Dynamic Network Slicing for the Tactile Internet

Polachan, Kurian; Turkovic, Belma; Prabhakar, T. V.; Singh, Chandramani; Kuipers, Fernando

DOI
10.1109/ICCPS48487.2020.00020

Publication date
2020

Document Version
Final published version

Published in
Proceedings - 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems, ICCPS 2020

Dynamic Network Slicing for the Tactile Internet

Kurian Polachan*, Belma Turkovic§, Prabhakar T. V.†, Chandramani Singh‡, and Fernando A. Kuipers¶
* † ‡ Indian Institute of Science, Bangalore, India
§ ¶ Delft University of Technology, Delft, the Netherlands
Email: {*kurian, †tvprabs, ‡chandra}@iisc.ac.in, {§b.turkovic-2, ¶f.a.kuipers}@tudelft.nl

***Abstract*—"Tactile internet" refers to a network that can support real-time interactions between human operators and remote cyber-physical systems as if they were near to each other. For this, the network should support ultra-low latency communication, often referred to as the $1ms$ challenge. However, we observe that network requirements, such as latency and bandwidth, of tactile internet based cyber-physical systems or Tactile Cyber-Physical Systems (TCPS) are not static: they severely fluctuate over time. Therefore, for TCPS, static provisioning of network resources is sub-optimal. For optimal utilization of network resources, we propose a mechanism to, per TCPS flow, dynamically create, destroy and switch network slices, based on the network resources needed at that time. Our solution consists of two main components. First, we develop a clustering algorithm to determine the slices and their specifications required to support a TCPS flow. Second, we leverage Software-Defined Networking (SDN) and P4-programmable switches to enable on-the-fly provisioning and switching of these slices.**

I. Introduction

A teleoperation system consists of a human operator controlling a remote robot, called teleoperator, over a communication network. Here the communication network transports the kinematic (position/velocity) commands from the operator-side to the teleoperator-side and feeds back audio, video and/or haptic data in the reverse direction. Although teleoperation systems have been around for decades, the presence of significant end-end latencies and packet drops in communication networks restricts human operators from performing control actions that demand higher operator dynamics (e.g., hand speed) and/or cover large distances [1], [2]. Especially at higher operator's dynamics, the teleoperation systems can either become unstable or result in severe operator-side cybersickness causing severe degradation to the end-user experience [2]. For critical applications, like telesurgery, control loop instability can cause the remote-side robot to go out of control from the surgeon's hand resulting in injuries to patients. Similarly, cybersickness, which occurs when feedback delays are significant and noticeable to the human operator, can result in physical and physiological effects in human operators preventing them from extended use of the teleoperation system [3], [4]. To prevent the aforementioned problems, networks characterized by very low end-to-end latency (ideally 1ms), and very high packet reliability (approx 99.999%), termed tactile internet [2], are needed. We refer to these envisioned teleoperation systems as Tactile Cyber-Physical Systems (TCPS).

However, we observe that in many TCPS applications, operator dynamics widely fluctuate and, most of the time, stay away from their peak value. Consequently, at lower dynamics the network requirements of these flows can be relaxed, allowing for higher latencies and less bandwidth. For instance, consider Figure 1 showing the left-hand movements of a surgeon performing a suturing operation using a da Vinci Surgical System [5]. We find that the operator's dynamics (i.e., the hand speed of the surgeon) along the x axis vary throughout the procedure and stay below their peak value (≈ 0.13m/s) most of the time. If the network provider were to statically allocate network resources to such an application, following the peak value of operator dynamics, the performance would be guaranteed during the lifetime of the flow, but the assigned network resources would be underutilized most of the time.

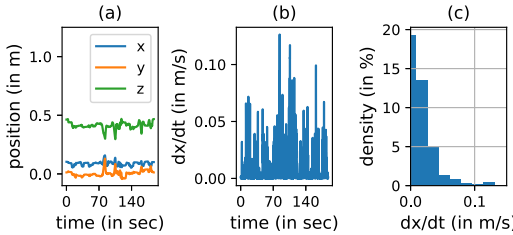


Fig. 1: (a) (x, y, z)-positions tracking the left hand movements of a surgeon performing the surgical task called suturing using a da Vinci Surgical System. (b) Dynamics in x-position. (c) Histogram of the dynamics in x-position.

As a solution, we propose a dynamic resource allocation scheme where the instantaneous operator's dynamics govern the amount of allocated network resources. To achieve that, we leverage an emerging networking architecture, called *network slicing* [6]. This concept envisions that network providers can divide a single physical network into several virtual networks, each tailored to a class of application requirements. By exploiting this concept, our scheme routes each TCPS flow through a set of network slices (each tailored to different dynamics) that are created and destroyed on-the-fly. Our scheme offers a solution to several tasks: (i) clustering of the operator's dynamics, where each cluster is mapped to a resource vector, (ii) profiling of resource requirements, which determines the fraction of time each resource vector is required and helps to check the availability of resources before starting the application, (iii) real-time creation and destruction of network slices (switching across resource vectors) according to the current operator's dynamics.

Additionally, since creating new slices, and switching flows between slices, consumes non-negligible time (potentially in the order of tens of ms), adversely impacting TCPS applications, we propose (i) identifying *a priori* the slices and their specifications before the beginning of the TCPS application,

2642-9500/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCPS48487.2020.00020

(ii) using a Software-Defined Networking (SDN) controller to pre-compute the paths for the identified slices, and (iii) using P4-programmable switches for real-time resource allocation and switching of slices [7]. This way we make sure that the slice switching latency is minimized and meets the latency constraints of a TCPS application. Finally, we show that our approach not only leads to a more efficient network resource utilization, but also results in enormous savings in aggregate switch memory, while keeping the slice switching latency in control.

A. Key Contributions

- *Clustering Algorithm for TCPS*: We propose a clustering algorithm to design network slices for TCPS applications. The slices designed using the proposed clustering algorithm maximize the network resource utilization (or minimize the network cost), while maintaining the required application quality.
- *Real-Time Slice Management Framework*: We propose an on-the-fly network slice provisioning system. Every time a slice switch occurs, the previously used network slice is destroyed and a new one created directly in the data-plane. This way the network resources assigned to the TCPS application are effectively scaled up or down.

B. Related Work

Traditional networks were static and offered very limited possibilities for Quality of Service (QoS) provisioning. When assigning resources, network providers were left with two choices: to either over-provision, i.e. reserve too many resources, but keep the average utilization low, or to under-provision, i.e. increase the utilization, but reserve insufficient resources to support the application at its peak load, potentially degrading the end-user experience.

By splitting the control-plane from the data-plane, SDN enabled more flexible, fine-grained QoS provisioning [8] and it facilitated the concept of network slicing, where, on top of a common physical infrastructure, different virtual networks, tailored to different traffic needs, can be created, enabling the coexistence of diverse services [9], [10], [11]. Several slicing frameworks, such as FlowVisor [12] and FlowN [13], have been proposed over the years. However, they focus on creating isolation between the slice tenants and do not address the specific and very strict per packet QoS requirements of a TCPS application.

Additionally, many SDN frameworks enabling resource reservations were proposed over the years [14], [15], [16], [17], [18], [19]. However, the time-varying resource requirements of flows were not taken into account. Moreover, even if dynamic rerouting and resource scaling would be added to these SDN frameworks, they would still violate the constraints (e.g., end-to-end latency) of a TCPS application. To add new or modify existing rules for the network, SDN controllers need to be informed first, resulting in a significant re-configuration latency penalty. Furthermore, variable latency between the controller and the switches can lead to inconsistencies in the switch tables. If all the switches are not updated at exactly the same time, packets can get dropped (due to non-existing rules) or processed by outdated rules potentially violating the service-level agreement between the network provider and the slice tenant [20]. To solve the aforementioned problem, programmable switches, along with domain-specific programming languages, such as P4, can be used [7]. They offer the possibility to respond quickly to traffic changes directly from the data-plane, while the data-packets are being processed [21].

