

End-to-End Routing Algorithm Based on Max-Flow Min-Cut in SDN Controllers

Nada Alzaben*, Daniel W. Engels**

* Dept. of Computer Science, College of Computer Science and Information, Princess Nourah Bint Abdulrahman University, Riyadh, Saudi Arabia

**AT&T Center for Virtualization, Southern Methodist University, Dallas, USA

nialzaben@pnu.edu.sa, Daniel.w.engels@gmail.com

Abstract— In this paper, we present seven novel Max-Flow Min-Cut (MFMC) based algorithms that solve the flow routing problem in the Software Defined Network Controller. Our algorithms identify potential traffic bottlenecks using the MFMC cut, and they avoid the bottlenecks during flow route construction through their choice of the cut edge added to the route. Our algorithms utilize either a random edge selection, a shortest path edge selection, or an edge capacity-based selection from the set of cut edges. Our simulation results show improvement in the network performance when using MFMC and shortest path edge selection compared to simple shortest path first algorithms, such that the mean wait time is reduced by 15.1%, the mean slowdown is reduced by 5.1%, reducing the maximum completion time by 9.6%, and increasing the mean throughput by 18.3%. Therefore, by explicitly considering congestion in routing decisions, better network performance is achieved.

Keyword— Heuristic algorithm, Peer-to-peer computing, Routing, Software defined networking.

I. INTRODUCTION

ROUTING in computer networks is a fundamental process for finding a communication path that connects two end points in the network for data transfer. The routing process is recognized as an essential factor in solving the network performance problem. In traditional networks the two main functionality of routing is coupled in the router, however, in the new network layer paradigm, the Software Defined Networking (SDN), the network functionality is decoupled in two separate planes of operation: the control plane and data plane [1]-[4].

Benefiting from the SDN global view, the control plane

runs the routing algorithms to find path for each flow in the network. Once the path is defined for the flow, the forwarding plane uses that path to deliver the data from one end to another through network switches [5].

With the increase use of Virtual Private Networks (VPNs), secured communications with Transport Layer Security (TLS), along with the high demand of sending video files, which consume the network bandwidth and dominate the network traffic, have highlighted the importance of flow routing problem. In this type of traffic, they demand that all packets follow the same path from source to destination. Using Software Defined Networks (SDNs) enables end-to-end routing of flows through its global view which result in optimizing a given objective function such as maximizing the mean throughput [6]-[9].

The end-to-end flow routing problem is defined as follows. Given a network graph $G(V,E)$, such that each edge $e \in E$ is with a capacity c , where $c \in \mathbb{Z}^+$, meaning only c flows may be sent at a time on that edge. Having a set of flows F where each flow $f \in F$ has a given number of packets and a release time R_t , non-preemptively route all flows in F on G to optimize the objective function.

The Shortest Path First algorithms (*SPF*) are most commonly used in end-to-end flow routing [10][11], however, due to their lack of explicitly considering congestion when routing causes network performance degradation in a high load traffic. Therefore, we present our Max-Flow Min-Cut based algorithm which is capable of explicitly considering congestion in routing decisions. The *SPF* algorithms are based on either statically or dynamically calculate the path with shortest distance or minimum number of hops. The static shortest path is a topological shortest path that ignores path capacity availability. Moreover, the dynamic shortest path is more logical in considering the edge capacity availability at the time of flow released before assigning a flow on it [12].

In a network with high traffic flow demand which place these flows over edges with limited capacity, such that the flow demand exceeds the edge capacity, consequently causes congestion in the network, i.e., bottleneck arise. In the Static Shortest Path (SSP) algorithm, the flows between any two ends will always take the short path regardless of available capacity, while for the Dynamic Shortest Path (*DSP*) algorithm, the algorithm finds the shortest available path

A manuscript received Jan. 26, 2021. A follow-up on the invited journal to the accepted and presented paper entitled “End-to-End Routing in SDN Controllers Using Max-Flow Min-Cut Route Selection Algorithm” of the 23rd International Conference on Advanced Communication Technology (ICACT2021).

Nada Alzaben is a PhD student in Southern Methodist University and a lecturer in the Department of Computer Science at Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. (corresponding author, phone:001-214-768-3050, e-mail: nialzaben@pnu.edu.sa).

Daniel W. Engels, is with AT&T Center for Virtualization, Southern Methodist University, Dallas, TX 75205 USA (e-mail: Daniel.w.engels@gmail.com).

which considers congestion implicitly during the flow release time. However, flows placed on these kinds of paths in a high traffic flow causes bottleneck problem which means that these flows are most likely going to be sharing same segments of the path in the network. This indicates the need for a routing algorithm that considers congestion explicitly to minimize this problem.

The Max-Flow Min-Cut (*MFMC*) algorithm, with a polynomial time complexity, is capable of identifying the minimum set of edges in the network; such that by their removal the source node is disconnected from the destination [13]-[15]. We present a suite of algorithms (*MC*) that utilize the *MFMC* algorithm in partitioning the network into two sub-networks, where the first one has the source whereas the other has the destination, by explicitly identifying the bottleneck edges that is represented by the minimum number of edges which hold the maximum possible flow across the cut. The *MC* algorithms runs the *MFMC* algorithm recursively and chooses one edge from the cut set to be part of the path in each run which consequently construct a path that connects the source node to the destination. By repeating the process of slicing the network using *MFMC* algorithm, we are explicitly identifying the bottleneck edges in the network through path construction process.

We evaluate all of our variant *MC* algorithms in our simulation by comparing the result of our measured objective functions to the results obtained by variant shortest path algorithms. These simulations evaluate our algorithms under different flow size distributions and under high load networks. The Shortest Path algorithms we evaluated are: shortest path with minimum hops, shortest path with minimum distance, and the shortest path based on the least congested links. In our evaluations, we adopted two types of each *SP* algorithm, either a static algorithm or dynamic which consider edge availability when routing. The *SP* variant algorithms are: Dynamic Shortest Path with Minimum Hop (*DSP-mh*) and Static Shortest Path with Minimum Hop (*SSP-mh*), Dynamic Shortest Path with Minimum Distance *DSP-md* and Static Shortest Path with Minimum Distance *SSP-md*, and the invert version of that which is the Shortest Path based on the least congested links, *DSP-inv* and *SSP-inv*.

