TCPSbed: A Modular Testbed for Tactile Internet-Based Cyber-Physical Systems

Kurian Polachan, Joydeep Pal, Chandramani Singh, T. V. Prabhakar, *Member, IEEE*, and Fernando A. Kuipers, *Senior Member, IEEE*

***Abstract*— Tactile Internet based Cyber-Physical Systems (TCPS) are highly sensitive to component and communication latencies and packet drops. Building a high performing TCPS, thus, necessitates experimenting with different hardware, algorithms, access technologies, and communication protocols. To facilitate such experiments, we have developed TCPSbed, a modular testbed for TCPS. TCPSbed facilitates the integration of different components, both real and simulated, to realize different TCPS applications and evaluate their latency and control performances. TCPSbed's latency analyzer tool employs a novel method to isolate latencies of individual TCPS components such as the latencies contributed by actuation, sensing, algorithms, and by the network, all in an online fashion. TCPSbed's method of analyzing stability is also novel. It involves the use of the step response analysis method, a classic control-theoretic method used for analyzing the stability of generic control systems. TCPSbed's support for edge intelligence modules enables prediction of command and feedback signals at the network's edge allowing TCPS applications to perform well in adverse network conditions. TCPSbed's source-code, made available through our GitHub page *TactileInternet*, allows developers to extend its features and functionalities further. In this paper, we describe the architecture and implementation details of TCPSbed and demonstrate its features through several proof-of-concept experiments.**

***Index Terms*— Tactile internet, tactile cyber-physical systems, haptic communications, testbed, step response analysis, latency analysis, edge computing.**

I. Introduction

THE Internet was designed primarily for the exchange of data. Over the years, however, it has evolved into a medium for transferring real-time audio and video as well. Recently, researchers have provided a new vision for the Internet, referred to as the *Tactile Internet*, in which the Internet is used to enable real-time human-in-the-loop Cyber-Physical Systems (CPS) involving tactile sensory (haptic) feedback and high frame-rate video feedback to control and steer faraway objects in real-time [2]–[4], see Figure 1. The futuristic network will enable these real-time CPS to run with utmost stability and provide enhanced quality of experience to the human operators involved. We refer to these real-time human-in-the-loop cyber-physical systems as Tactile Cyber-Physical Systems (TCPS) [5], [6].

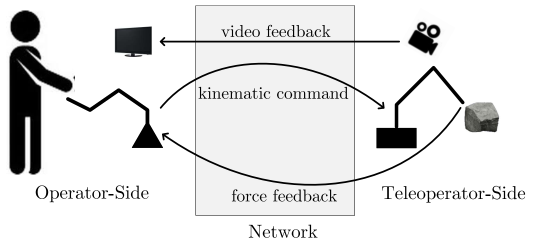

Fig. 1. An example of a human-in-the-loop real-time cyber-physical systems or TCPS — a teleoperation system involving haptic (force) feedback and high frame rate audiovisual feedback used for remote exploration of objects [7].

TCPS applications are highly sensitive to latency and packet drops. Higher of these may cause cybersickness in the use of TCPS applications and control-loop instabilities.

Cybersickness: In tactile-visual control applications such as gaming, at higher latencies, the video feedback in response to the operator's actions will be delayed and noticeable to the operator, resulting in cybersickness [8]. Cybersickness can cause physical and physiological effects in human operators, preventing them from using the system for a long duration [9]. The latency at which the operator starts to experience cybersickness depends on the operator dynamics (i.e., the operator's hand speed) and the operator side's video feedback configuration. For most use cases, latencies of more than a few milliseconds can already result in cybersickness.

Control Loop Instability: There exists a control loop in the TCPS of Figure 1. This control loop results from the kinematic command and force-feedback between the operator-side human-system interface and the teleoperator. Control loop instabilities can cause the remote-side robot to go out of synchronization with the operator's hand movements. In critical applications such as telesurgery, such instabilities can result in injuries to patients. Instability can arise in the control loop when the end-to-end latency, jitter, or packet drops are high [10]. The latency, jitter, and packet drop values at which the control loop destabilizes depend on several factors.

