# Resource Management in Software Defined Coalitions (SDC) through Slicing

Athanasios Gkelias*, Nitish K. Panigrahy†, Mohammad Mobayenjarihani†, Kin K. Leung*, Don Towsley†,
Patrick J. Baker‡, Olwen Worthington§,Laurence Fowkes¶,
*Imperial College London, UK. † University of Massachusetts Amherst, USA. ‡ Royal Air Force, UK
§ Defence Science and Technology Laboratory, UK ¶ Strategic Command, Defence Digital, UK
Email: agkelias@imperial.ac.uk, nitish@cs.umass.edu, mobayen@cs.umass.edu, kin.leung@imperial.ac.uk,
towsley@cs.umass.edu, pbaker@dstl.gov.uk, olworthington@dstl.gov.uk, laurence.fowkes903@mod.gov.uk

***Abstract*—Future defence infrastructure systems will require increased flexibility and agility to respond to changing application goals, external threats and complex environments. A key enabler for such agility is Software Defined Coalitions (SDC), where the network comprise multiple domains of resources owned by different defence units (partners) but dynamically joined together to form an infrastructure for communications and computation.**

**Software Defined (SD) Slicing aims to enable agile and near-real-time provision and configuration of "slices" of the infrastructure resources for supporting future communications and computing applications. An SD slice, makes use of the allocated resources distributed across several domains to support a set of applications including distributed analytic services.**

**We present a 3-level control architecture with a global SD-slice controller at the top, domain controllers (DC) in the middle, and dynamic, end-to-end flow controllers at the bottom. Associated with each DC is a domain inference engine whose function is to estimate availability of various resources in that domain. Based on the inferred resource availability in domains, the global controller determines the feasibility of supporting a new SD slice and if so, allocates resources to achieve/maintain the required performance of all slices. Based on the resources allocated by the global controller, the slice controller is responsible for sharing the resources across domains among data/processing flows to optimize resource utility. The end-to-end flow controllers then allocate resources to data flows or processing tasks according to dynamic conditions of resources for efficiency and robustness.**

***Index Terms*—Software Define Networking, Software Defined Slicing, resource allocation**

## I. Introduction

Software Defined Coalitions (SDC) architecture aims to enable agile and near-real-time provision and configuration of "slices" of resources needed to support military coalition missions. Each mission requires a set of distributed analytic services and at its inception an SDC slice is created to provide to and manage resources for a mission. These resources include processing, communications, storage, data-analytics, and sensing resources in various domains belonging to different coalition partners which may be both geographically and system physically separated, and in some cases with different use policies. Multiple SDC slices execute concurrently using and sharing the set of infrastructure assets. Hence resources must be carefully monitored and allocated to these slices.

Research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16- 3-0001.

Here we introduce a modular 3-level control architecture for the formation, resource management and optimisation of such slices, hereinafter referred to as SD slices (SDS). The proposed SDS control architecture comprises a global controller (GC) for all slices at the top, domain controllers (DCs) at the middle and dynamic flow controllers (DFCs) at the bottom. Associated with each DC is a domain inference engine (IE) whose function is to estimate availability of various resources in that domain. Based on the inferred resource availability in the domains, the GC determines the feasibility of supporting a new SD slice and if so, allocates resources to achieve/maintain required performance of all slices. Each slice will execute a collection of tasks and the slice controller (SC) is responsible for allocating its resources to these tasks by creating DFCs, one for each task (including data flow), that dynamically allocate resources to each task according to time-varying conditions of resources for efficiency and robustness.

The key enabler for SDS is Software Defined Networking (SDN), a concept that emerged over a decade ago as a networking architecture to enable networks to be intelligently and centrally controlled in order to improve performance, interoperability, flexibility and agility. The driving principle of SDN is the separation of the control plane from the underlying routers and switches that forward the data traffic (i.e., the data plane). In this way, network switches become simple forwarding devices and network control is held with software-based controllers, which are (at least logically) centralised. Recently, the concept of SDN has been generalised beyond network communication components and functionalities. Specifically, the DAIS ITA team has extended SDN to include other software defined capabilities and resources such as storage and computation to form the Software Defined Coalitions (SDC) architecture (see https://dais-ita.org/pub). The idea of Software Defined Environments (SDE) by bringing together software defined compute, network and storage under a unified control plane are also discussed in [6].