The end-to-end routing problem is simulated as a non-preemptive routing with no clairvoyant, where the flow size is only known once released. Moreover, different variations of the problem were as simulated based on network topology (Barabasi Albert, Mesh), release time (identical/arbitrary), flow size distribution (Pareto/Gaussian). Our results showed improvement in minimizing the mean wait time by 15.1%, the mean slowdown by 5.1%, the maximum completion time (makespan) by 9.6%, and maximizing the mean throughput by 18.3%. These results highlight the need to consider other algorithms other than *SP*.

Our Max-Flow Min-Cut based algorithm (*MFMC*) proved to improve our evaluated objective functions in the evaluated networks. Therefore, we conclude that by explicitly considering network congestion in routing decisions yield better performance.

The remainder of this paper is organized as follows. We cover the relevant related work in Section II. The Max-Flow Min-Cut routing algorithm is presented in Section III. In Section IV we introduce the simulation environment. In

Section V we present the simulation results and analysis of our algorithm. We draw a summary and conclusions in Section VI.

II. RELATED WORK

End-to-end routing algorithms in the traditional network routers base picking the next hop on their routing tables. Different algorithms for end-to-end routing were designed based on various criteria such as: bandwidth capacity and delay propagation [16][17], minimize end-to-end delay [18][19], minimize energy consumption [20]-[22], and minimize congestion [23]. All of which work on improving the network performance through end-to-end routing algorithms.

One of the oldest routing algorithms is the Shortest Path algorithm *SP* which was designed by Richard Bellman and Lester Ford as the Bellman-Ford algorithm in 1958 [24] and Edsger W. Dijkstra as the Dijkstra algorithm in 1956 [25]. Many network routing algorithms have been developed based on finding the shortest path as part of their algorithm, such as the Widest Shortest Path (*WSP*) [26] which finds all short paths and picks the path with the widest bandwidth. Moreover, the Shortest-Widest Path [27] is the opposite of the (*WSP*), where all edges less than the required bandwidth capacity are removed then the shortest path is selected using Dijkstra's algorithm.

There are two variations of any shortest path algorithm based on considering the network link state or not which is referred to as the static shortest path or the dynamic shortest path algorithms. The Static Shortest Path (*SSP*) calculates the path using the topological state of the network which ignores the links availability; however, the Dynamic Shortest Path algorithms consider the link state in their calculations. According to Chang H. S. et al. [28], the static algorithm minimizes the overhead of requesting updated link states, however, the dynamic approaches are far superior than static approaches.

In an arbitrary network, routing a set of arbitrary flows over it demands applying routing algorithms which are capable of achieving the goal of utilizing the network bandwidth. Routing algorithms usually are either based on using static link cost or dynamic link cost. For the static link state algorithms, for instance the Minimum Hop Routing Algorithm (*MHR*) [29][30] and Static Shortest Path Algorithm (*SSP*) [31], are finding short paths as on the minimum number of hops to reach the destination or as the shortest distance, however, they do not consider bottleneck congestion in their calculations.

In Software Defined Networks, these types of algorithms are preferable in case minimizing the update messages overhead of the link states between the controller and the network switches is under concern. However, these algorithms congest the network by neglecting the fact of link availability while routing through sending flows with same source and destination over the same path. Akin and Korkmaz [32] listed some of the dynamic shortest path algorithms which are based on considering link bandwidth availability, dynamic link cost algorithms, such as Dynamic Shortest Path (*DSP*). Even though the dynamic link cost algorithms would cause overhead communications by

sending link state update messages with the network switches in the SDN network, with known flows these algorithms utilize network bandwidth and balance load. In the online flow setting where the future flows are not known, their performance is not guaranteed. Our work is an extension of the line of research in the end-to-end routing problem with known flows in SDN controlled networks where we compare our proposed algorithm with the *DSP* and *MHR* algorithms.

In a huge network such as the World Wide Web, flows tend to be heavier tailed with an approximately ($\alpha=1$) [33]. Self-similarity traffic tends to be heavy tailed distribution file size, proved by Park K. et al. [34]. Consequently, that self-similarity effects the network performance. The Pareto flow size distribution is a one common heavy tailed distribution and for that we adopt it when we generate our flows in the simulation to evaluate our novel *MFMC* algorithm.

The network layer fragments large data files before transmitting them to fit the packet frame which does not exceed the Maximum Transfer Unit (MTU). The larger data size the more packet frames created to transfer that data. Previous studies shown that the packet size effect the network performance, where the size of a packet can degrade the throughput once it exceeds its dedicated packet size [35]. Since the space allocation for that packet to be transferred may be doubled in size which waste an unused space allocation. For instance, for a packet frame size of 10 and a flow size of 12 the MTU will cause the flow to be fragmented in two frames where the second one has 8 unused spaces, which doesn't utilize network resources. According to Lin et al. [36] sending large files over the network while maintaining high quality of service, one must select an intelligent routing and scheduling algorithms for fast transfer and better network utilization.

The shortest path algorithm has been widely adopted with both traditional networks and SDN controlled networks. Since it is widely applied in many routing algorithms, we choose to compare our algorithm performance to it.

III. MAX-FLOW MIN-CUT ROUTING ALGORITHM

In the 1950s, Ford Fulkerson and Elias-Feinstein-Shannon [37] developed the Max Flow Min Cut (*MFMC*) theory. Researchers since then have worked on a lot of algebraic topology versions made of *MFMC* [38][39]. The *MFMC* algorithm runs in polynomial time with a complexity measured by the number of edges E multiplied by the maximum number of flows f^* , represented by ($O(E f^*)$). To find the min cut edges in a graph of 50 nodes, assume each node sends out 10 flows, it is expected to consume 122 ms on a Microchip PIC at 5MIPS, while in a graph of 1000 nodes it is expected to consume 49.95 ms on Intel Core i7 with 100,000 MIPS [40].

Routing flows in a network is performed in two folds: choosing which flow to route from the flow set and then find a path to route that flow over it. Picking a flow from the flow set in our simulation is based on picking the shortest flow first to process and in case more than one flow have the same flow size then we pick one of them randomly. Considering consistency in our algorithm's evaluation, we use the same

flow pick order across the evaluated algorithms. After picking a flow, the next process step is defining the path for that flow where we use our novel Max-Flow Min-Cut (*MFMC*) algorithm [41][42].