Manuscript received July 8, 2020; revised June 9, 2021 and September 14, 2021; accepted October 24, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Chen. Date of publication November 15, 2021; date of current version April 18, 2022. This work was supported in part by the Visvesvaraya Ph.D. Scheme and in part by the Visvesvaraya Young Faculty Research Fellowship. An earlier version of this paper was presented at the 11th International Conference on Communication Systems and Networks (COMSNETS), Bangalore, India, 2019 [DOI: 10.1109/COMSNETS.2019.8711100]. *(Corresponding author: Kurian Polachan.)*

Kurian Polachan, Joydeep Pal, Chandramani Singh, and T. V. Prabhakar are with the Department of Electronic Systems Engineering, Indian Institute of Science (IISc), Bengaluru 560012, India (e-mail: kurian@iisc.ac.in; joydeeppal@iisc.ac.in; chandra@iisc.ac.in; tvprabs@iisc.ac.in).

Fernando A. Kuipers is with the Software Technology Department, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: f.a.kuipers@tudelft.nl).

Digital Object Identifier 10.1109/TNET.2021.3124767

They depend on the operator dynamics (i.e., the operator's hand speed), human-system interface and robot make, and even the material's stiffness at the remote side. For most use cases, however, latencies of more than a few milliseconds can destabilize the control loop [10].

Current ambitions for 5G communications and IETF Deterministic Networking (DetNet) standards stipulate network latencies for TCPS in the order of a few milliseconds, provided the distance between operator and teleoperator sides be within certain limits [11]. When the distance is considerable, irrespective of the underlying network technology, the network will introduce a communication delay no smaller than the distance divided by the speed of light. In such cases, to avoid control loop instability and cybersickness in TCPS (while the actual latency is still restricted by the speed of light and is beyond TCPS requirements), prediction algorithms may be used at the TCPS edges i.e., at the operator and teleoperator-sides [12]. Using prediction techniques is the only way that latency can be obtained that is seemingly lower than the speed of light. Using prediction techniques also improves the reliability of the network in scenarios where there exists considerable packet loss. In TCPS, the prediction blocks near the operator-side predict the control commands, and those at the teleoperator-side predicts feedback (e.g., haptic, video) to imitate the behaviour of a low-latency and reliable network. The method of using prediction-based approaches in TCPS to improve its network quality is referred to as edge intelligence [8], [13].

Even if we assume the availability of a low-latency and reliable network in the foreseeable future, designing a high-performing TCPS will still require us to experiment with different hardware, prediction algorithms, access technologies and communication protocols and evaluate their end-to-end latencies and stabilities [12]. Existing testbeds for teleoperation systems, 5G and IETF DetNet, however, are either application-specific or designed to evaluate the performance of specific components. They are not readily usable for the requirements of general TCPS applications and cannot help conduct experiments to evaluate the latency and stability of the system. This has motivated us to develop TCPSbed, a modular testbed specifically designed for TCPS with support for a broad range of TCPS components. TCPSbed allows the integration of different components, both real and simulated, such as networks in ns-3 or Mininet and robots in VREP, to realize end-to-end TCPS applications with reasonable effort [14]–[16]. For instance, TCPSbed out-of-the-box supports different TCPS flows, such as kinematic, audio, video, and haptic, which are common in many TCPS applications. TCPSbed also combines novel methods and tools to evaluate both end-to-end and component performance of TCPS applications, in particular, to evaluate latencies of TCPS components and the stability of TCPS control loops. TCPSbed's support for edge intelligence modules enables the integration of prediction algorithms at the network edges, which is key in lowering latency.

In this paper, we describe the architecture and implementation details of TCPSbed and demonstrate its features through several proof-of-concept experiments.