In the defence/military sector, SDN has been considered as the technology enabler for the development of Federated Mission Networks (FMN), based on the NATO Protected Core Networking (PCN) architecture, with the goal of supporting secure information sharing among coalition nations [3]. In PCN, a coalition core network is established by intercon-

necting network equipment and/or services provided by each participating network. The SDC architecture also represents a generalization of the PCN concept to include all type of coalition resources such as communication, storage, computation, databases, sensors, etc. The emphasis of the SDC has shifted from a single data center/enterprise domain environment to multiple coalition domain networks. For instance, [1] examines the application of software defined concepts to coalition operations and evaluates the key performance metrics of different SDC architectures, while [2] suggests a common architecture that allows SDN and Policy Based Management to coexist and work synergistically in SDC environments.

This paper further extends previous SDC related works by introducing a novel and robust way to formulate "slices" (i.e., SDS) and optimally manage coalition resources in such SDC environments. Our contribution is threefold: (1) we propose a 3-layer control architecture for SDS formation, resource allocation and optimisation; (2) we develop a machine-learning-based resource IE that estimates resource availability by capturing inter-dependencies among different resources and spatiotemporal network characteristics in each domain, and assists the GC to accept/reject new multi-domain slice requests; and (3) we develop a SC that manages tasks within the slice and determines their optimal data/task rates, and DFCs that enforce these rates while providing robustness to network changes, uncertainties and imperfect knowledge of resource availability.

The remainder of this paper is organized as follows. Section II presents the proposed 3-layer control architecture for SDS. The functionalities of the Domain Controller and the Inference Engine are described in sections III and IV, respectively. The Slice Controller and the Dynamic Flow Controller are presented in Section V, while conclusions are drawn in Section VI.

## II. SDC ARCHITECTURE

A coalition network is established by interconnecting domains of resources, such as communication, storage, computation, databases, sensors, etc., provided by participating coalition partners. A Software Defined Slice (SDS) is a group of resources and Virtual Network Functions (VNFs) spanning two or more coalition domains, which are reserved and used for the duration of a requesting coalition mission. In order to initiate a multi-domain slice formation, different domains need to communicate and negotiate the amounts and types of available resources they are willing to contribute (i.e., share with the other coalition partners). In our proposed architecture a logically centralised *global controller* (GC) coordinates resource sharing among SD slices across different domains. Using resource information from all domains and the resource requirements of a new SD slice request, the GC decides (e.g., by trying to optimise some utility function or based on predefined policies agreed on among the coalition partners) whether it is feasible to accept the new slice request. If so, the GC allocates resources across domains and requests the

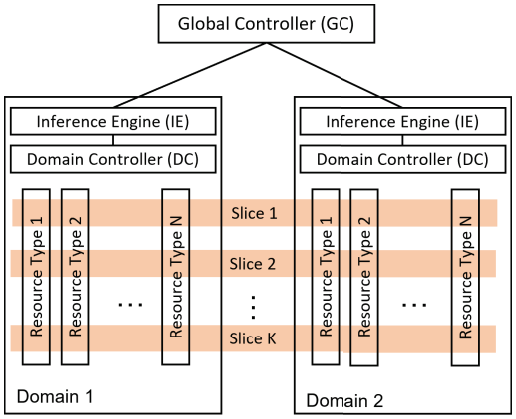



**Figure 1:** SDC architecture: GC, DC and IE interactions.

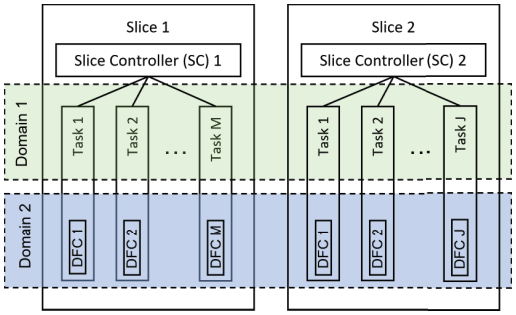



**Figure 2:** Slice Controller architecture.

*domain controllers* (DCs) to reserve the allocated resources for the new SD slice accordingly.