A general description of our *MFMC* algorithm is defined in Algorithm 1. Our algorithm is based on a recursive process of slicing the network graph by removing the edges in the cut set. The edges in the cut set represent the minimum number of edges which with their removal the graph gets disconnected. By picking one edge from the cut set in each loop we construct the path. We base our edge selection on one of these categories: random selection, shortest path selection, and capacity-based selection.

In an arbitrary network, to find a path that connects between two nodes (s, d), we use our Max-Flow Min-Cut routing algorithm as defined in Algorithm 1. Given $\{G\}$ as an undirected graph and a flow f_i to be routed over G , furthermore, knowing that the set **Cut** holds the minimum number of edges with the maximum flow obtained by following the min cut algorithm, the list **path** holds the selected edges from each cut set which defines the flow path from the source node s to the destination node d for the flow f_i .

Algorithm 1 General Min-Cut Routing Algorithm

```

1: find a path in  $G$  from source  $s$  to destination  $d$ .
2: Input: undirected graph  $G$  and a flow  $f_i=(s,d)$ 
3: Output: a path in  $G$  for  $f_i$  that connects  $s$  to  $d$ .
4: function: FIND PATH IN GRAPH ( $G,f_i,path$ )
5:   if  $\{Connectivity(f_s,f_d)\} = False$  then
6:     return  $path = Null$ 
7:   else
8:      $Cut = \text{get minimum edge cut between } (s,d)$ 
9:      $\text{Partition } G \text{ into } G' \text{ and } G'' \text{ by removing edges in}$ 
       $\text{cut from } G$ .
10:     $\text{pick edge } e=(e_1,e_2) \text{ from cut.}$ 
11:     $\text{append } e \text{ to path list}$ 
12:    if  $(f_s,f_d) == e$  then
13:      return  $path$ 
14:    else
15:       $\text{set new } f_s = e_2 \text{ and new } f_d = e_1$ 
16:       $\text{set } f'=(f_s, \text{new } f_d) \text{ and } f''=(\text{new } f_s, f_d)$ 
17:      if  $G' > 1$  then
18:         $\text{FIND PATH IN GRAPH } (G',f',path)$ 
19:      if  $G'' > 1$  then
20:         $\text{FIND PATH IN GRAPH } (G'',f'',path)$ 
    
```

In our recursive process of cutting the network edges, during each step we result in having a set of edges each one could be a possible edge within the path. Our suit of algorithms *MFMC* have variant variations based on the edge selection criteria used for each algorithm. In this paper we present seven different variations as stated in Table I.

In our variant variations of *MFMC*, we are constructing different paths based on the edge selection criteria adopted by the algorithm. In random based selection algorithms, *MC1* and *MC2*, the edge selection criteria from the cut set are based on randomness. However, algorithm *MC1* does not consider cycles within the process and remove cycles after constructing the path, while *MC2* removes any edge that may cause cycles from the candidate edges in the cut set.

In shortest path selection algorithm, the *MC3*, the edge that

is selected from the cut set is the edge that leads to an available shortest path. The difference between a path constructed by this algorithm and a DSP-md is that MC3 explicitly consider capacity availability when routing which reduce network congestion caused by high traffic demand.

Table I
Edge selection criteria

Algorithm	Edge selection criteria
MC1	the edge is selected randomly and then once path is constructed any cycles in the path will be removed.
MC2	any edge in the cut edge list causing a cycle will be removed then the edge is selected randomly.
MC3	the edge selection is based on being part of possible available shortest path based on minimum hop.
MC4	the edge with minimum available capacity will be selected
MC5	the edge with maximum available capacity will be selected
MC6	the edge with minimum capacity will be selected
MC7	the edge with maximum capacity will be selected

The third category in our edge selection-based criteria is capacity-based selection. We have four variant algorithms based on this category: MC4, MC5, MC6, and MC7. When considering edge capacity, we have another two classification which consider static edge capacity or dynamic edge capacity. In the static edge capacity, we select the edge with minimum capacity with MC6 algorithm and we select the edge with maximum capacity with MC7. However, in the dynamic edge capacity, we consider the available remaining edge capacity of an edge in the cut set. Furthermore, choosing the edge with minimum remaining capacity is *MC4 algorithm* and choosing the edge with maximum remaining capacity is *MC5*. The *MC6* and *MC7* algorithms consider the edge capacity, regardless of available amount of capacity at the time of the cut, where *MC6* select the edge with the minimum capacity in the cut set while *MC7* selects the one with maximum capacity. When an algorithm considers the edge capacity, it is actually considering the most congested or the least congested. When routing using *MC4* the flows tend to use the remaining capacity available of an edge to send the flow over it while in *MC5* the flow tends to load balance by sending the flows over the least congested paths.

IV. SIMULATION ENVIRONMENT

In our simulation, the network is modeled as an undirected graph $G(V, E)$, where the set of nodes V are connected by edges from the edge list E . Each edge $e_i(v_{i-1}, v_i) \in E$ is characterized by two nodes v_{i-1} and v_i and hold a capacity $(C(e) > 0)$. The flows are in our model are defined by the set flows (F) , where each $f_i \in F$ is characterized by a source node indicated as (s) and the destination node by (d) . The flow path which connects (s) to (d) is defined by the set P , where $(P = \{e_{(s,v1)}, e_{(v1,v2)}, \dots, e_{(vi,d)}\})$. The path P bandwidth is indicated by $(b(P))$ which defines the minimum edge capacity in the path P .

Two network topologies are simulated to evaluate the proposed algorithms, the Max-Flow Min-Cut suit of algorithms *MFMC*, the Mesh network and the Barabasi Albert network. The Mesh network, modeled as graph G , is

the network where in general each node v_i is connected to at least one node in G . The full connected mesh network is the mesh network where each node v_i is connected with every other node in V and it has an edge degree of $N-1$, otherwise we call it partial connected mesh network. We simulate a partial connected mesh network which gives a distinctive result in calculating an objective function such as maximizing the mean throughput compared to the full connected mesh, therefore, we choose the partial connected mesh network with an edge degree of $N/2$.