Specific to TCPS flows, [22], [23], [24] list the benefits of network slicing to guarantee lower latency, higher reliability and security. However, they consider the network slices to be static, i.e., the lifetime of these slices extends over the full duration of a TCPS flow. While the authors of [25], [26] discuss dynamic-aware routing of TCPS flows, exploiting the burstiness in the packet arrival rates, it is limited to latency and bandwidth optimization in radio access networks alone. It also does not take into account the varying dynamics of the TCPS operator. Several works examined how to use time series methods to cluster/classify human hand motion in a general context (see [27], [28]). However, a framework to adopt these methods in the context of TCPS and dynamic network slicing, i.e., how to use these algorithms to design slice specifications for given TCPS quality requirements, is still missing.

In the context of wireless embedded systems, several works have attempted to guarantee communication quality to ensure the stable control of remote devices. Some of these works also address how to dynamically change the communication path between a controller and a remote device without compromising on stability (see [29] and references therein), a concept that bears some similarity to the concept of dynamic network slicing discussed in this paper. These works, however, focus only on embedded wireless systems working on a limited number of hops. In particular, they are not tried and tested on IP network components such as switches, nor do they account for the peculiarity of historical data to design slices.

C. Outline

We organize our paper as follows: In Section II, we describe the main building blocks of our system, enabling dynamic resource provisioning and routing of TCPS flows. In Section III, we propose a clustering algorithm to estimate the number and specifications of network slices for TCPS applications. In Section IV, we describe how to use programmable switches to realize real-time slice management. In Section V, we evaluate our proposed system. We conclude the paper in Section VI.

II. System Overview

Figure 2 shows our proposed TCPS architecture to enable dynamic network slicing. The architecture consists of two main blocks, the edge controller and a real-time slice management framework. In this section, we describe the functions and design details of these blocks.

A. Edge Controller

The edge controller resides at the operator-side. It serves the following two purposes.

- Before the start of every TCPS flow, the edge controller requests network slices of specific bandwidth and latency from the SDN controller. The slice configuration block of

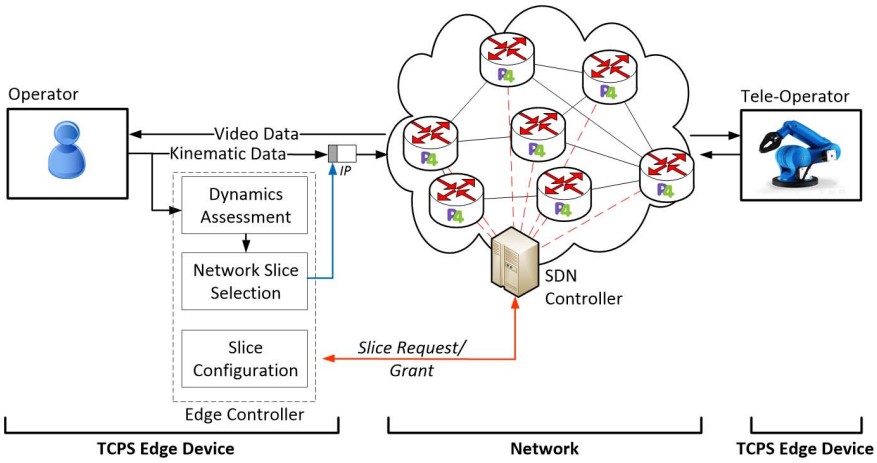


Fig. 2: Proposed TCPS architecture to enable dynamic network slicing.

the edge controller performs this request. Only if this request is granted and acknowledged by the SDN controller, the TCPS application is allowed to start sending data.

- At run-time, the edge controller (through the network slice selection block) decides which slice to use based on the current operator dynamics (e.g., hand speed) measured by the dynamics assessment block. The edge controller then embeds this slice information in the IP header of the TCPS packet before sending it out onto the network.

For the slice configuration block to request slices and for the network slice selection block to decide which slice to use, the blocks should a priori know the number of slices, their specifications, and how to map current operator dynamics to an available slice. To cater to these needs, we propose a custom unsupervised clustering algorithm. We use this algorithm to find clusters in hand-speed values, which we obtain from different trials of the TCPS operation. We map each cluster to a slice and then use the cluster statistics to derive the slice specifications and also the dynamics-to-slice map. We describe this algorithm in Section III.

Figure 3 shows a sample implementation of the dynamics assessment and the network slice selection blocks.

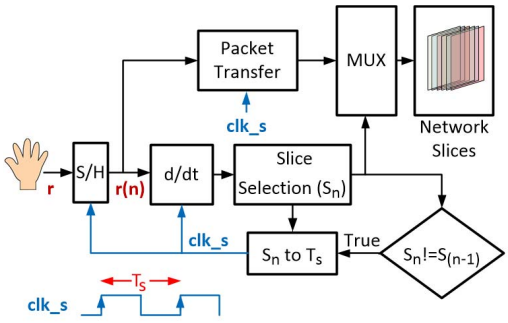


Fig. 3: Design of the dynamics assessment and network slice selection blocks. S/H - sample and hold block.

The dynamics assessment block measures the operator's dynamics (i.e., hand speed) from the operator's hand position, $r \triangleq r(t)$, at every rising edge of the sampling clock clk_s (of period T_s), using an ideal derivative block (d/dt).

The network slice selection block selects a suitable slice to route the TCPS flow based on the measured dynamics. Also, based on the selected slice, it modulates the frequency of clk_s. Modulating the frequency of clk_s serves two purposes. First, it helps in optimizing the data transmission rate (and the bandwidth), and second, it helps in accurate measurement of the operator dynamics over a wide range.

We change clk_s during slice switches. We set the period of clk_s proportional to the Round Trip Time (RTT) of the slice in use, since, RTT of the slice in use is directly dependent on the operator dynamics (p is an appropriate proportionality constant, $0 < p < 1$). However, this setting can cause TCPS packets to experience RTT overheads during slice switches, particularly when switching from a slice of low RTT to high RTT. To minimize this overhead, we can tune p. Note that a lower p minimizes switching overhead, but increases the rate of data transmission and may also degrade the quality of dynamics measurements. In our work, we set $p = 0.1$.

We model the overhead in RTT at sampling instance $n \triangleq nT_s$ using the following equation. We assume that the overhead is negligible when RTT of the current slice, $RTT(S_n)$, is higher than the RTT of the previous slice $RTT(S_{n-1})$.

$$RTT_{overhead}(n) = \begin{cases} 0, \ if \ S_n = S_{n-1} \\ 0, \ if \ RTT(S_n) > RTT(S_{n-1}) \\ p \times RTT(S_{n-1}), \ \text{otherwise} \end{cases}$$

Remark: In Figure 3, we use the same clock for sampling, dynamics assessment and data transmission, assuming that all three are executed at the same rate. However, the sampling rate can be any value higher than the dynamics assessment rate, which in turn can be higher than or equal to the data transmission rate.

B. Real-Time Slice Management Framework

Our real-time slice management framework enables on-demand provisioning of network resources, i.e. the resources are made available on-the-fly only when needed, similarly to computing resources in cloud environments. This way, network utilization is constantly high, as no resources are over-provisioned, while the QoS is guaranteed during the whole lifetime of the TCPS application, i.e. the network is adapting its behaviour to match the current TCPS application's needs. The scaling process is done at the expense of any other non-TCPS flows, which get the remaining resources in the network. To enable this on-the-fly slice management, our slice management framework consists of two main components:

- A *central SDN controller* (control-plane) that has a global view of the whole network, as well as the currently present traffic. For every new TCPS flow, it finds the appropriate routes that satisfy the end-to-end slice requirements according to the current global network state.
- *Slice configuration protocol* (data-plane), deployed using the network programming language P4 [7], to create/destroy the slices on-the-fly in the data-plane using the information from the edge controller and the pre-computed inputs from the SDN controller. It provides a fast update loop, enabling the switches to react quickly to the changes in the application dynamics without the need to contact the SDN controller. Thus, the slices are created/destroyed at run-time, with negligible latency overhead.