The GC may not need to know the exact network topology of the SDC infrastructure, nor how the corresponding domains distribute their resources among their components. In other words, the GC operates based on the knowledge of the aggregated available resources in each domain, which are estimated by resource *inference engines* (IEs) associated with the domain controller (DC) of the domain. Without revealing domain details, the IE learns and reports the available capacity of aggregated resources (e.g., a set resources that are used in highly correlated manners) to the GC.

The *domain controller* (DC) has precise knowledge of network topology and the distribution of the resources among all components within the associated domain. The DC receives the slice resource reservation request from the GC and decides whether the domain can provide the requested resources or not (i.e., admit or reject a proposed slice from the prospective of that domain). If a new slice is accepted, the DC enforces the resource reservation within its domain according to its resource-allocation mechanisms. The precise network and resource distribution information, collected and maintained by the DC, is used as input to the IE for the domain. In this way, the IE processes the detailed resource domain information and estimates the aggregated available resources in the corresponding domain.

After all DCs for the domains involved in the SD slice

accept the corresponding resource requests from the GC, a new *slice controller* (SC) is set up. For each slice, the unique SC is responsible for accepting or rejecting tasks within the slice and for optimal allocation of resources to the accepted tasks. The slice and the associated SC will be terminated when the military mission is completed. *A dynamic flow controller (DFC)* is activated for each accepted task. The DFC aims to achieve the optimal data/processing rates determined by the SC, while providing dynamic adjustment and robustness over network changes, uncertainties and imperfect knowledge of resource availability.

## III. Domain Controller (DC)

The DC serves two main functions related to resource management: (1) to accept or reject resource reservation requests from the GC and enforce resource reservations determined by the GC for the accepted slices in the corresponding domain; and (2) to assist the IE to generate estimates of the aggregated available domain resources for the GC to make inter-domain resource reservation decisions for slices. Compared to the admission control in traditional networks (e.g., [5]) or in SDN where resource sharing between slices is allowed, the resource-reservation approach has the following two advantages:

(1) Acceptance or rejection of resource-reservation requests at the DC is based on the *amount of already reserved resources in the domain*, which remain unchanged for the whole duration of a slice, and not on the current resource usage, which is usually stochastic and depends on the underlying tasks running on the slice. Note that slices may not operate at their maximum resource utilisation. Slice resource utilisation depends on the number and type of running tasks in the slice, managed by the SC. Therefore, information about the reserved resources needs to be updated only when a slice is admitted or terminated, and not when new tasks on a slice are generated or terminated.

(2) The DC has precise knowledge of the network topology and status for the associated domain. It decides for every new slice how, when, where and what resources in the domain will be reserved (if the slice request is accepted by the GC). Therefore, the DC possesses all information required for assisting the GC in slice admission decisions and resource reservation *centrally and at any given time*, without the need for the network nodes to signal their remaining resources status back to the DC. In a nutshell, the DC always has a precise and overall picture of the resource reservations at every network component within its domain, and is the one which approves, coordinates and tracks any changes.