The second network topology we simulate is the Barabasi Albert (BA) network which is based on generating networks with power law degree distribution. The power law degree distribution starts with a minimum two nodes n_1 and n_2 then it starts to expand by adding one node at a time with a degree of K , where $K < [n]$. Every new node has the choice to be connected with another with higher degree in the network which highlight node popularity or most visited [40].

The *MFMC* algorithms are evaluated over these two-network topology, the partial connected Mesh and the Barabasi Albert network. Our simulator is developed in Python language and to generate our network graphs, we have used the NetworkX library¹ to develop graph nodes and edges and used its built-in functions to maintain the graphs. The network size ranges from 10 to 50 nodes and the edges capacity are generated randomly, where each edge capacity C is bound by $(1 \leq C(e) \leq 10)$. The flows are generated randomly by randomly selecting a source node s and a destination node d in a uniform distribution.

The problem, as defined earlier, with having a network graph G and a set of flows F we need to non-preemptively route those flows over G such that we optimize a set of defined objective functions. These objective functions are defined in Equations (1) to (5). The source node is represented as s and the destination as d . The time is t , which is measured in unit time, therefore, flow release time is denoted by Rt , flow completed or finished time by Ft and the maximum completion time by $max(Ft)$. The minimum travel time between s and d is denoted by Tt , as defined in Equation (1), and the flow size and path length are denoted by $l(f_i)$ and $l(p_i)$, respectively. Our simulated objective functions represent an optimization factor which calculates network performance such as: the mean wait time representing average delay rate WT , the mean throughput TH , the mean slowdown SD , the mean makespan MS , and the maximum completion time $(max(MS))$.

$$Tt_i = l(f_i) + l(p_i) - 1 \quad (1)$$

$$WT_i = Ft_i - Rt_i - Tt_i \quad (2)$$

$$MS_i = Ft_i - Rt_i \quad (3)$$

$$TH_i = l(f_i) / (Ft_i - Rt_i) \quad (4)$$

$$SD_i = (Ft_i - Rt_i) / Tt_i \quad (5)$$

In our simulation each node in the graph could be either a source node, a destination node or a router. Flows are generated randomly by randomly picking a source node and a destination node and define its release time either randomly or all flows release times are set to zero. The size of each flow is defined by either a Gaussian distribution or Pareto

¹ <https://networkx.github.io>

distribution. The parameters for the Gaussian distribution are set as ($\mu=5$) and ($\sigma=1$) and for the Pareto distribution, to represent very long flows as large files in the network, is set as ($\alpha=1$).

The algorithms, the *MFMC* suit of algorithms and the Shortest Path *SP* algorithms, are evaluated under different set of problem scenarios. Under different network topology, the flows could be generated with all release times set to zero ($Rt=0$) to evaluate the algorithms under a huge burst of flows in the network, otherwise, flows are released in arbitrary times set by a uniform distribution where ($Rt \geq 0$). In both scenarios the flow size is set by either a Gaussian flow size distribution or a Pareto flow size distribution.

In simulating a non-preemptive scheduling problem with a non-clairvoyant constrain, we simulate an uninterruptible flow packets sequence where the flows are not known before they are released. In the non-clairvoyant we hold no knowledge of the future network flows in advance, therefore, the flows are visible to the scheduler once they are released.

V. RESULTS AND ANALYSIS

Our Max-Flow Min-Cut based routing algorithm *MFMC* is evaluated within different problem variations based on the network and flow characteristics. *MFMC* is evaluated against different Shortest Path algorithms which consider short in number of hops or short in distance. These shortest path algorithms are noted as: (SSP-md/DSP-md), which calculates the Static/Dynamic Shortest Path based on minimum distance, (SSP-mh/DSP-mh), which calculates the Static/Dynamic Shortest Path based on minimum hops, and (SSP-inv/DSP-inv) which chooses the most congested path in the network by applying Dijkstra's algorithm over the inverted edge weights. With this kind of evaluation in variant problem formulation, it gives clear evaluation of our algorithm performance.

The performance of the algorithms was measured by the results of the underlying objective functions. We have evaluated four objective functions: We evaluated four objective functions: minimizing the Mean Wait Time, minimizing the Mean Slowdown, minimizing the maximum completion time (makespan) and maximizing the Mean Throughput.

The performance of the dynamic routing algorithms performs better than the static algorithms because they take the network link states in consideration when finding the path. We have evaluated the performance of the three shortest path algorithms for both static state and dynamic. For example, in a Barabasi Albert network, as in Fig 1, we show the results of running the dynamic shortest path algorithms next to the static once in minimizing the mean wait time. In the results and analysis of our *MFMC* algorithms, we omit the static version of the shortest path algorithms for simplicity. Furthermore, we compared our *MFMC* suit of algorithms to each other where we have found that *MC3* performed better than the rest of *MFMC* algorithms. As an illustration we show in Fig 2, the mean wait time is minimized by *MC3* on average more than the rest of *MFMC* algorithms by 28.2%, minimize the mean slowdown by 7.24%, minimizing the maximum completion time by 18.5%

and maximizing the mean Throughput by 5.93%. The *MC6* and *MC7* algorithms in a sense are the static version of *MC4* and *MC5* since they consider the minimum/maximum edge capacity instead of the minimum/maximum available edge capacity, therefore, for the rest of the analysis section we omit it from our figures for better visibility. Detailed analysis of these algorithms is stated in Table II, where denote the mean wait time by (MWT), the mean slowdown by (MSD), the mean throughput by (MTH), and the maximum completion time by (Makespan).