Every time a new TCPS flow is initiated, the edge controller forwards the slice specifications it wishes to use to the central SDN controller (Figure 2). The SDN controller, with its up-to-date overview of the current network state, uses these slice specifications to calculate the routes that satisfy the QoS (e.g., delay and bandwidth) requested for each slice. Finally, these pre-calculated routes are forwarded and stored in the first switch on the path (i.e., the edge switch to which the TCPS Edge Device is connected) and ready to be used by the second component of our solution: the *slice configuration protocol*.

The slice configuration protocol enables the creation/destruction of network slices on-the-fly, directly in the data-plane, using just the routes stored in the edge switches. Every time the TCPS dynamics change, i.e. a different slice is requested by the edge controller, the assigned network slice (with certain delay and bandwidth constraints) is destroyed, and the resources freed to be used by other services. At the same time, a new slice, corresponding to the new dynamics, is created. To ensure that both creation/destruction actions are executed in real-time, our protocol *enables the data-packets to program the data-plane as they pass through the switches*.

For example, to create the slice, the first switch appends a special *slice configuration header*, containing among other things the pre-calculated route, to the original TCPS packet that was received from the edge controller. By reading this special header, every switch in the path configures/updates its forwarding and bandwidth reservation rules to correspond to the currently requested slice (as explained further in Section IV). Thus, as the switches in the path process the first TCPS packet, a new slice is configured and ready to be used. All the subsequent TCPS packets follow this first packet and are routed using the newly created rules, which prevents any temporary inconsistencies. In addition, every time a new slice is created, a similar packet is sent to destroy the slice that was previously used. Every switch that processes this packet, deletes the configured rules and frees the allocated bandwidth. This way, the number of installed rules is minimized and the bandwidth released to be used by other services. Every time the edge controller detects a new change in dynamics, the whole process repeats.

During the lifetime of a TCPS application, at any moment only one slice per application is configured/used in the switches. Thus, although the rules (processing as well as bandwidth reservations) for multiple slices are calculated by the controller, only one set of rules (corresponding to the currently used slice) is active. As a consequence, resources assigned to slices that are not currently used are free and have no impact on other traffic present in the network.

When calculating the routes, the SDN controller makes sure that, if the resources of other slices are to be requested (due to a change in dynamics), they would be available instantaneously. To do so, the controller keeps track of the amount of bandwidth allocated on every link to all slices of TCPS flows. Hence, new TCPS flows are only admitted through switches that will have enough resources available. As a consequence, during the entire duration of any TCPS flow, resource availability is always guaranteed. However, this approach also limits the maximum number of TCPS flows that can be present in the network at the same time. Depending on the exact TCPS application, this requirement can be relaxed by providing a trade-off between the maximum number of flows and the probability of a QoS degradation.

III. Clustering Algorithm for TCPS

In the previous section, we proposed using an unsupervised clustering algorithm to determine the number of network slices and their specifications. In this section, we motivate the need and describe the design of our proposed clustering algorithm, guided by the following two performance metrics.

A. Performance Metrics

1) Visual Performance Index: We model and quantify cybersickness using a visual performance index E. We define E as the percentage of time during a given TCPS task for which the visual error, i.e., the error between the position of the robot displayed on the operator-side screen and the current hand position of the operator, is within $1mm$. $E = 100\%$ implies that the error during a TCPS task is always within $1mm$. Since errors of less than $1mm$ are not noticeable to human operators, this condition eliminates the presence of cybersickness. A TCPS application may have a minimum desired visual performance index, E_{spec}. Applications that demand stringent requirements on visual quality may require a higher value of E_{spec}.

For a TCPS, the required RTT to avoid cybersickness depends on the operator dynamics, i.e., hand speed. If the hand speed is at the natural maximum limit of human hand speed, i.e., $1m/s$, we need an RTT of less than or equal to

$1ms$ to keep the visual error within $1mm$ [1] (See Figure 4). However, at lower hand speeds, we can allow higher RTT, e.g., if the hand speed is 0.1m/s, any RTT less than or equal to $10ms$ ensures that the error is within $1mm$. In the context of dynamic network slicing, we use the following equation to decide RTT for a slice S.

$$RTT(S) = \frac{1}{V_{max}(S)}\ ms \tag{1}$$

In the above equation, $V_{max}(S)$ is the maximum hand speed (in m/s) that the slice S is required to support. Note that even if individual slices maintain these RTTs, E will still be less than 100%. This is due to the presence of RTT overhead caused by slice switching.

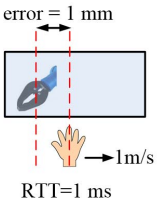


Fig. 4: At a hand speed of $1m/s$, error is $1mm$ when RTT is $1ms$ [4].

2) Price Index: We quantify the cost of the required network slices using a price index, I. Generally, rates (costs per unit time) of network slices that offer high-bandwidth, low-latency routes are high. Considering this, we assume the rate of a slice S is proportional to its bandwidth $B(S)$ and inversely proportional to its RTT $RTT(S)$, specified by the function, f. Further, the price of using a set of slices depends on the fraction of time the individual slices in the set is used. We use the following equation to define the price index.

$$I = \sum_{S} B(S) \times f(RTT(S)^{-1}) \times U(S) \tag{2}$$

In the above equation, $U(S)$ is the fraction of time a slice S is used and is referred to as the utilization factor of S. Also, $f(\cdot)$ is an increasing function. In our work, we consider $f(x) = x$. For many use cases of TCPS, the bandwidth is consumed mainly by video feedback [30], [31]. We can optimize this bandwidth by modulating the frame rate of the feedback video as a function of operator hand speed, i.e., we increase the frame-rate when operator hand speed is higher and decrease it when the hand speed is lower. In our work, we consider the case where the RTT of the slice decides the frame-rate, i.e., we change the frame-rate every time a slice switch happens. In this scenario, $B(S)$ will be inversely proportional to $RTT(S)$. See Appendix A for details.

B. Objective

Our goal is to design network slices and to determine their utilization factors for a specific TCPS application. Observe that using low bandwidth and high latency slices may result in a visual performance index E less than E_{spec}. Using high bandwidth and low-latency slices leads to better E, but also a higher I. Given a set of network slices, E and I also depend on the slices' utilization factors. Our objective is to determine the slices, and their utilization factors, that *minimize I subject to $E \geq E_{spec}$*.

Below we propose a custom clustering algorithm to cluster the operator's hand-speed values. In Appendix D, we elaborate why popular clustering algorithms such as k-means [32] and mean-shift [33] do not suit our requirement.

Remark: We can configure and use network slices for tactile applications even without a priori knowledge of historical data or dynamics. However, we use historical data to arrive at a "good" number of slices and "good" slice specifications. Our proposed algorithm yields slice specifications such that the number of switchings is not excessive and the slices used are also close to the "least required" most of the time.

C. Design

Our approach consists of clustering operator's hand speeds obtained from the application's historical data sets and then to map each cluster to an appropriate network slice. Cluster boundaries determine the slices' characteristics, whereas hand speed distribution determines the slices' utilization factors. We can formally describe the problem as follows. Let the historical dataset contain N hand-speed values, $v(n)$ $(n = 1, \ldots, N)$. Let $F(v)$ be the distribution of the hand speeds

$$F(v) = \frac{|\{n : v_n \leq v\}|}{N}.$$

Let there be K hand-speed clusters, k^{th} cluster being (θ_{k-1}, θ_k) with $1 \leq k \leq K$, notice that $\theta_k = V_{max}(k)$. We take $\theta_0 = 0$ and $\theta_K = v_{max}$. The cost of clustering can be expressed using the price index I (assuming $f(x) = x$ in (2)) as:

$$I = \sum_{k=1}^{K} B(k) RTT(k)^{-1} U(k)$$

where the utilization factor of the k^{th} cluster $U(k) = F(\theta_k) - F(\theta_{k-1})$. Further, the bandwidth of the k^{th} cluster, $B(k)$, should be inversely proportional to $RTT(k)$ (see the last paragraph of Section III-A2). We let κ_1 be the proportionality constant. Hence, for all $k = 1, \ldots, K$,

$$B(k) RTT(k)^{-1} = \kappa_1 RTT(k)^{-2} = \kappa_1 \theta_k^2$$

where the last equality follows from (1). Thus,

$$I = \kappa_1 \sum_{k=1}^{K} \theta_k^2 U(k) = \kappa_1 \sum_{k=1}^{K} (F(\theta_k) - F(\theta_{k-1}))\theta_k^2$$

