We propose that this algorithm can be run in the CU using the AoW framework, thereby providing an almost instantaneous prediction in short term and much lesser storage in the longer run by saving only the analysis results from the massive sensor readings in the data centers.

7. Trade-off and limitations of the current implementation

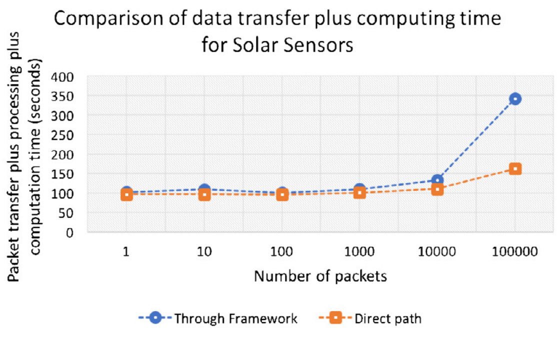
The trade-off for computing "on the wire" is the delay in the AoW framework for steering the traffic flows through a constrained path. But the latency introduced can vary from being



(a) Comparison of data transfer plus computing time for Forex data



(b) Comparison of data transfer plus computing time for clickstream data



(c) Comparison of data transfer plus computing time for solar sensors data

Fig. 13. Comparison of total time taken for data transfer plus processing of data packets plus performing the computation algorithm on them, through the framework versus directly towards the data center (destination) in which we process and compute on the packets at the destination. We do not need to necessarily compare these two cases, as our focus here is the ability to do computation on real-time streaming data while it is already on flight and also because there is no fixed set of standards for the timeline when computing happens at the data center. Our intention here is just to draw a comparison between best case computation at the data center with the worst case possible in the AoW framework. The y-axis shows the time taken in seconds for the number of packets on the x-axis.

negligible in case of non-compute intensive scenarios to significant for compute heavy algorithms at the CU. We are looking into ways to minimize this delay through advanced hardware as FPGAs and/or GPUs.

In the remainder of this section, we point out certain limitations and challenges encountered while testing our initial AoW implementation.

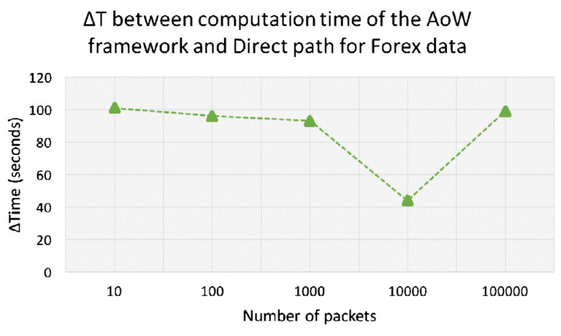
Data fragmentation - Throughout the paper, we assumed that data packets can contain a full set of information to be processed by the CU (e.g. a payload from a solar sensor shown in Fig. 10). Although this is a limitation of the initial implementation for simplicity, in the general case, we expect that a payload may span multiple packets and that a packet may contain fragments of more than one payload. This requires on-the-fly reconstruction of a data set before it is fed to the CU. We plan to add this capability in the next development iteration.

Fault tolerance - We have not handled scenarios where one or more framework components fail or stop responding. A component failure leads inexorably to packets being lost which can be disruptive to the data stream. It is our intention to closely monitor the operation of the framework and have backup components to fail over and alleviate such problems.

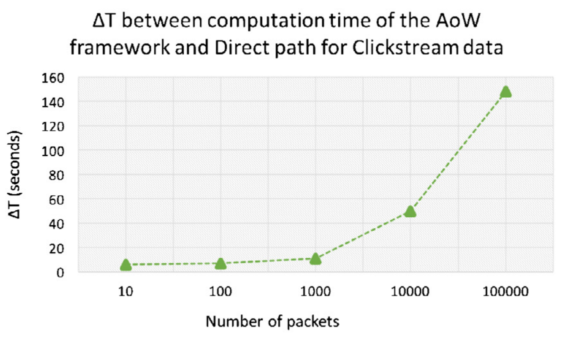
Security - The integrity and security of the framework needs to be carefully examined. The CU performs operations on raw data and should be protected from unauthorized codes that would manipulate data in unforeseen ways. A Traffic Checker or a Forwarder can be compromised, directly or through the SDN controller to mislabel and misdirect traffic. As we proceed with the research and development of AoW, we plan to keep a close look at such issues.