A. Contributions

Our main contributions are the following:

- We present the modular architecture of TCPSbed, an open-source testbed designed specifically for TCPS applications.
- We discuss edge intelligence in the context of TCPS and further describe the structure and usage of TCPSbed's edge intelligence components.
- We describe methods and tools to perform latency characterization and stability analysis of TCPS and its components.
- We present several proof-of-concept experiments to demonstrate the flexibility and the novel features of TCPSbed.

B. Related Work

1) Testbed: Past works on testbeds for teleoperation systems are intended for specific applications and often designed to analyze the performance of a specific testbed component [17]–[22]. For instance, in [17], the authors describe a testbed specifically designed for hotline robot applications. The same is true for testbeds for the Tactile Internet [23]–[25]. Their designs focus on evaluating the performance of specific testbed components, such as 5G and Wi-Fi access technologies. Application and component specific testbeds of the above kind are not readily portable to the requirements of general TCPS applications and cannot help conduct experiments with different components, such as hardware, access technologies, and communication protocols. Recently, after our work [1], authors in [26] has proposed TIXT, a testbed specifically designed for TCPS. TIXT however has the following shortcomings compared to TCPSbed.

- Its source code is not open and the API framework is not defined/documented.
- It lacks edge intelligence components.
- It does not support Mininet, the go-to emulator to experiment with software-defined networks, out-of-the-box.
- It does not provide methods or tools to evaluate the latency or stability of TCPS. Latency and stability analysis of TCPS is essential to study the effects of cybersickness and control-loop instabilities.

2) Latency Characterization: Component-level latency characterization is crucial in TCPS. It helps in optimizing the components to meet the stringent end-to-end latency and reliability requirements of a TCPS application. However, none of the testbeds mentioned above provides methods or tools to perform component-level latency characterization of TCPS, which is a key contribution of our paper.

Several open/commercial tools such as ping, traceroute, solarwinds, and visualroute exist for evaluating the latency of network components of a TCPS [27], [28]. However, these tools are not suitable for evaluating the latencies of TCPS non-networking components, e.g., the latency associated with sensing, actuation, and the algorithms. To the best of our knowledge, there exists no tool to simultaneously evaluate the networking and non-networking components in a TCPS, which motivated us to develop a custom latency characterization method and tool, as described in this paper.

3) Performance Analysis: Several subjective and objective methods to evaluate TCPS-like systems are found in the literature [29]–[34]. Subjective methods employ human operators to evaluate TCPS. They often measure the human operator's quality of experience to evaluate instabilities in TCPS indirectly — a lower quality of experience indicates higher levels of instabilities. Being subjective, often they are time-consuming and cannot detect control-loop instabilities that lie outside the sensory perception range of human operators. Objective methods use Quality of Service (QoS) metrics such as latency, jitter, and packet drops to evaluate TCPS. However, it is not easy to arrive at the right combination of these metrics that result in stable control loops. These shortcomings motivated us to propose an alternative method to characterize the performance of TCPS that uses the classical step response analysis method. In our work, we design and develop the necessary framework and tool to conduct step response analysis on TCPS.

C. Outline

This paper is structured as follows: Section II describes the architecture of TCPSbed, its constituting components, their APIs and interfaces. Section III presents different methods to evaluate the performance of TCPS using TCPSbed. Specifically, describing how to conduct latency and step-response characterizations. Section IV describes how we implemented TCPSbed in our lab. We evaluate the performance of TCPSbed and demonstrate its applications through several proof-of-concept experiments in Section V and Section VI. We conclude in Section VII.

II. Testbed Architecture

A TCPS consists primarily of three subsystems — the operator-side that includes the human operator, the teleoperator-side that includes the remote side robot, and the communication network, referred to as tactile internet, connecting the two sides. TCPS applications being latency-sensitive, the components in these three subsystems have to meet stringent delay requirements. Often this necessitates exploring different hardware, algorithms and protocols for these components. In such experiments, a researcher would often focus on only one of the subsystems, and in many cases, on a particular component of the subsystem. To facilitate such experiments with minimal overhead, we design a modular architecture for TCPSbed, see Figure 2.