The optimal clustering problem entails determining K and $(\theta_0, \theta_1, \ldots, \theta_K)$ to minimize I, while ensuring that E exceeds E_{spec}. For fixed $(\theta_0, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta_K)$, we can unilaterally change θ_k to minimize I. More precisely, we can set

$$\begin{aligned} \theta_k &= \underset{\theta \in (\theta_{k-1}, \theta_{k+1})}{\arg\min} \sum_{k=1}^{K} (F(\theta_k) - F(\theta_{k-1}))\theta_k^2 \\ &= \underset{\theta \in (\theta_{k-1}, \theta_{k+1})}{\arg\min} \ (F(\theta) - F(\theta_{k-1}))\theta^2 - F(\theta)\theta_{k+1}^2. \end{aligned}$$

Also, the visual performance index, E, degrades with the number of slice switches from low latency slices to high latency slices. For a cluster boundary θ, the normalized number of such switch-overs is,

$$s_\theta = \frac{|\{n : v(n) \leq \theta < v(n+1)\}|}{\sum_{\theta'} |\{n : v(n) \leq \theta' < v(n+1)\}|}.$$

These motivate the following heuristic to arrive at a "good" clustering: Start with the number of clusters $K = 2$ and arbitrary cluster boundaries $(\theta_0, \theta_1, \ldots, \theta_K)$. Randomly pick a boundary θ_k $(0 < k < K)$ and update it as

$$\theta_k = \underset{\theta \in (\theta_{k-1}, \theta_{k+1})}{\arg\min} (F(\theta) - F(\theta_{k-1}))\theta^2 - F(\theta)\theta_{k+1}^2 + \beta s_\theta$$

where β is a design parameter. Using higher β gives more weight to visual performance compared to the price. Continue such unilateral updates until the cluster boundaries converge. Let the resulting boundaries be $(\theta_0^{K,\beta}, \theta_1^{K,\beta}, \ldots, \theta_K^{K,\beta})$. Record the corresponding visual performance index $E^{K,\beta}$. In Appendix B, we illustrate the dependence of $E^{K,\beta}$ and $I^{K,\beta}$ with β. If $E^{K,\beta}$. exceeds E_{spec}, repeat this exercise for $K+1$ clusters else output $K-1$ as the final number of clusters and $(\theta_0^{K-1,\beta}, \theta_1^{K-1,\beta}, \ldots, \theta_{K-1}^{K-1,\beta})$ as the final cluster boundaries ($\theta_0^{K-1,\beta} = 0$ and $\theta_{K-1}^{K-1,\beta} = v_{\max}$). We formally describe the algorithm as follows.

Algorithm 1 Determining Clusters, Given β and E_{spec}

```
1: procedure CLUSTER(β, E_spec)
2:     K ← 2
3: loop:
4:     compute E^{K,β}, (θ_0^{K,β}, ..., θ_K^{K,β})
5:     if E^{K,β} > E_spec then
6:         K ← K + 1
7:         goto loop
8:     else
9:         return K − 1, (θ_0^{K−1,β}, ..., θ_{K−1}^{K−1,β})
```

Discussion: The proposed algorithm does not necessarily give the optimal cluster specifications. Given that the speeds are quantized into Q values and the number of clusters to be used is K, computing optimal cluster specifications following a brute-force approach requires $\Theta(Q^K)$ computations. As we have to compute optimal cluster specifications for several values of K, the brute-force approach is intractable. We have found in our simulations that the proposed algorithm converges much quicker. Moreover, it leads to cluster specifications that are close to the optimal cluster specification. We demonstrate this in Appendix C using a dataset from the da Vinci Surgical System.

IV. Slice Configuration Protocol

To switch a TCPS flow f, between two TCPS endpoints, from slice A (with a pre-calculated route r_A) to slice B (with a pre-calculated route r_B) two actions need to be performed: (i) creation of a new slice B, i.e. updating the forwarding rules and allocating bandwidth for flow f on all switches on path B, and (ii) deleting the old slice A, i.e. deleting the rules and freeing the allocated bandwidth for flow f on all switches on path A. To ensure that (i) and (ii) are executed in real-time, we designed a new slice configuration protocol that enables the data-packets to create/destroy a network slice as they pass through the switches.

To perform the above-mentioned actions, our slice configuration protocol uses two different messages (shown in Figure 5): (1) "Slice setup" message to create a new slice and (2) "Slice delete" message to delete the previously used slice. The field *Ports array* represents the pre-calculated route r_B (r_A) as a sequence of output ports from all the switches on path B (A). The size of this field depends on the number of switches used on the path (specified by the field *Header Length*). *Slice ID* corresponds to the bandwidth constraint needed on the path. Based on this parameter, the switches will reserve the needed amount of bandwidth as the packet passes through them.

In Section IV-A, we describe how the pre-calculated routes (by the central SDN controller) are used in the edge switches, while in Section IV-B we describe how we use our protocol to change routing entries on the intermediate switches (i.e., in the network core).

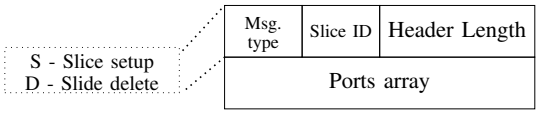


Fig. 5: Slice Configuration Protocol header.

A. Processing on the Network Edge Router

When a first packet belonging to a TCPS flow is received at the network edge, the router, based on the flow identifier (source and destination IP addresses, protocol field, and source and destination ports) and the ToS (Type of Service) field, inserts a new "slice setup" header between the Ethernet and the IP headers. For example, as shown in Figure 6, when a packet with flow identifier 35 is received, the router checks the table containing the headers for all the potential slices this flow can use (shown on the left in Figure 6, for memory overhead calculation see Appendix E).

Since the ToS field of the packet is set to 1, the slice with ID 1 is chosen and a header containing the values $1-4$ is inserted between the Ethernet and IP headers. The header field 1 represents the $Slice\ ID$ (Figure 5) and corresponds to a certain predefined amount of bandwidth that will be allocated at all the switches for this flow. Similarly, the next header (value 2) represents the length of the route r_B and indicates that two intermediate switches are present between the TCPS Edge Devices. All other values represent port numbers used at the intermediate switches (port 3 at the first intermediate switch and port 4 at the second intermediate switch). In addition, the Ethernet type is changed to a specific value (0xBB) to indicate the presence of the new header.

To delete the previously used slice, the same procedure is used. An additional packet, containing "Slice delete" header is sent on the route r_A (determined by the flow identifier and the previously used ToS field).

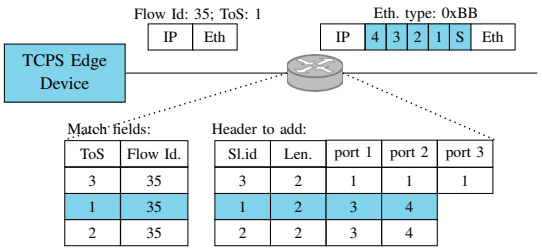


Fig. 6: Processing on network edge. "Slice setup" header is inserted between the Ethernet and IP headers.

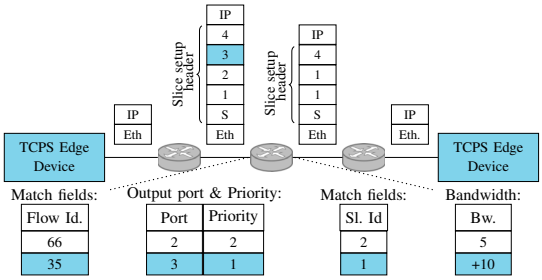


Fig. 7: Processing of a "slice setup" message in the network core. New rules are added as the packet passes through the switch.