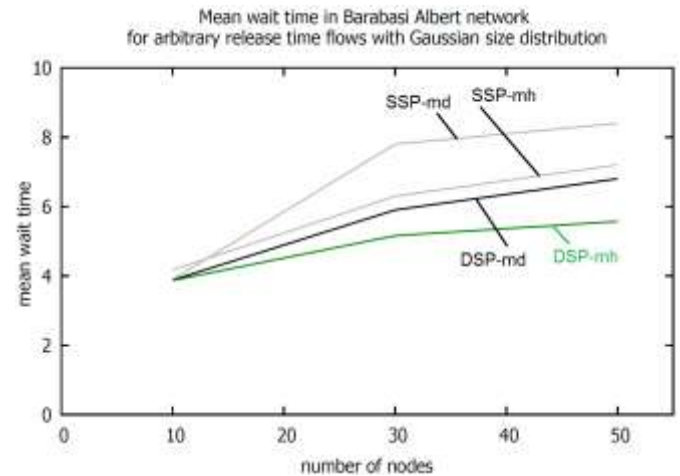


Fig. 1. Comparing the performance of Static Shortest Path (SSP) algorithms with the Dynamic Shortest Path (DSP) algorithms in minimizing the mean wait time

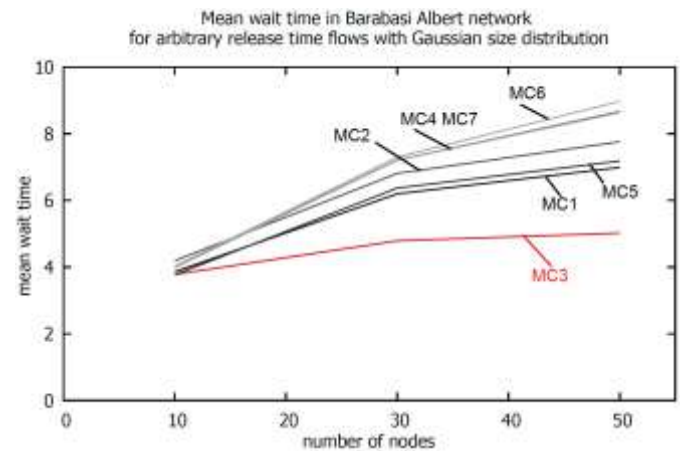


Fig. 2. Comparing the performance of our *MFMC* suit of algorithms in minimizing the mean wait time

maximizing the mean Throughput by 5.93%. The *MC6* and *MC7* algorithms in a sense are the static version of *MC4* and *MC5* since they consider the minimum/maximum edge capacity instead of the minimum/maximum available edge capacity, therefore, for the rest of the analysis section we omit it from our figures for better visibility. Detailed analysis of these algorithms is stated in Table II, where denote the mean wait time by (MWT), the mean slowdown by (MSD), the mean throughput by (MTH), and the maximum completion time by (Makespan).

In minimizing the mean wait time regardless of network topology we have found that our *MC3* algorithm minimized the mean wait time more than the *DSP-md*, *DSP-mh*, *DSP-inv* on average by 18.4%, 11.7% and 15.3%, respectively.

However, the remaining of the *MFMC* algorithms on

Table II
Comparing Max-Flow Min-Cut Algorithm to the Path Finding Algorithms

Objective Function	Network Type	Flow Size Distribution	Percent difference with DSP-md							Percent difference with DSP-mh				
			MC1 & 2	MC3	MC4	MC5	MC6	MC7	MC1&2	MC3	MC4	MC5	MC6	MC7
Minimize MWT	BA	Gaussian	12.6%	-14.5%	25.2%	8.9%	27.7%	24.8%	22.6%	-7.0%	36.3%	18.6%	38.9%	35.8%
		Pareto	-5.6%	-23.6%	7.5%	-3.1%	2.6%	10.6%	10.7%	-10.4%	26.1%	13.7%	20.4%	29.7%
	Mesh	Gaussian	37.5%	-16.1%	36.9%	20.5%	28.5%	20.7%	26.0%	-8.4%	49.5%	31.4%	40.2%	31.7%
		Pareto	-10.2%	-25.1%	4.9%	-4.7%	3.8%	-4.2%	7.0%	-10.7%	25.1%	13.6%	23.7%	14.2%
Minimize MSD	BA	Gaussian	1.8%	-4.8%	6.6%	0.7%	6.7%	5.6%	4.5%	-2.3%	9.4%	3.4%	9.6%	8.4%
		Pareto	-4.1%	-7.3%	-3.8%	-4.9%	-3.3%	-2.5%	2.5%	-0.8%	2.9%	1.8%	3.5%	4.3%
	Mesh	Gaussian	4.7%	-5.9%	8.3%	3.1%	5.9%	2.9%	9.1%	-2.0%	12.9%	7.5%	10.4%	7.2%
		Pareto	-2.9%	-7.4%	1.1%	-3.2%	2.4%	-2.8%	3.8%	-0.9%	8.1%	3.5%	9.5%	3.9%
Maximize MTH	BA	Gaussian	-5.2%	6.8%	-3.7%	7.3%	-1.9%	-1.8%	-8.9%	2.6%	-7.5%	3.1%	-5.8%	-5.6%
		Pareto	1.3%	15.6%	4.8%	9.4%	7.8%	8.5%	-3.4%	4.5%	-5.3%	-1.1%	-2.6%	-2.0%
	Mesh	Gaussian	-4.9%	7.8%	-1.1%	7.1%	2.5%	3.3%	-5.0%	7.7%	-1.3%	6.7%	2.3%	3.2%
		Pareto	-4.2%	10.4%	-1.3%	11.6%	1.8%	3.6%	-9.6%	4.2%	-7.0%	5.3%	-4.0%	-2.3%
Minimize makespan	BA	Gaussian	7.1%	-7.7%	11.1%	6.6%	14.4%	12.9%	11.8%	-3.7%	16.0%	11.3%	19.1%	17.8%
		Pareto	3.1%	-5.3%	16.2%	3.11%	27.5%	11.1%	11.7%	2.5%	27.4%	11.7%	31.1%	20.3%
	Mesh	Gaussian	29.3%	-10.9%	11.8%	-0.7%	16.4%	7.3%	36.7%	-1.0%	24.2%	10.4%	29.2%	19.2%
		Pareto	21.8%	-10.6%	8.2%	-9.9%	10.3%	-2.7%	28.8%	-5.5%	14.3%	-4.8%	16.5%	2.9%

average have increased the mean wait time compared with all *DSP* algorithms by 16.4%. In addition, comparing the

performance of the algorithms with different flow size distributions we have found that the mean wait time is minimized with flows with Pareto size distribution more than with flows of Gaussian size distribution by 20.6% and 16.1%, respectively.

We illustrate our results in the objective function minimizing the mean wait time with the example: having a Barabasi Albert network with arbitrary release time flows with Pareto flow size as in Fig 3. The Red line represent our *MC3* algorithm and the gray lines represent the other variation of *MFMC* and we compare them to the dynamic algorithms, *DSP-md*, *DSP-mh*, *DSP-inv*.