TCPSbed separates various TCPS subsystems into multiple components of distinct functions and glues them together through a set of APIs with standardized interfaces to realize various TCPS data-flows such as audio, video, kinematics and haptics. Separation of subsystems into components would allow researchers to focus on certain components without bothering about the implementation and interface details of others. This kind of architecture makes TCPSbed generic, modular and extensible.

In this section, we first describe the various components of TCPSbed and then explain the structure and placement of their APIs.

A. Components

TCPSbed consists of the following components:

- *[tactile-operator, tactile-teleoperator]*: The *tactile-operator* represents the human operator, and the *tactile-teleoperator* represents the remote-side robot being controlled. Due to the cost and resource constraints, researchers sometimes may want to use a simulated *tactile-teleoperator* . The Virtual-Robot Experimentation Platform, *V-REP*, serves this need. It simulates different types of robots and their physical surroundings.
- *[ms-embsys, ss-embsys, ms-embsys-app, ss-embsys-app]*: TCPSbed uses embedded computing boards both at the operator and the tele-operator sides. *ms-embsys* is the board at the operator-side and houses sensors and algorithms to capture the operator's kinematic movements. The board also includes drivers to display remote-side audiovisual information and to convey haptic feedback to the operator. The board on the teleoperator-side, *ss-embsys*, houses actuators to drive the robot and sensors to capture audio, video, and haptic information. In some cases, it may be useful to simulate *ms-embsys* and *ss-embsys* boards in code, e.g., to re-play a human operator's hand movements from a dataset or to evaluate the performance of a robot using its mathematical model. We refer to the simulated versions of the embedded computing boards as *embsys-app*'s. The simulated version of *ms-embsys* is referred to as *ms-embsys-app* and the simulated version of *ss-embsys* as *ss-embsys-app*.
- *[ms-com, ss-com]*: Different applications may use different communication technologies for connecting the operator and teleoperator sides to the network. For instance, some applications may use WiFi links, while some others may use point-to-point Ethernet links. In a testbed for TCPS, it is desirable to have separate communication components to facilitate easy porting and testing of different communication technologies. *ms-com* and *ss-com* serve as the communication components in TCPSbed. They connect *ms-embsys* and *ss-embsys* to the network.
- *[ms-ei, ss-ei]*: TCPSbed comes with ready-to-deploy edge intelligence components *ms-ei* and *ss-ei* at the operator and teleoperator sides, respectively. We use Python in a Linux environment to create these edge intelligence components. We can host these components in any IP-based computing boards that support Linux and interface with the rest of the testbed components. We also expose a set of APIs in these components which the developer can modify to code the desired prediction algorithm. The exact placement of *ms-ei* and *ss-ei* depends on the TCPS application and complexity of the prediction algorithm in use. For instance, if the last mile is a wireless link, it is more appropriate to place the edge intelligence component one-hop away from *ms-com* and *ss-com* and near the radio base-station. Such a placement enables accurate

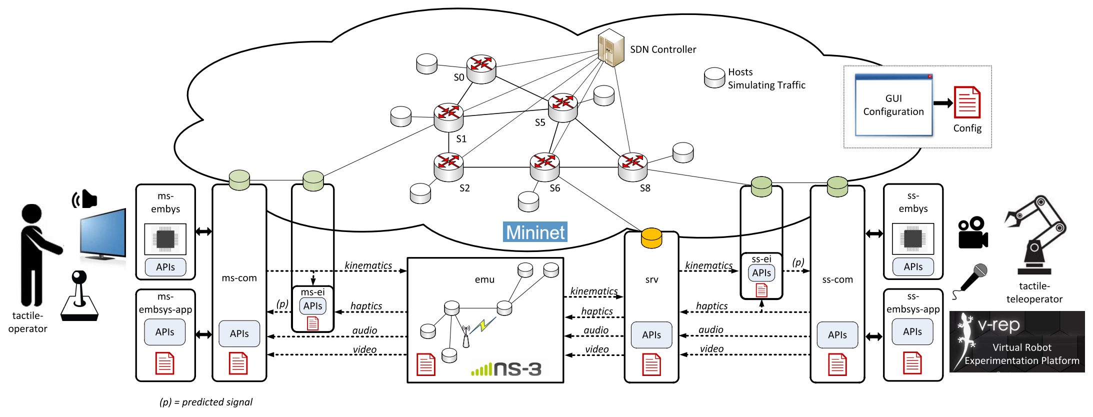

Fig. 2. The architecture of TCPSbed.