B. Processing in the Network Core

When a packet with the *Slice Configuration Protocol* header is received by the next switch in the path, additional bandwidth is reserved, and the forwarding table updated (Figure 7). The first intermediate switch, after processing the "Slice setup" header, inserts a new entry in the forwarding table (shown on the left bottom corner in Figure 7). All the subsequent packets, with flow identifier equal to 35 and belonging to slice 1 (indicated by ToS equal to 1), are processed by this rule and output to port 3. In addition, the resources reserved for slice 1 are increased by 10 units of bandwidth (shown on the right bottom corner in Figure 7), the used port number field (with the value 3) removed, and the header length reduced by 1 to represent the number of switches left to configure. Similarly, when this packet is received by the second intermediate switch, a new rule (to output packets to port 4) is inserted and the bandwidth allocation table updated. All subsequent packets with the flow identifier 35, are processed by these newly installed rules (e.g., output to port 3 on the first switch and port 4 on the second switch), preventing any temporary inconsistencies. When a slice needs to be deleted, a similar process occurs on every switch in the network. The only difference is that, when a switch detects a "slice delete" header, network resources would be released (or scaled down) and processing rules removed.

V. Evaluation

In this section, we evaluate (i) the performance of our clustering algorithm, (ii) the benefit of dynamic network slicing, and (iii) the performance of our programmable network in managing slices in the presence of traffic.

A. Performance of our Clustering Algorithm

In Table I, we compare the performance of Algorithm 1 with k-means and mean-shift using the hand speed time series data from the da Vinci Surgical System database [5]. The datasets we use for clustering (Suturing-B001, B002, B003, B004, B005), correspond to a surgeon performing five trials of a suturing operation ($v_{max} \approx 0.18m/s$). We see that Algorithm 1 (Q set to 200) presents significant cost savings in comparison to mean-shift and performs better than k-means for an E_{spec} of 95%. Besides, by varying β, Algorithm 1 demonstrates the ability to tune E, a feature that k-means and mean-shift do not possess.

In Appendix D, we compare the performance of Algorithm 1 and k-means using hand speed time series that have multiple peaks in their histogram. We show that Algorithm 1 distinctively outperforms k-means in such scenarios.

Remark: For comparison, k-means and mean-shift algorithms were tuned for optimal values of K, which minimize I and ensure $E > E_{spec}$.

TABLE I: Performance comparison of Algorithm 1 with k-means and mean-shift for five trials of a suturing operation. E and I values averaged over the five trials are used for comparison.

	k-means	mean-shift	Algorithm 1		
			β=0	β=0.05	β=0.1
$E(average)$	96.9%	96%	97.1%	98.1%	98.6%
$I(average)$	361.9	427.0	337.7	374.1	412.5
K	4	5	4	4	4

B. Performance of Dynamic Network Slicing

Dynamic network slicing can result in significant cost savings with acceptable degradation in visual performance. To demonstrate this, we run Mininet simulations to compare E and I with and without dynamic network slicing [34]. We use the cluster specifications generated by Algorithm 1 in Section V-A to design the slices for dynamic network slicing ($E_{spec} = 95\%$, $K = 4$). Figure 8 shows the evaluation setup. The four links with RTTs $117.4ms$, $29.5ms$, $13.4ms$, and $5.7ms$ simulate four network slices. To simulate the TCPS flow, we first packetize the data corresponding to the surgeon's hand position (from Suturing-B001 dataset) along with Slice-ID. These packets are then sent from host h_1 to host h_2. The edge switches identify the Slice-ID in the incoming IP packets and decide the link to route the flow. h_2 receives these packets and echoes them back to h_1 to simulate the feedback (remote-side robot's position).[1] For experiments without dynamic network slicing, we choose a single point-to-point link with RTT of $11.22ms$ between h_1 and h_2. The RTT of $11.22ms$ corresponds to the largest among the RTTs that result in an E of 100%.

Table II captures the designed and measured E and I values with and without dynamic network slicing. Recall that with

[1]Note that we simulate the feedback without considering physical devices at the teleoperator side, such as a robot or a video camera, because, in the experiments, our focus is only on the communication network.

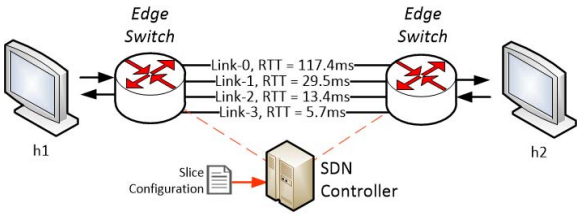


Fig. 8: Simulation setup to evaluate dynamic network slicing.

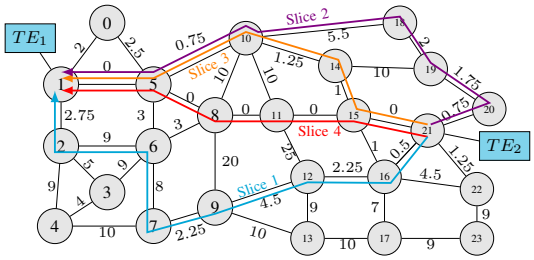


Fig. 9: USNET topology. Link delays are in ms.

dynamic network slicing, the links were designed to meet an E_{spec} of 95%. The design thus suggests a decrease in E by 5% (100% to 95%). The decrease in E measured, however, is close to 10%. We can attribute this difference to setup overheads and the simplified overhead model we used for designing the slices. The measured decrease is acceptable for many TCPS applications. For TCPS with stringent quality requirements, the decrease in E can be controlled by designing slices for a higher value of E_{spec}. We measure I via a separate experiment. The experimental setup used for measuring I is the same as used for measuring E, except we modulate the bandwidth of the TCPS flow as a function of the slice RTT to simulate the varying bandwidth requirement of the video feedback (see Section III-A). In practice, the bandwidth of the TCPS flow varies due to video frame-rate modulation. However, for the experiment, to simulate the bandwidth profile, we fix the frame-rate and adapt the UDP packet size at every sampling instant as a function of the slice RTT. We do this because the dataset corresponding to the suturing operation is available for a constant sampling rate of $30Hz$. Measurements show a reduction in I by 77% with dynamic network slicing, indicating excellent cost savings.

TABLE II: Variation in E and I with and without dynamic network slicing.

Dynamic Slicing	E		I	
	Designed	Measured	Designed	Measured
Disabled	99.66%	99.62%	778	742
Enabled	95.77%	90.48%	230	167

C. Network performance

Experiment setup: To evaluate our slice switching protocol, we emulate the USNET topology (Figure 9), using Mininet with the P4 software switch (behavioural model [35]). Multiple TCPS flows are generated between two TCPS Edge Devices TE_1 and TE_2 and delay, jitter, and throughput are measured in both directions. After receiving a packet, TE_2 bounces it back to TE_1 using the same slice. As in V-B, TCPS flows are routed through 4 slices with RTTs equal to $117.4ms$, $29.5ms$, $13.4ms$, and $5.7ms$. Thus, for each TCPS request, a set of four routes is calculated by the SDN controller (example set shown in Figure 9). To simulate the traffic belonging to other services, additional TCP traffic is generated using iperf between different switches in the network. All measurements were repeated 30 times [5].

Traces: Each TCPS trace is 98 seconds long and contains data from the da Vinci Surgical System database. For each packet in this trace, the ToS header is set using the clustering algorithm described in Section III and indicates the slice the packet should be processed in. Packet lengths were fixed to $140B$ (including the headers) and sent at a rate that depended on the current slice (one packet each $RTT/2$). Thus packets belonging to slices with stricter latency requirements (and higher dynamics) are also sent at a faster rate, creating sudden bursts of TCPS traffic in the network.

Comparison baslines: Our approach (P4 + Slicing) is compared to (i) an approach that uses an SDN controller to compute and install a new slice (both route and bandwidth reservation) each time a switch occurs (SDN + Slicing), and (ii) an approach that does not use slicing, but provides QoS guarantees (No Slicing) by reserving either the maximum or average bandwidth needed by the flow.