8. Conclusion and future work

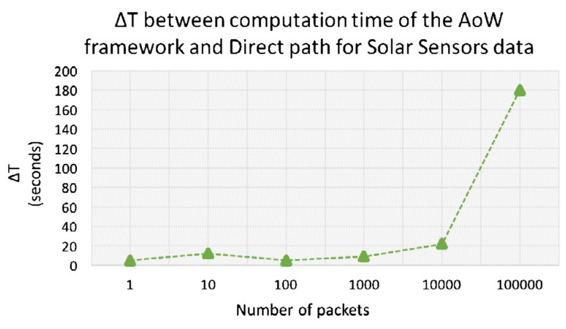
We have presented the design, implementation, and evaluation of the AoW framework - a SDN framework capable of performing computations and analysis on streaming data in the network fabric. AoW can help in saving resources at data centers for the current as well as the impending Big Data age, earlier decision-making, and faster results. We have demonstrated three use cases on the AoW framework — analyzing Forex data stream to plan a better future trade investment, targeted advertising through clickstream data analysis by media publishers, an example of IoT through analyzing solar sensor readings of LISF to detect device failures earlier. These examples indicate a major opportunity to perform computations on the wire. The clickstream and solar sensor data analysis add minimal overhead as they involve simple computations. In the Forex data case, we observe a significant latency as in this case we compute patterns from the streaming data, store them and predict similar patterns for future data by comparing with stored patterns. We argue that in order to handle compute-intensive scenarios such as this, we would need compute as well as network acceleration in the AoW framework. We are already looking into both accelerating the network operations (e.g., encapsulation/decapsulation) and the computation by employing FPGAs and/or GPUs, to alleviate the additional network overhead imposed by an SFC architecture and minimize the execution time of compute intensive algorithms. While in this paper we focused on demonstrating the concept of the AoW framework using a single computing unit and an all-virtual setup for proof-of-concept, we are moving to a hybrid test bed with SDN switches and servers enhanced with FPGAs and/or GPUs, to study performance and scalability issues. We further plan to investigate fault-tolerance issues, so that the AoW can "do no



(a) ΔT between computation time of the AoW framework and Direct path for Forex data



(b) ΔT between computation time of the AoW framework and Direct path for Clickstream data



(c) ΔT between computation time of the AoW framework and Direct path for Solar sensors data

Fig. 14. Comparison of ΔT when data packets reach the destination directly vs when they go through the AoW framework. 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 CU (Data processor module plus Algorithm module), NSH header processing at each unit in the framework like decreasing SI by the CU and removing headers by the Forwarders. The y-axis shows the delay in seconds for the number of packets on the x-axis.

harm''. Any kind of failure in the SFC components should be detected and traffic forwarding should fall back to default parameters (e.g., directly forwarded to destination).

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 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 the AoW framework.

References

[1] CERN. CERN Data Centre passes the 200 -petabyte milestone, 2017. https://phys.org/news/2017-07-cern-centre-petabyte-milestone.html/.

[2] Amazon. Streaming data, 2017. https://aws.amazon.com/streaming-data/.

[3] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81. http://dx.doi.org/10.1109/MC.2016.145.

[4] S. Jak Jones, Edge computing: The cloud, the fog and the edge, 2018. https://www.solid-run.com/edge-computing-cloud-fog-edge/.

[5] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39. http://dx.doi.org/10.1109/MC.2017.9.

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

[7] M. Liyanage, C. Chang, S.N. Srirama, mepaas: Mobile-embedded platform as a service for distributing fog computing to edge nodes, in: 2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT, 2016, pp. 73–80. http://dx.doi.org/10.1109/PDCAT.2016.030.