predictions of incoming signals that do not depend on wireless channel quality. For a wired network, placing the edge intelligence component in a more resourceful host can enable to run more intelligent and sophisticated prediction mechanisms.

- *[srv]*: Several TCPS applications demand the need to run computationally intensive algorithms, such as the kinematics and inverse-kinematics algorithms, both at the operator and the teleoperator sides. A testbed for TCPS can include embedded computing boards, *ms-embsys*, and *ss-embsys* with powerful processors to execute such algorithms. However, such a testbed design is challenging as it is not easy apriori to assess the processing requirements of the embedded computing boards to support different TCPS applications. To work around this difficulty, we introduce the component *srv* in TCPSbed. The TCPS flows from the operator to the teleoperator side and back pass through *srv*. If the computational requirement of a TCPS application is higher than what embedded computation boards can handle, we offload the algorithms to *srv*. *srv* could be a real server or a high-performance computer. During the TCPS development phase, *srv* provides insights into the computing power required by the embedded computers, which helps in choosing optimal embedded computing boards for the final TCPS design.
- *[network]*: Many times during development, it is useful to evaluate the performance of the TCPS using simulated computer networks in addition to real networks. In TCPSbed, network simulators ns3 and Mininet serve this need. When using ns3, the testbed components are run on real hosts and are interconnected using the component *emu*, which runs a simulated ns-3 network. When using Mininet, both the hosts running the testbed components and network are simulated.

B. APIs

Typical TCPS applications require four types of data-flows, namely kinematic, haptic, audio, and video flows. TCPSbed uses custom APIs in the testbed components to realize these

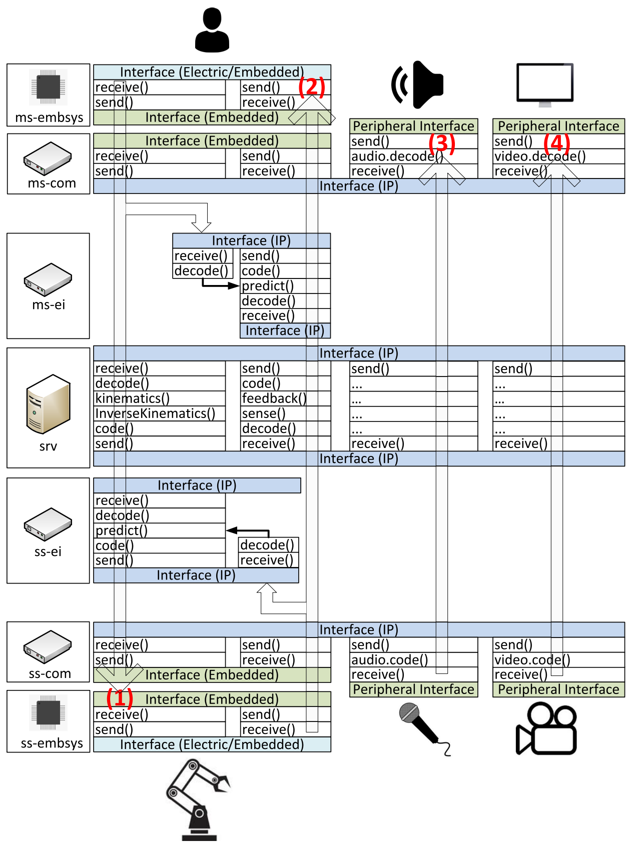

Fig. 3. APIs in the forward and backward flow paths in a TCPS. (1) represents the kinematic flow. (2), (3), and (4) represent the haptic, audio, and, video flows respectively.

flows. Figure 3 shows these APIs and how they connect various testbed components.