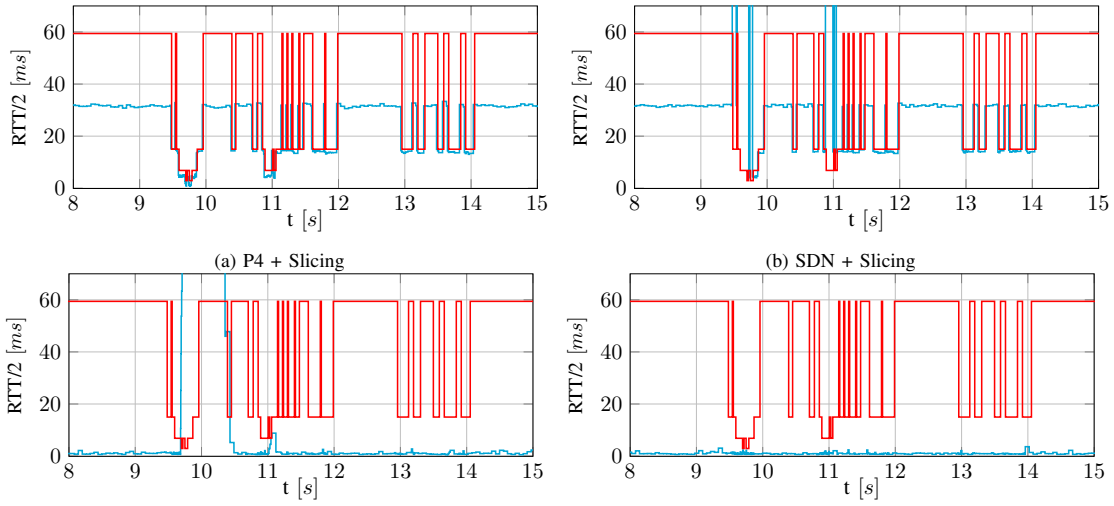
1) Switching Delay: To demonstrate the advantages of our slice configuration protocol, we evaluate the time needed to switch between two different slices. This switching delay is measured as the difference in the delay between the first (with the additional Slice Configuration header) and second packet processed by the slice. Compared to the solution that uses a centralized controller to reroute packets (SDN + Slicing), our solution was able to reduce the switching delay significantly ($0.34ms$ on average compared to $72.68ms$ on average). The main reason for the huge increase in performance is the fact that the (SDN + Slicing) solution forwards the first packet to the controller introducing a significant delay penalty, while our solution enables the packets to program the data-plane.

2) Performance Guarantees: Figure 10 and Table III show that, when using our proposed (P4+Slicing) solution, the network can guarantee the performance at any moment during the duration of the TCPS flow. The blue line, representing the one-way delay between the two TCPS edges (TE_1 and TE_2) is under the red line, representing the delay constraint of the system corresponding to the current perceived dynamics (i.e., the maximum allowed delay each packet can experience), during the whole flow duration. When switching from a high to low RTT slice, packet reordering can occur at the TCPS edge, due to the difference in the slice's RTTs. However, this will not degrade the system's performance, as the later-arriving, outdated packets can simply be discarded.

In contrast, in the scenario (SDN+Slicing), the first packets processed by the slice experience a significant increase in delay (Figure 10). This effect is more significant for low-latency slices (Slice 3 and 4), in which the delay constraints

TABLE III: Maximum and average RTT values measured for each slice for the USNET topology. (metrics calculated for 30 different runs). Values are in ms.

		No Additional TCP Traffic						With Additional TCP Traffic							
		P4 + Slicing		SDN + Slicing		No Slicing (Max. Reserved)		P4 + Slicing		SDN + Slicing		No Slicing (Max. Reserved)		No Slicing (Avg. Reserved)	
	Slice RTT constraint	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT	Avg. RTT	Max. RTT
1	117.48	65.37	67.00	32.50	44.13	-	-	63.07	66.24	63.10	67.64	-	-	-	-
2	29.51	27.46	29.28	29.28	36.13	-	-	27.65	29.39	39.28	91.92	-	-	-	-
3	13.48	10.71	12.77	5.77	24.75	-	-	9.92	12.67	57.31	998.74	-	-	-	-
4	5.75	3.43	5.74	5.86	21.46	4.53	18.38	4.11	5.21	67.64	781.76	2.051	15.93	138.88	1539.11



(a) P4 + Slicing

(b) SDN + Slicing

(c) No Slicing (average bandwidth reserved)

(d) No Slicing (maximum bandwidth reserved)

Fig. 10: Influence of external traffic: Observed one-way delay ($RTT/2$) for 7 seconds of the TCPS flow when 4 slices are used for the USNET topology shown in Figure 9. Blue line represents the $RTT/2$ measured between two TCPS edges, while the red line represents the $RTT/2$ constraint of the current used slice determined based on the current application dynamics.

are violated for almost every packet. Due to a higher packet rate, packets arrive at the switches faster than processed by the controller, resulting in more packets being forwarded to it (as the switches are stateless and hence unaware that a packet was already forwarded to the controller until the new route is configured), thereby flooding it. This results in significant packet reordering at the TCPS edge.

In case (No Slicing) is used and resources corresponding to the maximum possible dynamics (Slice 4) are reserved, the delay observed by the TCPS flows is the lowest possible (except for the first packet that is forwarded to the SDN controller and used to setup the route). However, if we decrease the reserved bandwidth to match the total amount of resources used by the other two solutions (*P4+Slicing*, *SDN+Slicing*), *No Slicing* behaves the worst among all the analyzed solutions, by having the highest average, as well as maximum RTT (see Table III). Especially in moments in which high QoS is required (Slices 3 & 4), and the packet rate high, TCPS packets are queued due to insufficient resources violating the RTT constraints (Figure 10).

3) Bandwidth utilization:: In our proposed (P4+Slicing) solution, scaling of the reserved bandwidth happens when the packet (having a different ToS set, indicating a switch) is processed at each switch. Thus, the amount of the bandwidth reserved for TCPS traffic corresponds to the current packet rate (see Figure 11). Similarly, in case (SDN+Slicing) is used, the SDN controller matches the reserved bandwidth to the one required by the current slice. However, the reconfiguration delay depends on the delay between the switches and the controller, potentially decreasing the performance. In case (No Slicing) is used, the resource utilization is either constantly low (max. reserved) resulting in significant over-provisioning, or insufficient to account for the high dynamics (avg. reserved) resulting in noticeable degradation to the end-users.

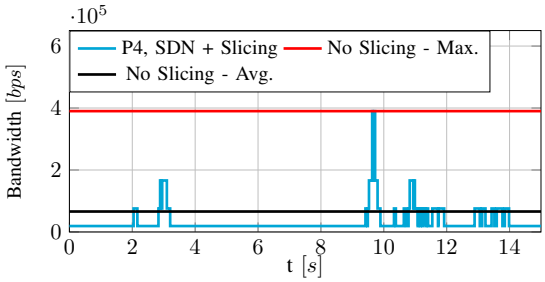


Fig. 11: Reserved bandwidth. (P4+Slicing) and (SDN+Slicing) are reserving the same amount of bandwidth.

To conclude, our (P4+Slicing) solution is able to satisfy the

latency constraints of the TCPS flow during the whole duration of the flow. In addition to guaranteeing the performance of a TCPS flow, by dynamically routing the traffic through multiple paths and by scaling the resources, it also minimizes the resources that need to be assigned by the network provider and maximizes the utilization of the resources assigned to it.

VI. Conclusion

For a Tactile Cyber-Physical System (TCPS), we observe that the dynamics of the human operator vary over time and are usually under their peak value most of the time. As network requirements depend on the operator's dynamics, we argue that it is suboptimal to route a TCPS flow through a single network slice designed to support the worst-case dynamics. Instead, we propose dynamic network slicing, a mechanism to, per TCPS flow, dynamically create, destroy, and switch slices, based on the operator's current dynamics. To determine the specification of these slices, we designed a clustering algorithm. Further, we used an SDN controller together with P4 switches to implement these slices in the network. Through experiments, we demonstrated (i) the performance of our clustering algorithm in minimizing the network cost for TCPS applications, and (ii) how our network slice management approach, while maintaining all the advantages of a typical SDN architecture, such as the centralized control and the possibilities of advanced traffic engineering, is able to provide hard QoS guarantees needed for TCPS applications.