The analysis result in minimizing the mean slowdown we also found that *MC3* algorithm minimized this objective function more than the *DSP-md*, *DSP-mh*, *DSP-inv* on average by 6.4%, 1.5% and 7.2%, respectively. Moreover, the mean slowdown is again minimized by Pareto flow size distribution more than Gaussian flow size distribution by 8.4%, respectively. However, the remaining *MFMC* algorithms have increased the mean slowdown over all *DSP* on average by 5% as illustrated in Fig 4.

With the aim to maximize the mean throughput, regardless of network topology, our analysis shows that our *MC3* algorithm has maximized the mean throughput more than all *DSP* on average by 18.3%. Moreover, comparing the results of this objective function between flows with Pareto size distribution and flows with Gaussian size distribution they both performed the same by 18.6% and 17.2%, respectively. However, *MC1*, *MC2* and *MC4* have not increased the mean

throughput over the *DSP*, it has minimized it by 2.7%. As for *MC5*, *MC6* and *MC7*, they maximized the mean throughput more than all *DSP* on average by 2.4%.

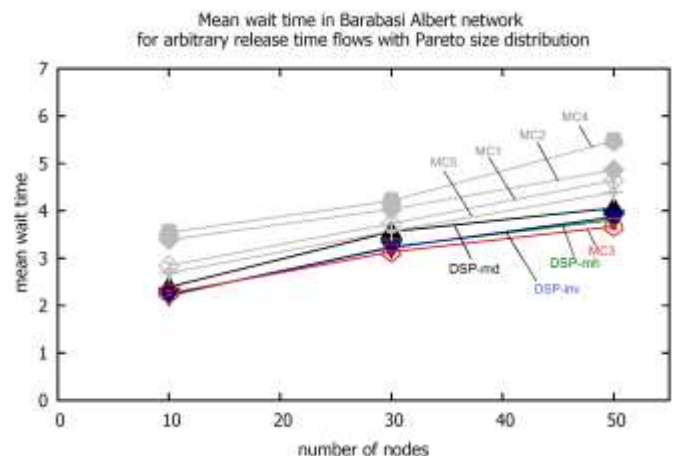


Fig. 3. Mean wait time in Barabasi Albert network with arbitrary release time flows and Pareto size distribution

We illustrate our results in the objective function maximizing the mean throughput with the example: having a Barabasi Albert network with arbitrary release time flows with Pareto flow size as in Fig 5. The Red line represent our *MC3* algorithm and the gray lines represent the other variation of *MFMC* and we compare them to the dynamic algorithms, *DSP-md*, *DSP-mh*, and *DSP-inv*.

In minimizing the maximum completion time (makespan), regardless of network topology, we also found that our *MC3* algorithm minimizes the maximum completion time more than all *DSP* on average by 9.6%. Moreover, this objective

function is minimized with flows of Pareto size distribution more than with flows of Gaussian size distribution by 18.2% and 7.5%, respectively. However, the remaining *MFMC* algorithms have increased the makespan over the *DSP* on average by 5.7%.

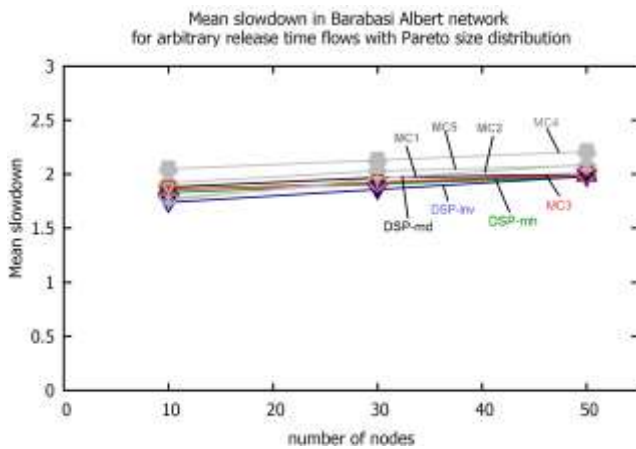


Fig. 4. Mean Slowdown in Barabasi Albert network with arbitrary release time flows and Pareto size distribution

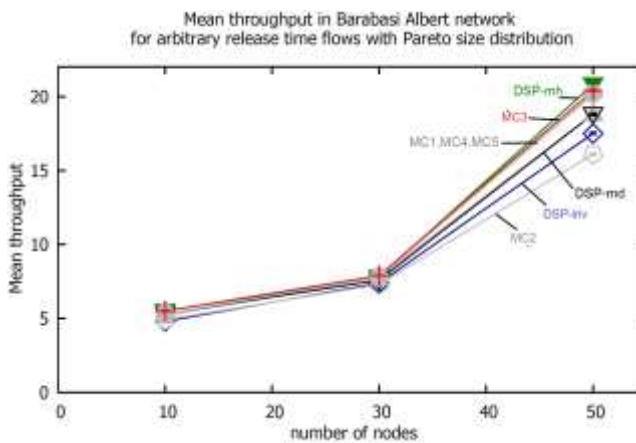


Fig. 5. Mean Throughput in Barabasi Albert network with arbitrary release time flows and Pareto size distribution

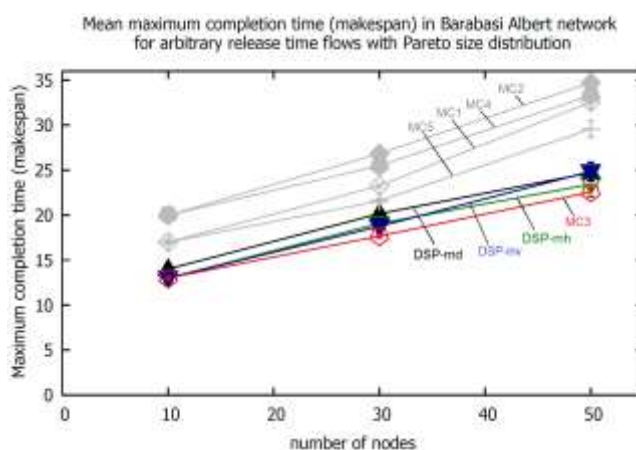


Fig. 6. Mean Maximum Completion Time in Barabasi Albert network with arbitrary release time flows and Pareto size distribution

In Fig 6, we illustrate the results of minimizing the maximum completion time in a Barabasi Albert network with flows released in arbitrary times with Pareto flow size distribution. Again, the red line represents our *MC3* algorithm while the gray lines represent the remaining

MFMC algorithms and we compare them to the dynamic shortest path algorithms as well.