## IV. Inference Engine (IE)

Efficient resource allocation requires knowledge of the interdependence among different resources. For instance, consider the simple network topology in Fig. 3, comprising two servers, C1 and C2 with 10 and 1 MIPS computation power, respectively; two network gateways (i.e., A and B); and links that support different maximum data rates. If we access the network only through gateway A, the aggregated network resource capacity will be calculated either as $\{10$ MIPS computation power, accessible at 15Mbps$\}$ or $\{11$ MIPS computation power, accessible at 5Mbps$\}$. However, if we access the network only through gateway B, the aggregated network resources will be calculated as $\{11$ MIPS computation power, accessible at 1Mbps$\}$. As networks increase in size and

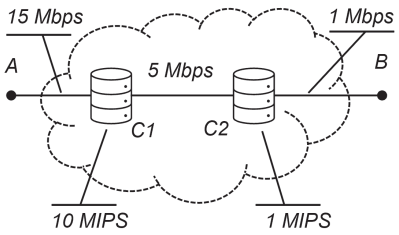



**Figure 3:** Example network topology to demonstrate the interdependencies among different resources.

complexity and we consider various types of resources (e.g., storage, energy, etc.) and resource performance properties (e.g., packet delay, outage probability, etc.), capturing such resource interdependencies becomes increasingly difficult. It becomes evident that the aggregated remaining resources in a domain cannot be captured by a single vector, but rather by a multidimensional manifold of values (where the number of dimensions equals the number of different types of resources and resource properties under consideration).

To select the optimal amounts and types of resources each domain has to allocate to a new slice, the GC needs to know all of the combinations of resources that each domain can contribute. In other words, the GC needs to know the aforementioned manifold for each domain. However, the DC, which maintains the up-to-date information of the resource distributions in its domain (i.e., network status), can only respond to a specific request of resource combination from the GC, and accept/reject it following DC's own admission control decision. In other words, the DC can only calculate (upon GC's request) a single point on the overall resource manifold. To overcome this issue, we propose the IE associated with each DC, which utilizes the DC's admission control decisions to calculate the remaining resource manifold for the current domain network status. The GC will admit a slice only if the corresponding DCs are able to admit the individual partial slice resource requests (by the GC) in their own domains.

### A. Inference Engine

Unreserved resources at every network component are easily summarised in a single vector, hereinafter referred to as *domain status vector (DSV)*. The DSV is a snapshot of the remaining capacities of all resources in the domain at any given time, and is updated by the corresponding DC every time a new slice is admitted or terminated. A similar vector is used to represent slice resource requests, hereafter referred to as *resource request vector (RRV)*. The RRV consists of the amounts of requested resources and for the slice spanning multiple domains, the IDs of the nodes which serve as ingress and egress points to the slice's data traffic in the corresponding

domain. Note that in some cases the ingress and egress points may be the same). For instance, consider the network of six

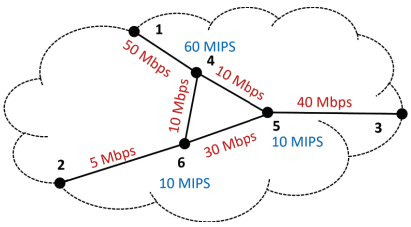



**Figure 4:** Network topology considered in our simulations

nodes in Fig. 4. Nodes 4, 5 and 6 have computing power and data forwarding capabilities, while nodes 1, 2 and 3 serve as domain gateways and can only forward data. The throughput and computation power values in the figure represent the remaining/unreserved resources at the corresponding links and nodes The DSV is $[50, 5, 40, 10, 10, 30, 60, 10, 10]$, where the first 6 values correspond to the remaining link capacities in Mbps and the following 3 values to the computing power in MIPS at nodes 4,5 and 6, respectively. For a slice resource request of 9 Mbps throughput and 35 MIPS computation power, to be served between gateway nodes 1 and 3, the corresponding RRV is $[9, 35, 1, 3]$. The IE concatenates DSV