References

[1] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, March 2014.

[2] ITU. (2014) The tactile internet. [Online]. Available: https://www.itu.int/en/ITU-T/techwatch/Pages/tactile-internet.aspx

[3] J. J. LaViola, Jr., "A Discussion of Cybersickness in Virtual Environments," *SIGCHI Bull.*, vol. 32, no. 1, pp. 47–56, Jan. 2000.

[4] K. Polachan, T. V. Prabhakar, C. Singh, and D. Panchapakesan, "Quality of control assessment for tactile cyber-physical systems," in *IEEE SECON 2019*, 2019.

[5] Y. Gao, S. S. Vedula, C. E. Reiley, N. Ahmidi, B. Varadarajan, H. C. Lin, L. Tao, L. Zappella, B. Bejar, D. D. Yuh, C. C. G. Chen, R. Vidal, S. Khudanpur, and G. D. Hager, "Jhu isi gesture and skill assessment working set (jigsaws) a surgical activity dataset for human motion modeling," *Modeling and Monitoring of Computer Assisted Interventions (M2CAI) – MICCAI Workshop*, pp. 1–10, 2014.

[6] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, May 2017.

[7] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[8] D. Kreutz, F. M. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[9] H. Zhang, N. Liu, X. Chu, K. Long, A.-H. Aghvami, and V. C. Leung, "Network slicing based 5g and future mobile networks: mobility, resource management, and challenges," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 138–145, 2017.

[10] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[11] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, 2017.

[12] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, vol. 1, p. 132, 2009.

[13] D. Drutskoy, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2012.

[14] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Gonçalves, R. Figueiredo, D. Morris, M. Pickavet, and P. Demeester, "Implementing quality of service for the software defined networking enabled future internet," in *2014 Third European Workshop on Software Defined Networks*.

[15] H. Krishna, N. L. M. van Adrichem, and F. A. Kuipers, "Providing bandwidth guarantees with openflow," in *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, Nov 2016, pp. 1–6.

[16] F. Li, J. Cao, X. Wang, Y. Sun, and Y. Sahni, "Enabling software defined networking with qos guarantee for cloud applications," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, June 2017, pp. 130–137.

[17] S. Tomovic, N. Prasad, and I. Radusinovic, "Sdn control framework for qos provisioning," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*, Nov 2014, pp. 111–114.

[18] R. C. R. Wallner, "An sdn approach: Quality of service using big switchs floodlight open-source controller," *Proceedings of the Asia-Pacific Advanced Network*, 2013.

[19] S. Dwarakanathan, L. Bass, and L. Zhu, "Cloud application ha using sdn to ensure qos," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 1003–1007.

[20] T. Mizrahi and Y. Moses, "Time4: Time for sdn," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 433–446, 2016.

[21] B. Turkovic, F. Kuipers, N. van Adrichem, and K. Langendoen, "Fast network congestion detection and avoidance using p4," in *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*. ACM, 2018, pp. 45–51.

[22] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, D. Aziz, and H. Bakker, "Network slicing to enable scalability and flexibility in 5g mobile networks," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, May 2017.

[23] M. Dohler, T. Mahmoodi, M. A. Lema, M. Condoluci, F. Sardis, K. Antonakoglou, and H. Aghvami, "Internet of skills, where robotics meets ai, 5g and the tactile internet," in *2017 European Conference on Networks and Communications (EuCNC)*, June 2017, pp. 1–5.

[24] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5g-enabled tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 460–473, March 2016.

[25] Z. Hou, C. She, Y. Li, T. Q. S. Quek, and B. Vucetic, "Burstiness-aware bandwidth reservation for ultra-reliable and low-latency communications in tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 11, pp. 2401–2410, Nov 2018.

[26] M. Condoluci, T. Mahmoodi, E. Steinbach, and M. Dohler, "Soft resource reservation for low-delayed teleoperation over mobile networks," *IEEE Access*, vol. 5, pp. 10 445–10 455, 2017.

[27] F. Zhou, F. D. l. Torre, and J. K. Hodgins, "Hierarchical aligned cluster analysis for temporal clustering of human motion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 3, pp. 582–596, March 2013.

[28] L. Li and B. A. Prakash, "Time series clustering: Complex is simpler!" in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 185–192.

[29] D. Baumann, F. Mager, R. Jacob, L. Thiele, M. Zimmerling, and S. Trimpe, "Fast feedback control over multi-hop wireless networks with mode changes and stability guarantees," 2019.

[30] National Science Foundation, "NSF Follow-on Workshop on Ultra-Low Latency Wireless Networks Executive Summary," no. 1448360, p. 28, 2016. [Online]. Available: http://inlab.lab.asu.edu/nsf/files/WorkshopReport-2.pdf

[31] M. Maier, M. Chowdhury, B. P. Rimal, and D. P. Van, "The Tactile Internet : Vision , Recent Progress , and Open Challenges," no. May, pp. 2–9, 2016.

[32] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, 2007, p. 1027–1035.

[33] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.

[34] mininet. (2019) Mininet. [Online]. Available: http://mininet.org/

[35] "P4 behavioral model," https://github.com/p4lang/behavioral-model, accessed: 19-03-2018.

APPENDIX

A. Equation for Slice Bandwidth

For many use cases of TCPS, the data-rate of the video feedback decides the end-to-end bandwidth requirement [30], [31] and in these cases we can optimize the TCPS bandwidth by modulating the video frame-rate. A high frame-rate is necessary when the operator dynamics are higher, to avoid cybersickness. In general, for a TCPS, we use (3a) to calculate the video frame-rate, $f(t)$, at time t. α is a constant and corresponds to the number of frames required to avoid cybersickness in a given TCPS application. Specific to the dynamic network slicing scheme, we define $f_{max}(S)$, which corresponds to the maximum of the $f(t)$ values supported by network slice S. To compute $f_{max}(S)$, equations (1) and (3a) are used (see (3b)). For computing the peak bandwidth requirement of slice S, $B(S)$, (3c) is used. Here c is a constant, and it corresponds to the bandwidth required for streaming video at a frame-rate of $1fps$. c thus depends on the type (3D or 2D) and quality (HD or SD) of the video. $U(S)$ denotes the utilization factor of slice S. It is used for computing the average bandwidth requirement, $\overline{B(S)}$ (See (3d)).

$$f(t) = \frac{\alpha \times \frac{dr}{dt}}{1m/s}\ fps \tag{3a}$$

$$f_{max}(S) = \alpha \times \frac{1ms}{(RTT(S)\ in\ ms)}\ fps \tag{3b}$$

$$B(s) = c \times f_{max}(S)\ bps \tag{3c}$$

$$\overline{B(S)} = B(S) \times U(S)\ bps \tag{3d}$$

B. Effect of Varying β

Let us consider the hand speed time series shown in Figure 12. The time series shows that hand speeds periodically oscillate between $0.1m/s$ and certain higher values, namely $0.2m/s$, $0.275m/s$, $0.325m/s$, $0.4m/s$, and $0.475m/s$. We cluster hand speeds using Algorithm 1 with different values of β and number of clusters, $K = 2$; the corresponding cluster boundaries are shown in Table IV. When $\beta = 0$, Algorithm 1 accounts only for the cost: it puts $0.1m/s$ and $0.2m/s$ in one cluster and the remaining speeds in the other cluster, causing 4 slice switchings per cycle. As we increase β, Algorithm 1 tends to give more weight to visual performance. When $\beta = 2$, it puts $0.1m/s$, $0.2m/s$ and $0.275m/s$ in one cluster and the remaining ones in the other cluster incurring higher cost but reducing the number of slice switchings per cycle to 3. Similarly, as β reaches 5 and 16, the number of slice switchings per cycle further reduces to 2 and 1, respectively, at the expense of increasingly higher costs. As expected, E increases with β (see Table IV).