In our various simulations we have found that constraint on flows release time, with arbitrary or identical, had no impact on the evaluation on our objective functions. Moreover, selecting the edge from the cut set in our *MFMC* algorithms based on being part of a minimum hop path optimized the network performance in minimizing the mean wait time, minimizing the mean slowdown, minimizing the maximum completion time and maximizing the mean throughput.

VI. CONCLUSION

In summary, we present seven Max-Flow Min-Cut (*MFMC*) based algorithms that solve the flow routing problem in the Software Defined Network Controller. The performance of end-to-end routing in the Software Define Network controllers can be maximized with the use of a routing algorithm that is fast and near optimal. These algorithms identify potential traffic congestion using several *MFMC* cut through their choice of edges in constructing the route to send flows over them. The route construction mechanism is based on one of three categories: random selection, capacity-based selection, and shortest path selection. In random selection, the path is constructed by randomly picking edges from the cut set which our results didn't show improvements in the network objective functions. However, with capacity-based selection the flows tend either to route through the most congested paths with minimum capacity or they tend to balance network load by routing through maximum edge capacity. In the third category, we considered shortest path-based algorithm where we have found that *MC3* algorithm optimized the network performance, such that the mean wait time is reduced by 15.1%, the mean slowdown is reduced by 5.1%, reducing the maximum completion time by 9.6%, and increasing the mean throughput by 18.3%. By comparing shortest path based selection to the capacity based selection, we found that *MC3* performed better which indicates that topology based decision is more important than capacity based in choosing an edge from the cut set.

The evaluation results of our simulation showed that our novel Max-Flow Min-Cut (*MFMC*) algorithm with an edge selection based on shortest path which picks the edge in the cut set with minimum hops provides superior performance contrast to the Shortest Path *SP* algorithm variants. Using *SP* algorithms for network routing often degrade the network performance due to the bottlenecks it causes by sending most flows over the most congested links which minimizes the throughput and increase the slowdown. These simulations show that by explicitly identifying bottlenecks in a routing algorithm through the least available flows in the cut edges along with picking the edge that is part of potential shortest path yield better network performance. Our *MFMC* is capable to finding the path which result in significant improvement over the commonly used shortest path algorithms. We have demonstrated with our *MFMC* algorithms that choosing the edges that leads to shortest path, such as in *MC3*, yields better performance across a broad range network types and flow size distributions compared to a balanced or longer path approach.