[8] D. Zhang, H. Ning, K.S. Xu, F. Lin, L.T. Yang, Internet of things, J. UCS 18 (2012) 1069–1071.

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

[10] B.A.A. Nunes, M. Mendonca, X.N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634. http://dx.doi.org/10.1109/SURV.2014.012214.00180.

[11] K. Kirkpatrick, Software-defined networking, Commun. ACM 56 (9) (2013) 16–19. http://dx.doi.org/10.1145/2500468.2500473. URL http://doi.acm.org/10.1145/2500468.2500473.

[12] D. Katramatos, M. Yue, S. Yoo, K.K. van Dam, J. Xu, J. Zhang, Streaming data analysis on the wire, in: 2016 New York Scientific Data Summit, NYSDS, 2016, pp. 1–7. http://dx.doi.org/10.1109/NYSDS.2016.7747816.

[13] HashiCorp. Vagrant, 2013. https://www.vagrantup.com/intro/index.html.

[14] Docker. Docker, 2017. https://www.docker.com/what-docker.

[15] OpenDayLight. Opendaylight, 2017. https://www.opendaylight.org/.

[16] SFC. Opendaylight sfc demo, 2017. https://github.com/opendaylight/sfc.

[17] OVS. Open vswitch, 2017. http://openvswitch.org/.

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

[19] S. Kumar, L. Kreeger, S. Majee, W. Haeffner, R. Manur, D. Melman, 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), 2017. URL https://tools.ietf.org/html/draft-kumar-sfc-nsh-udp-transport-03.

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

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

[22] G. Vasiliadis, L. Koromilas, M. Polychronakis, S. Ioannidis, Gaspp: A gpu-accelerated stateful packet processing framework, in: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, USENIX ATC'14,