- *[receive(), send()]*: *receive()* reads data from the connected physical sensors or adjacent TCPS components. At *ms-embsys*, *receive()* reads data from the kinematic sensors attached to the operator. At *ss-embsys*, *receive()* reads data from position sensors and haptic sensors mounted on the robotic arm. At *ms-com* and *ss-com*, *receive()* reads data from the embedded boards *ms-embsys* and *ss-embsys*, respectively, running embedded protocols (e.g., I2C, SPI or USB). At *srv*, *receive()* reads data from both *ms-com* and *ss-com* in either directions, running network transport protocols (e.g., UDP). *send()* transmits data to the connected physical actuators or adjacent TCPS components. Like *receive()*, its input and output depends on the TCPS component on which it resides, on the embedded and network protocols in use, and the types of sensors and actuators.

- *[code(), decode()]*: Testbed components use these APIs for coding and decoding data. The code/decode format depends on the TCPS component and its interface protocols. Specific to the audio and the video flows are *audio.code()*, *audio.decode()*, *video.code()*, and *video.decode()* APIs.
- *[kinematics(), inverse_kinematics()]*: *kinematic()* uses data from the kinematic sensors mounted on the operator (or on the operator-side human-computer interface) to compute the operator's pose and movements. *inverse-kinematic()* uses these computed values to generate commands to drive the remote-side robot actuators to recreate the operator's pose.
- *[sense(), feedback()]*: *sense()* does calibration, scaling, and filtering of the haptic data it receives from the haptic sensors mounted on the remote-side robot. *feedback()* uses the processed haptic data to drive haptic actuators at the operator-side.
- *[predict()]*: At *ms-ei*, the API predicts haptic data using the current kinematic sample it receives from the operator-side and the past haptic samples it receives from the teleoperator-side. At *ss-ei*, the API predicts kinematic data using the current haptic sample it receives from the teleoperator-side and from the past kinematic samples it receives from the operator-side. The predict() can also be configured to modulate the data transmit interval (i.e., how fast predictions are done) and the data receiving interval (i.e., the sampling interval).

C. Testbed Configuration

To use TCPSbed, a user has to configure various networking and non-networking components and their interfaces. These configurations include (i) assigning hosts (e.g., PCs) and IP addresses to *ms-com*, *emu*, *srv*, *ss-com*, *ms-ei* and *ss-ei* (ii) defining the interface types, addresses and port numbers to realize the TCPS flows, (iii) defining the embedded device types - real or simulated and their settings, (iv) defining the network type - real or simulated, and (v) configuring the network simulator settings such as latency, bandwidth, and packet drops. In TCPSbed, all of the above configuration details are entered in a spreadsheet and then converted to a *config* file and stored in the TCPSbed's run directory, a copy of which resides in every testbed host.

TCPSbed supports several placement possibilities for its components. For instance, it is possible to run several testbed components on a single host or just one component per host. It is also possible to run testbed components on simulated Mininet hosts, or real hosts separated over an intercontinental link. We demonstrate these possibilities through the experiments in Section V.

III. Latency and Step Response Characterization

In this section, we describe how to perform latency and step response characterization of a TCPS application using TCPSbed.

A. Latency Characterization

Since a TCPS consists of multiple distributed components, it is essential to get an insight into the performance of each component. For example, it is desirable to measure a remote robotic arm's reaction time to a user command to budget for end-to-end latencies over a production network. Such measurements require a tool similar to ping, with enhanced support for TCPS. Such a tool should also measure processing delay and actuation delays in addition to the network latencies.

We can use TCPSbed as an in-situ latency test ecosystem with the help of a *Test-PC* and echo points. Echo points are locations on the forward and backward data paths where we examine the latency values. Each TCPS component can have multiple echo points. A *Test-PC*, as in Figure 4, issues echo commands targeting a particular echo point and then measures the time it takes for the echo to arrive. The process is repeated multiple times and for multiple echo points to get the latency distribution. The echo commands are UDP-based wrappers around the data commands: they contain data that the *ms-embsys* and *ss-embsys* components expect. Meaning, the system is fully functional while the test is in progress and, hence, the resultant latency numbers are representative of the TCPS under test.