C. Comparing Algorithm 1 with Brute-Force Optimization

We use the Suturing-B001 dataset from the da Vinci Surgical System database to compare Algorithm 1 with brute-force optimization. Here $v_{max} = 0.18m/s$. As in Section V-A, we set $E_{spec} = 95\%$, $Q = 200$, and $\delta = v_{max}/Q$. We fix the number of clusters K to two. In the brute-force approach, we exhaustively search for the optimum intermediate cluster boundary among $Q = 200$ potential values. In particular, we compute E and I corresponding to each potential intermediate boundary value and determine the cluster boundary that corresponds to the least I, while ensuring $E > E_{spec}$. We depict E and I corresponding to all the potential boundary values in Figure 13. In Table V, we compare the so obtained optimal E and I with those obtained from Algorithm 1. We see that Algorithm 1 gives E and I very close to their optimal values (see Table V). We have made similar observations using other datasets from the da Vinci Surgical System database as well.

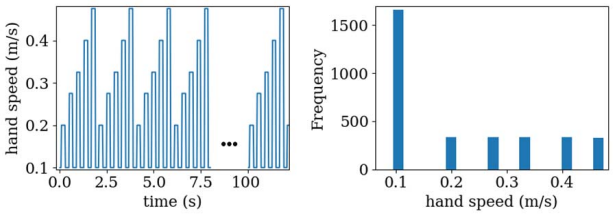


Fig. 12: Hand speed time series used in Appendix B; the right-side figure shows the histogram.

TABLE IV: Effect of varying β on E an I.

β	E	I	Cluster Boundaries
0	92.83	11391	(0m/s, 0.2m/s, 0.475m/s)
2	94.65	12012	(0m/s, 0.275m/s, 0.475m/s)
5	96.47	12908	(0m/s, 0.325m/s, 0.475m/s)
16	98.23	22562	(0m/s, 0.4m/s, 0.475m/s)

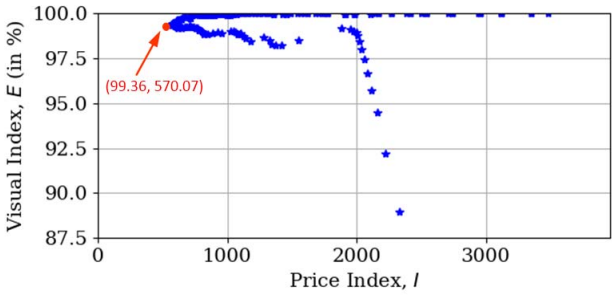


Fig. 13: E vs. I plot in the brute-force approach; the red dot depicts the optimal E and I.

TABLE V: Comparison of Algorithm 1 with the brute-force approach.

	Cluster Boundaries	E	I
Algorithm 1	(0m/s, 0.0579m/s, 0.186m/s)	99.36%	570.92
brute-force	(0m/s, 0.0569m/s, 0.186m/s)	99.36%	570.07

D. Motivation for Algorithm 1

Popular clustering algorithms, such as k-means and mean-shift, generally do not give satisfactory slice specifications for the following reasons.

- They do not account for the time-order of data points which determine the switching requirements. In other words, they yield the same clusters for different time series if they have the same histogram. Our proposed clustering algorithm accounts for the dynamics of the data points.
- They are based on the Euclidean metric, which is not commensurate with the performance and cost metrics of interest. For instance, in k-means the "cost" associated with a cluster-head depends on the Euclidean distances to all the points inside the cluster, whereas in the slicing problem under consideration it depends on the absolute value of the cluster-head and on the fraction of points inside the cluster.

The slice specifications resulting from these algorithms could be far from optimal if the histogram of the data-points has multiple peaks as illustrated via the following examples.

1) Example 1: Let us consider the hand speed time series shown in Figure 14. The hand speed stays at $0.5m/s$ for the first half, then drops to $0.28m/s$ for a tiny fraction of time, and then further drops to $0.1m/s$ and stays there for the remaining time. Setting $K = 2$, k-means and Algorithm 1 yield cluster boundaries as in Table VI. Notice that the clustering given by k-means warrants bandwidth and RTT corresponding to $0.28m/s$ even when the hand speed is $0.1m/s$. Neither of the clusterings causes noticeable visual performance deterioration, but the clustering given by Algorithm 1 incurs 20% less cost compared to the one given by k-means (see Table VI).

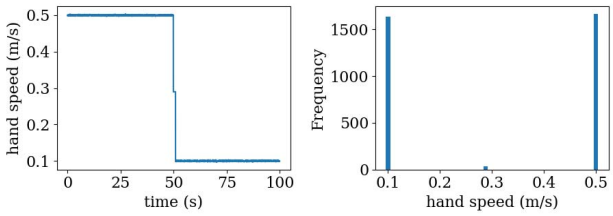


Fig. 14: Hand speed time series used in Example 1. The right-side figure shows the histogram.

TABLE VI: Comparison of k-means with Algorithm 1.

	k-means	Algorithm 1
Cluster Boundaries	(0m/s, 0.28m/s, 0.5m/s)	(0m/s, 0.1m/s, 0.5m/s)
E	99.93%	99.84%
I	16850	13415 ($\downarrow$ 20%)

2) Example 2: Now we consider the hand speed time series shown in Figure 15. The hand speed is $0.5m/s$ for a small fraction of time in the beginning, after which it oscillates between $0.1m/s$ and $0.3m/s$. Setting $K = 2$, k-means and Algorithm 1 yield cluster boundaries as in Table VII. The clustering given by k-means warrants bandwidth and RTT corresponding to $0.5m/s$ when the hand speed is $0.3m/s$ and also leads to frequent slice switchings. On the other hand, the clustering by Algorithm 1 warrants bandwidth and RTT corresponding to $0.3m/s$ when the hand speed is $0.1m/s$, but ensures that there are no frequent slice switchings. This not only incurs 30% less cost, but also 6% higher visual performance index (see Table VII).

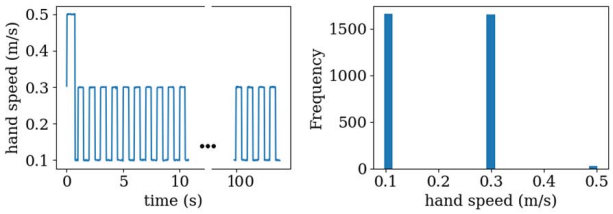


Fig. 15: Hand speed time series used in Example 2. The right-side figure shows the histogram.

TABLE VII: Comparison of k-means with Algorithm 1.

	k-means	Algorithm 1
Cluster Boundaries	(0m/s, 0.1m/s, 0.5)	(0m/s, 0.3m/s, 0.5)
E	94.02%	99.72% ($\uparrow$ 6%)
I	13179	9306 ($\downarrow$ 30%)

E. Network Switching Overhead

1) Delay Overhead: The contribution to the "switching overhead" by the network is represented by the additional transmission delay from all the switches due to the increase in packet size of the first packet processed by the new slice. In the case of a slice setup message, this overhead is equal to (4). Here S_{hdr} is the size of the Slice Configuration Protocol header (except the *Ports Array* field), n is the total number of switches in the path, R_x is the speed of the output link of switch x and S_{port} is the size in bits used to represent one port (usually $8bits$).

$$t_{network} = \sum_{1 \le x \le n} \frac{S_{hdr} + (n - x) \cdot S_{port}}{R_x} \tag{4}$$

2) Memory Overhead: To support such a system, an additional table containing all the possible headers that can be added needs to be maintained for every TCPS flow at the edge switch. This overhead for one TCPS flow can be calculated using (5). Here n_{slice} is the number of slices, S_{flowID} is the size of the chosen flow identifier, usually the 5-tuple (source IP, destination IP, source port, destination port, transport protocol). However, the memory consumption is reduced on all the other switches in the path (compared to a solution where rules are reconfigured in all the switches), as the number of rules needed in the core switches to process a TCPS flow is reduced to 1 from n_{slice}.

$$M_{network} = \sum_{1 \le x \le n_{slice}} (S_{port}\,(n+1) + S_{flowID} + 1) \tag{5}$$