USENIX Association, Berkeley, CA, USA, 2014, pp. 321–332 URL http://dl.acm.org/citation.cfm?id=2643634.2643668.
[23] Nallatech. FPGA Network Processing Cards, 2018. http://www.nallatech.com/solutions/fpga-network-processing/.
[24] M. Saecker, V. Markl, Big data analytics on modern hardware architectures: A technology survey. Business Intelligence: Second European Summer School, eBISS 2012, Brussels, Belgium, July 15–21, 2012, Tutorial Lectures, 138, 125, 2013.
[25] M. Véstias, H. Neto, Trends of cpu, gpu and fpga for high-performance computing, in: 2014 24th International Conference on Field Programmable Logic and Applications, FPL, 2014, pp. 1–6. http://dx.doi.org/10.1109/FPL.2014.6927483.
[26] J. Fowers, G. Brown, P. Cooke, G. Stitt, A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA'12, ACM, New York, NY, USA, 2012, pp. 47–56. http://dx.doi.org/10.1145/2145694.2145704. URL http://doi.acm.org/10.1145/2145694.2145704.
[27] D. Singh, C.K. Reddy, A survey on platforms for big data analytics, J. Big Data 2 (1) (2014) 8. http://dx.doi.org/10.1186/s40537-014-0008-6. URL https://doi.org/10.1186/s40537-014-0008-6.
[28] N.K. Govindaraju, B. Lloyd, W. Wang, M. Lin, D. Manocha, Fast computation of database operations using graphics processors, in: ACM SIGGRAPH 2005 Courses, SIGGRAPH'05, ACM, New York, NY, USA, 2005. http://dx.doi.org/10.1145/1198555.1198787. URL http://doi.acm.org/10.1145/1198555.1198787.
[29] Harrison. Pattern recognition algorithm on forex tick datasets, 2013. https://pythonprogramming.net/machine-learning-pattern-recognition-algorithmic-forex-stock-trading.
[30] Clickstream. Clickstreamdata, 2017. https://opendata.stackexchange.com/questions/1779/clickstream-sample-dataset/.
[31] IoT. Internet of things, 2017. https://www.forbes.com/sites/bernardmarr/2017/05/05/internet-of-things-and-predictive-maintenance-transform-the-service-industry/.
[32] K.L. Calvert, S. Bhattacharjee, E. Zegura, J. Sterbenz, Directions in active networks, IEEE Commun. Mag. 36 (10) (1998) 72–78. http://dx.doi.org/10.1109/35.722139.
[33] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, G.J. Minden, A survey of active network research, IEEE Commun. Mag. 35 (1) (1997) 80–86. http://dx.doi.org/10.1109/35.568214.
[34] K. Psounis, Active networks: Applications, security, safety, and architectures, IEEE Commun. Surv. 2 (1) (1999) 2–16. http://dx.doi.org/10.1109/COMST.1999.5340509.
[35] R. van der Pol, Experiences with OpenDaylight Service Function Chaining (SFC). Presentations, SURFnet, 2016. URL https://kirk.rvdp.org/presentations/FOSDEM-2016-rvdp-SFC.pdf.
[36] T. Hoefler, S. Di Girolamo, K. Taranov, R.E. Grant, R. Brightwell, spin: High-performance streaming processing in the network, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC'17, ACM, New York, NY, USA, 2017, pp. 1–59. http://dx.doi.org/10.1145/3126908.3126970. URL http://doi.acm.org/10.1145/3126908.3126970.
[37] N. Ge, Power grid monitoring sensor network launched in New York, 2018. http://www.smart2zero.com/news/power-grid-monitoring-sensor-network-launched-new-york/.
[38] GE. Predix, 2018. https://www.ge.com/digital/predix-platform-foundation-digital-industrial-applications/.
[39] M. DePhillips, D. Katramatos, S. Bhattacharyya, Making a case for high-bandwidth monitoring - a use case for analysis on the wire, in: 2017 New York Scientific Data Summit, NYSDS, 2017, pp. 1–6. http://dx.doi.org/10.1109/NYSDS.2017.8085037.
[40] Chhabra, Classifying Elephant and Mice Flows in High-Speed Scientific Networks, 2017. https://scinet.supercomputing.org/workshop/sites/default/files/Chabbra-classifying_flows.pdf/.
[41] W. Cheng, R. Ren, W. Jiang, K. Qian, T. Zhang, R. Shu, Isolating mice and elephant in data centers, 2016.
[42] Wiki. Solar irradiance, 2016. https://en.wikipedia.org/wiki/Solar_irradiance.
[43] J. Xu, S. Yoo, J. Heiser, P. Kalb, Sensor network based solar forecasting using a local vector autoregressive ridge framework, in: SAC, 2016.

Mrs. Shilpi Bhattacharyya is a doctoral student in the computer science department at Stony Brook University. She is co-advised by Dr. Dimitrios Katramatos (Brookhaven National Laboratory) and Dr. Barbara Chapman (Stony Brook University). Mrs. Bhattacharyya is interested in High performance computing in networks, Software defined networking, Machine learning and Big data. Before embarking on graduate school, Mrs. Bhattacharyya has worked in the software industry for almost 6 years with Oracle Server Technologies, Samsung Research Institute and Sapient Nitro.

Dr. Dimtrios Katramatos is a technology architect at Brookhaven National Laboratory. His areas of interest include, but not limited to High performance networking, Intelligent networking services and integration of such services into applications, Software Defined Networking, Network resource Scheduling and co-scheduling with storage and compute resources, Network Virtualization. He holds a Ph.D. from University of Virginia, Charlottesville, VA. and a Masters from Kent State University, Kent, OH.

Dr. Shinjae Yoo is computational scientist of Computational Science Initiative at Brookhaven National Laboratory and an adjunct assistant professor of Institute of Advanced Computational Science at Stony Brook University. He received his Ph.D. and Masters degree from Carnegie Mellon University. His Masters degree was received at Seoul National University, Korea. He received Bachelor's degree from Soong-sil University, Korea.