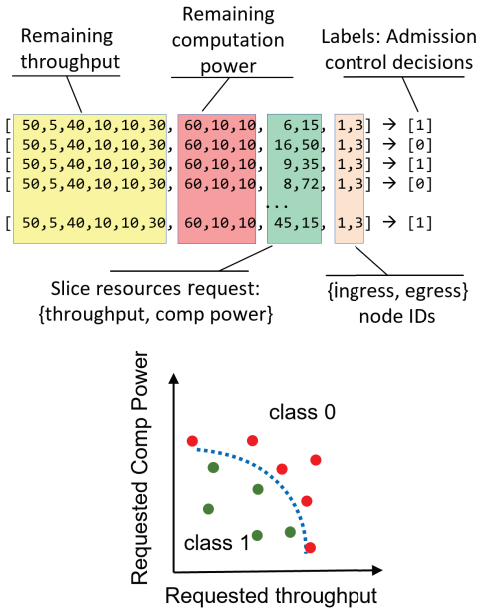



**Figure 5:** IE classification training data/vectors and labels (top). Classification output (bottom).

and RRV into a single vector labeled '1' if the slice requested resources have been accepted and '0' if they have been rejected. This dataset is used to train a deep learning classifier that can predict whether the resource request of a new slice request can be satisfied by the domain (given the current network state), thus to admit or reject a future request. In essence, this machine-based-IE captures the interdependencies among different resources and spatiotemporal network characteristics to predict the next resource-allocation decision by the DC. Therefore, for any given network state, we can estimate the capacities of the aggregated available resources by identifying the boundaries between these two classes (i.e., '0' and '1'), as demonstrated in Fig. 5.

### *B. Simulation Platform*

In order to demonstrate the feasibility of the proposed IE, we have developed a simulation platform. The purpose of this simulation platform is twofold, *i)* to generate data for the training of the machine-learning-based IE, and *ii)* to evaluate the performance of the engine. Without loss of generality, we consider a single domain topology represented by the graph in Fig. 4. Vertices (nodes) represent routers, some of them (i.e., nodes 4, 5 and 6) are equipped with data servers for computation. Edges represent communication links. Link and processing capacities are predefined and remain constant throughout the simulation. New slice requests, which requires reservation of communication and computation resources, are generated according to a Poisson process. The start and end nodes for each slice are randomly selected among the network gateways (i.e., nodes 1, 2 and 3). Data throughput and possible computational requirements for each slice are randomly chosen from a uniform distribution.

If a slice request is accepted the slice reserves its allocated resources for a time duration randomly chosen from a uniform distribution. The DC assumes the following procedure to select the processing node and data route for the new slice. For slices that request only communication resources, it selects the minimum hop-count route that can satisfy the throughput requirement of the new slice from the start to end nodes. If a slice requests both communication and computing resources, the DC first selects the node with the maximum available/unreserved computational power to meet the computational requirement. Then, it finds the shortest path from the start node to the selected processing node, and then another shortest path from the processing node to the end node that satisfies the throughput requirements. We assume that computation generates additional data, so the throughput requirement after the processing node is higher than that leading to the node, and is given by $T'_{req} = (1 + \frac{C_{req}}{C_0})T_{req}$ (where $T_{req}$ and $T'_{req}$ are the throughput requirements before and after the processing nodes, $c_{req}$ is the computing requirement and $C_0 > C_{req}$ a fixed system parameter). A new slice request is admitted only if the above procedure can identify such a server and route that satisfy the computing and throughput requirements for the slice; otherwise, the slice request is rejected. Let us assume that we want to calculate the frontier of the remaining throughput and computing power between nodes 1 and 3 for the given network state in Fig. 4. First we generate and input a number of vectors to the already trained IE. The vectors consist of the network state, followed by some requested throughput $T_{req}$ and computation power $C_{req}$ values, and the IDs of the input-output nodes (i.e., [50,5,40,10,10,30,60,10,10, $T_{req}$, $C_{req}$,1,3]). The output of the IE classifier to each of these vectors will be either '1' (if