REFERENCES

- [1] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic engineering in software defined networks. In 2013 *Proceedings IEEE INFOCOM*, pages 2211–2219. IEEE, 2013
- [2] C. J. Bernardos, A. de la Oliva, P. Serrano, A. Banchs, L. M. Contreras, H. Jin, and J. C. Zúñiga. An architecture for software defined wireless networking. *IEEE Wireless Communications*, 21(3):52–61, June 2014
- [3] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN: An Intellectual History of Programmable Networks. *SIGCOMMComput. Commun. Rev.*, 44(2):87–98, Apr 2014.
- [4] N. McKeown. *How SDN will shape networking*, Oct 2011.
- [5] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association, 2005
- [6] H. Kim and N. Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, February 2013
- [7] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.
- [8] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMMComput. Commun. Rev.*, 38(2):69–74, March 2008.
- [9] A. Detti, C. Pisa, S. Salsano, and N. Blefari Melazzi. Wireless Mesh Software Defined Networks (wmSDN). In 2013 *IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 89–95, Oct 2013.
- [10] Q. Chen, K. Zheng, F. Ouyang, X. Gan, Y. Xu, and X. Tian. A shortest path routing algorithm based on virtual coordinate in NELs. In 2016 *8th International Conference on Wireless Communications Signal Processing (WCSP)*, pages 1–5, Oct 2016.
- [11] A. Nanda and A. K. Rath. Cost effective mod leach-a x2217; search algorithm for shortest path routing in wireless sensor networks. In 2016 *Sixth International Symposium on Embedded Computing and System Design (ISED)*, pages 147–151, Dec 2016.
- [12] Dexiang Xie, Haibo Zhu, Lin Yan, Si Yuan, and Junqiao Zhang. An improved Dijkstra algorithm in GIS application. In *World Automation Congress 2012*, pages 167–169, June 2012.
- [13] L. R. Ford and D. R. Fulkerson. Network flow and systems of representatives. *Canadian Journal of Mathematics*, 10:78–84, 1958
- [14] Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.
- [15] Lester R Ford Jr. *Network flow theory*. Technical report, Rand CorpSanta Monica Ca, 1956.
- [16] P. Huang, C. Wang, and L. Xiao. Improving end-to-end routing performance of greedy forwarding in sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):556–563, March 2012.
- [17] Nageswara S. V. Rao, Sridhar Radhakrishnan, and Bang-Young Choel. Netlets: Measurement-based routing for end-to-end performance over the internet. In *Pascal Lorenz*, editor, *Networking ICN 2001*, pages 184–193, Berlin, Heidelberg, 2001. Springer
- [18] L. Wang and H. Liang. Research and Improvement of the Wireless Sensor Network Routing Algorithm GPSR. In 2012 *International Conference on Computing, Measurement, Control and Sensor Network*, pages 83–86, July 2012.
- [19] Y.-F. Wen and Y.-S. Lin. Fair bandwidth allocation and end-to-end delay routing algorithms for wireless mesh networks. *IEICE Transactions on Communications*, E90-B(5):1042–1051, 2007
- [20] T. N. Nagabhushan, S. P. S. Prakash, and K. Krinkin. Power-saving routing algorithms in wireless mesh networks: A survey. In 2012 *11th Conference of Open Innovations Association (FRUCT)*, pages 107–115, April 2012
- [21] Rajan Sharma. ANN Based Framework for Energy Efficient Routing In Multi-Hop WSNs. *International Journal of Advanced Research in Computer Science*, 8(5), 05 2017
- [22] Hua R. Wu and Li Zhu. A wireless sensor network routing algorithm based on Zigbee. *Applied Mechanics and Materials*, 513-517:1845–1849, 2014.
- [23] Abdeljalil Rachadi, Mohamed Jedra, and Nouredine Zahid. Self-avoiding paths routing algorithm in scale-free networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 23(1), 2013.
- [24] Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958
- [25] Donald B Johnson. A note on Dijkstra’s shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973.
- [26] Jang-Ping Sheu, Quan-Xiang Zeng, R. Jagadeesha, and Yeh-Cheng Chang. Efficient unicast routing algorithms in software-defined net-working. In 2016 *European Conference on Networks and Communications (EuCNC)*, pages 377–381, June 2016.
- [27] Man-Ching Yuen, Weijia Jia, and Chi-Chung Cheung. Simple math-ematical modeling of efficient path selection for qos routing in load balancing. In 2004 *IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No. 04TH8763)*, volume 1, pages 217–220. IEEE, 2004.
- [28] Hong Seong Chang, Byoung Wan Kim, Chang Gun Lee, Sang Lyul Min, Yanghee Choi, Hyun Suk Yang, Doug Nyun Kim, and Chong Sang Kim. Performance comparison of static routing and dynamic routing in low-earth orbit satellite networks. In *Proceedings of Vehicular Technology Conference - VTC*, volume 2, pages 1240–1243 vol.2, 1996.
- [29] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- [30] J. J Garcia-Luna-Aceves. A minimum-hop routing algorithm based on distributed information. *Computer Networks and ISDN Systems*, 16(5):367 – 382, 1989..
- [31] R. A. Guerin, A. Orda, and D. Williams. Qos routing mechanisms and ospf extensions. In *GLOBECOM 97. IEEE Global Telecommunications Conference*. Conference Record, volume 3, pages 1903–1908 vol.3, Nov1997
- [32] E. Akin and T. Korkmaz. Comparison of Routing Algorithms with Static and Dynamic Link Cost in Software Defined Networking (SDN). *IEEE Access*, 7:148629–148644, 2019.
- [33] Martin F. Arlitt and Carey L. Williamson. Web server work load characterization: The search for invariants. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’96, page 126–137, New York, NY, USA, 1996. Association for Computing Machinery.
- [34] Kihong Park, Gitae Kim, and M. Crovella. On the relationship between file sizes, transport protocols, and self-similar network traffic. In *Proceedings of 1996 International Conference on Network Protocols (ICNP-96)*, pages 171–180, 1996.
- [35] A. U. Shah, D. H Bhatt, P. R. Agarwal, and P. R. Agarwal. Effect of packet-size over network performance. *International Journal of Electronics and Computer Science Engineering*, 1:762–766, 2012.
- [36] Chuan Lin, Yuanguo Bi, Hai Zhao, Zeshen Wang, and Jinfa Wang. Scheduling algorithms for time-constrained big-file transfers in the internet of vehicles. *Journal of Communications and Information Networks*, 2(2):126–135, Jun 2017.
- [37] R. Ghrist and S. Krishnan. A topological max-flow-min-cut theorem. In 2013 *IEEE Global Conference on Signal and Information Processing*, pages 815–818, Dec 2013.
- [38] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Networkflows: Theory, applications and algorithms*. Prentice-Hall, Englewood Cliffs, New Jersey, USA Arrow, KJ (1963). Social Choice and Individual Values, Wiley, New York. Gibbard, A. (1973). “Manipulation of Voting Schemes: A general result”, *Econometrica*, 41:587–602, 1993.
- [39] Honghua Hannah Yang and D. F. Wong. *Efficient Network Flow Based Min-Cut Balanced Partitioning*, pages 521–534. Springer US, Boston, MA, 2003.
- [40] S. Tie-li, D. Jing-wei, and D. Kai-ying. Scale-free network model with evolving local-world. In 2008 *Fourth International Conference on Natural Computation*, volume 1, pages 237–240, Oct 2008
- [41] Nada Alzaben and Daniel W. Engels. End-to-end routing in SDN controllers using max-flow min-cut route selection algorithm. In 2021 *23rd International Conference on Advanced Communication Technology (ICACT)*, pages 461–467. IEEE, 2021
- [42] Nada Alzaben and Daniel W. Engels. End-to-end routing approach for SDN based wireless ad hoc network. In 2021 *IEEE Symposium on wireless technology and applications (ISWTA)*, 2021, pp. 16-21.



Nada Alzaben. This author became a Member (M’18) of IEEE in 2018. Saudi citizen born in Riyadh, Saudi Arabia. Received her B.S. (2001) and M.S. (2012) in computer science and information from King Saud Univ. in Riyadh, Saudi Arabia.

She is currently working on her Ph.D. in computer science at Southern Methodist Univ., Dallas, TX, USA. Her research interests include network communications, routing and scheduling, network performance analysis. She is a Lecturer in computer science and information college at Princess Nourah Bint Abdulrahman Univ.,

Riyadh, Saudi Arabia. Previously she worked as a senior programmer at King Saud Univ. She has 5 research publications in the field of computer networks and AI.



Dr. Daniel W. Engels received his Ph.D. from the Massachusetts Institute of Technology. Born in the US.

Dr. Engels is currently Head of AI, Cyber security Science and Analytics, at HSBC. Previously, Dr. Engels was a professor at Southern Methodist University (SMU) where he was one of the creators of, and the first Director of, the Master of Science in Data Science (MSDS) program where he grew the program from 0 students to more than 300 students in its first two years of operation. In addition to his data science expertise, Dr. Engels is

an expert in security and in RFID and IoT technologies, systems, and applications. He was one of the founders of the IEEE International Conference on RFID, and he was the Chair of the IEEE Technical Committee on RFID in 2011 and in 2012. Dr. Engels is the former Director of Research of the Auto-ID Labs at MIT, where he led the development of several RFID protocols including the original “Gen2” protocol, and he is an original member of the research team that founded the Auto-ID Center at MIT and created the EPC System.

He has over 100 peer reviewed publications and 8 issued U.S. patents in data science, RFID, RFID applications, Internet of Things, security, embedded computing, and computer-aided design. Dr. Engels received the 2014 AIM Ted Williams Award in recognition of his contributions to the AIDC industry. Dr. Engels is a member of AIDC 100 and is a Senior Member of IEEE.