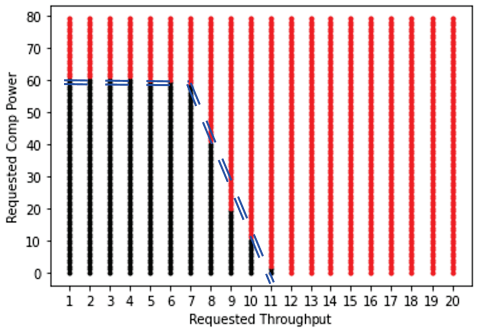
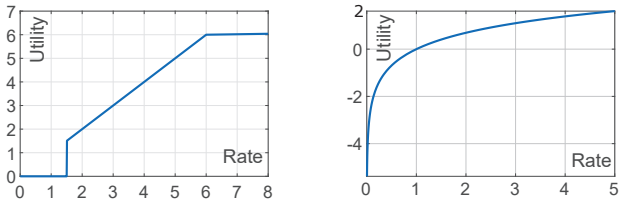



**Figure 6:** IE output: Black dots correspond to accepted resource requests, red dots to rejected requests. The blue line represents the available resources frontier.




(a) Video utility function (b) Log utility function

**Figure 7:** Utility functions for tasks

the $T_{req}$ and $C_{req}$ values can be supported) or '0' if they cannot. We start with some small $T_{req}$ and $C_{req}$ values that are classified as '1', and gradually increase the values until the input vector is classified as '0'. Each dot in Fig. 6 represents the corresponding throughput and computational power requests. Black dots correspond to admitted combinations of resource requests (i.e., '1's), while the red dots to rejected (i.e., '0's). The blue dashed line then denotes the remaining available resources frontier in the domain.

## V. Slice and Dynamic Flow Controllers

A slice can support a number of tasks arriving at different times, which require either communications and/or computing resources. Let $\mathcal{L}$ denote the set of resources and $\mathcal{K}$ the set of tasks. We assume that task $k \in \mathcal{K}$ requires resources $\mathcal{L}_k \subset \mathcal{L}$.

### A. Slice Controller (SC)

Each accepted slice uses its allocated resources to support communication/processing tasks. The responsibility of the SC is to accept or reject tasks running the slice and calculate their optimal rates. We associate with every task $k$ a non-decreasing utility function $U_k(x_k)$, where $x_k \geq 0$ represents the task's service rate. Examples of video and log utility functions are given in Fig. 7. The SC outputs a target rate $x_k^*$ for each task $k$ by solving the following optimization problem.

$$\textbf{SC:} \quad \max \sum_k n_k U_k(x_k), \tag{1}$$

$$\text{s.t.} \quad \sum_{k:l\in\mathcal{L}_k} a_{k,l} n_k x_k \leq B_l, \quad l \in \mathcal{L}, \tag{2}$$

$$0 \leq x_k^{min} \leq x_k \leq x_k^{max}, \quad \forall k \in \mathcal{K}, \tag{3}$$

$$n_k \in \{0,1\}, \quad \forall k \in \mathcal{K}, \tag{4}$$

where $B_l$ is the capacity for resource $l$ and $a_{k,l}$ is the rate multiplier associated with task $k$'s use of resource $l$, $k \in \mathcal{K}$, $l \in \mathcal{L}$. $x_k^{min}$ and $x_k^{max}$ denote the minimum and maximum rates that task $k$ can be handled. $n_k$ takes value one if task $k$ can be admitted (i.e., can be handled a rate of at least $x_k^{min}$) and zero otherwise. Here $B_l$ represents the resource capacity measured in Mbps when $l$ corresponds to a communications link or in MIPS when $l$ is a computing resource and $a_{k,l}$'s are multipliers used to normalize the consumption of different resources as illustrated next. Consider a database query task where each query requires is on average 10KB in size, 600K instructions to execute at a database server, and 25KB in size for a reply. The query size and response size are 0.08Mb and 0.2Mb respectively. Since the communication link capacities are expressed in Mbps, the multipliers associated with the query and reply will be 0.08 and 0.2 respectively. Similarly, since each query executes 0.6M instructions on average and computation capacity is measured in MIPS. Hence, the multiplier associated with the query execution is 0.6.

### B. Dynamic Flow Controller (DFC)

The goals of the DFCs are twofold, *i)* to achieve the optimal task rates determined by the SC with the assumption of perfect knowledge of available resource capacities), and *ii)* to be robust to inaccurate capacity knowledge of the SCs and dynamic changes to available resources over time. For each admitted task $k$, the DFC optimizes the transport-layer utility function $V_k(x_k) = \alpha_k \log x_k$. Here the weights $\alpha_k$ are chosen such that the rates converge to the optimal rates provided by the the SC and such that small uncertainties in resource availability result in small deviations from these rates. The design of the DFC follows from the theory of network utility maximization [4]. It has been proven that if the DFC has the optimal rates and their slopes (provided by SC), it will converge to the optimal SC output rates [9], thus solving (1).

We have considered three types of feedback for signalling the presence or absence of contention, i.e., loss based (LB), queue-length based (QB), and delay based (DB). All of them were initially proposed and studied in the context of TCP congestion control. Here we study them in the context of SDS involving communication and computation resources.

**Loss and Queue Length Based:** Both the LB and QB controllers update the task (flow) rate depending on the contention on resources that the task uses. To be precise, the rules governing how the flow rate of a task is set are given by

$$\begin{aligned} x_k &\leftarrow \min\{x_k^{max}, x_k + \gamma_k \alpha_k\}, \quad \text{no contention,} \\ x_k &\leftarrow \max\{x_k^{min}, x_k - \gamma_k x_k\}, \quad \text{contention present,} \end{aligned} \tag{5}$$

i.e., the flow rate is decreased when a contention signal is received and increased in case of no contention. Here, $\gamma_k$ is the gain parameter determining controller aggressiveness.

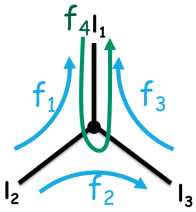



**Figure 8:** Network topology and flows

| Controller Type | $x_{f_1}$ | $x_{f_2}$ | $x_{f_3}$ | $x_{f_4}$ |
|---|---|---|---|---|
| Slice Controller | 16.766 | 17.233 | 16.766 | 1.67 |
| Loss Based | 16.84 | 18.08 | 16.84 | 4.90 |
| Queue Length Based | 16.18 | 16.63 | 16.18 | 1.664 |
| Delay Based | 16.766 | 17.233 | 16.766 | 1.666 |

**Table I:** Comparison of flow rates obtained from different controllers after convergence.

Similar to TCP Reno, the LB controller takes data loss as an indication of contention and the absence of data loss as an indication of no contention. The QB controller relies on the explicit signalling of contention. Suppose each resource $l$ in the coalition network maintains a queue of packets/processing requests. Denote $q_l$ as the queue length of resource $l$ and $b$ as the contention threshold. Under this mechanism, a contention is signaled on the resource when $q_l > b$.

**Delay Based:** In a DB flow controller, the flow rate for is updated based on queueing and processing delays observed by the task. One such flow controller used for TCP congestion control is the FAST TCP [10]. Here we apply FAST TCP to design a delay based flow controller for SDS that involves communication and computation resources. Denote the minimum end-to-end delay of task $k$ as $d_k$. This includes propagation and minimum processing delays. Let $T_k$ denote the overall end-to-end delay (propagation, queueing, & processing). Let $x_k^{new} = x_k + \gamma_k[\alpha_k - x_k(T_k - d_k)]$. Then under FAST, the flow rate of a task is updated according to the following rule:

$$x_k = \max\{x_k^{min}, \min\{x_k, x_k^{max}\}\} \tag{6}$$

### C. Performance Evaluation

Solving the SC optimization problem is NP-hard. In this paper, we have used MATLAB OPTI toolbox [8], which applies nonlinear optimization with mesh adaptive direct search algorithm (NOMAD) [7]. After solving the optimization problem, the SC sends the optimal rates and the gradients of the utility functions to the corresponding DFCs.

To evaluate the performance of different DFCs, we have developed a differential equation, fluid model simulator and considered the network topology shown in Fig 8. We assume three communication flows ($f_1, f_2,$ and $f_3$) and one database flow ($f_4$). Link capacities for all the communication links ($l_1, l_2,$ and $l_3$) are 34 Mbps. The database flow uses both communication and computation resources. The available capacity of the computation resource is 1 MIPS. The multipliers associated with query, query execution, and reply are 0.08, 0.6, and 0.2 respectively as mentioned above. We assume all the flows have log utilities. Also, $\alpha_1 = \alpha_2 = \alpha_3 = 1$ and $\alpha_4 = 10$. The results are presented in Table I. We observe that the delay based DFC converges to the rates provided by the SC for all four flows. On the other hand, the rates do not converge for the loss and queue-length based controllers. Note that, both the loss and queue-length based controller designs are based on a primal approach while the delay based design employ a dual approach. We further performed simulations involving only communication resources. We found the LB and QB controllers to be promising for these kinds of communication focused applications.

## VI. Conclusions

A 3-level control architecture for software-defined (SD) slices has been proposed, where domains of resources are owned and contributed from multiple coalition partners at geographically dispersed locations. The architecture comprises a Global Controller (GC) for all SD slices at the top, Domain Controllers (DCs) at the middle, and Dynamic Flow Controllers (DFCs), one for each task/flow, at the bottom. A novel resource inference engine (IE), associated with each DC, has been developed, whose function is to estimate availability of various resources in that domain. Based on the inferred resource availability in domains, the GC determines the feasibility of accepting requests of new SD slices and allocates resources to achieve/maintain the required performance of all slices. Based on the allocated resources, a slice controller (SC) is created and responsible for determining the processing rate of tasks running on the slice. By dynamic adjustments of task rate, each DFC allocates resources to its associated task or data flow according to dynamic conditions of resources for efficiency and robustness.

## References

[1] V. Mishra, D. Verma, C. Williams and K. Marcus, “Comparing Software Defined architectures for coalition operations," 2017 International Conference on Military Communications and Information Systems (ICMCIS), 2017, pp. 1-7, doi: 10.1109/ICMCIS.2017.7956476.

[2] C. Williams, E. Bertino, S. C. D. Verma, K. Leung and C. Dearlove, “Towards an architecture for policy-based management of software defined coalitions," IEEE SmartWorld 2017, pp. 1-6, doi: 10.1109/UIC-ATC.2017.8397420.

[3] J. Spencer and T. Willink, “SDN in coalition tactical networks," MILCOM 2016 - 2016 IEEE Military Communications Conference, 2016, pp. 1053-1058, doi: 10.1109/MILCOM.2016.7795469.

[4] F. Kelly, A. Maulloo, D. Tan. J Operational Res Soc 49 (1998) 237-252.

[5] C.H. Liu, K.K. Leung and A. Gkelias, “A Generic Admission-Control Methodology for Packet Networks," IEEE Transactions on Wireless Communications, Vol. 13, No. 2, pp.604-617, February 2014.

[6] Li, C-S., et al. “Software defined environments: An introduction," IBM Journal of Research and Development 58.2/3 (2014): 1-1.

[7] S. Le Digabel, "Algorithm 909: NOMAD: Nonlinear Optimization with the MADS Algorithm," ACM Transactions on Mathematical Software 37(4), pp. 44:1 - 44:15, 2011

[8] (2021 Jul. 20). [Online]. Available: https://www.inverseproblem.co.nz/OPTI/index.php/Solvers/NOMAD

[9] Spang, Bruce, Anirudh Sabnis, Ramesh Sitaraman, Don Towsley, and Brian DeCleene. “MON: Mission-optimized overlay networks." IEEE INFOCOM 2017, 1-9. IEEE, 2017.

[10] Cheng Jin, David X. Wei, and Steven H. Low, “FAST TCP: Motivation, architecture, algorithms, performance," Proceedings - IEEE INFOCOM, pp.2490-2501, 2004.