In Figure 4, *Test-PC* inserts the echo commands in the kinematic data path through *ms-com*, bypassing *ms-embsys* which does not have an IP address. Thus the resultant echo trajectory of the kinematic data path does not cover the latency between *ms-embsys* and *ms-com*. As a solution, we propose using the echo trajectory of the haptic data path to measure and isolate the forward kinematic latencies involved between *ms-embsys* and *ms-com*.

Note: It may be possible for a loaded OS kernel in hosts to affect latency measurements. In order to circumvent this problem, we measure a component latency by finding the difference of the latency associated with echo points placed at the entry and exit points of the component. We expect the kernel-dependent offset in the latency measurements to be present in the latencies of both the entry and exit echo points. Taking a difference will thus cancel this offset. We also send echo commands in an interleaved fashion to the targeted echo points to avoid temporal variations in the kernel's load from affecting the latency measurement. In this method, if we want to measure the latency of echo points A and B placed at the entry and exit point of a component by sending m echo points each, we send the first echo command targeting to A, next command targeting to B, next targeting to A, next targeting to B and so on until m echo points are sent to both A and B. This contrasts with sending the first m echo commands to A and then sending the next m echo commands to B.

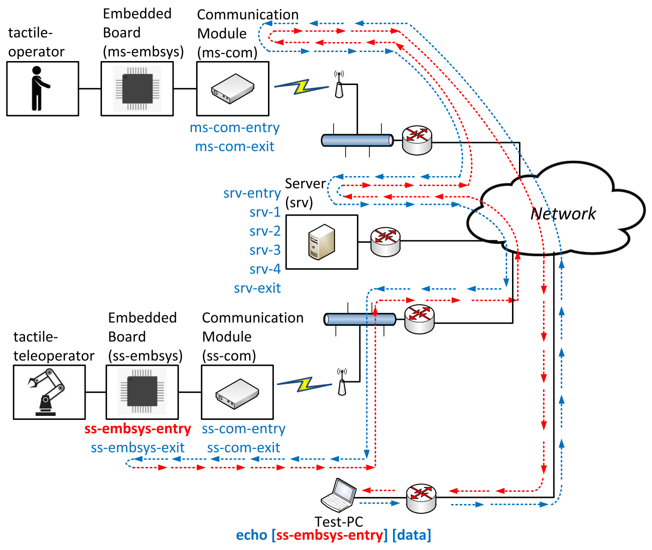

Fig. 4. In-situ latency test. The path taken by the echo command targeting echo point *ss-embsys-entry* is marked. Only the echo points in the forward kinematic data path are shown.

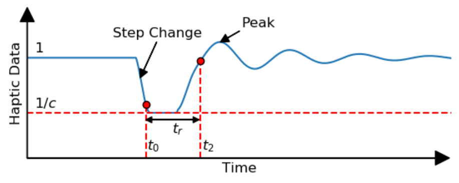

Fig. 5. Sample step response profile with normalized haptic data. Step change is simulated at t_0.

B. Step Response Characterization

We can model TCPS applications using control system elements. In this model, the *tactile-operator* acts as the controller and the *tactile-teleoperator* as the device being controlled. The haptic data comprises the feedback to this controller, and the kinematic data forms the controller output. With this control model in place, it is possible to evaluate the control performance of the TCPS application using classical control-theoretic methods. In our work, we use the step response method to profile the control performance of TCPS applications [35], [36].

To perform the analysis, we have created two *embsys-app*'s: one is connected to the *ms-com* and the other to *ss-com*. The *ss-embsys-app* at the *ss-com* replaces the tactile teleoperator. It takes kinematic data from the operator side, k, as input and simulates haptic data, h, as a linear function of the kinematic data; $h = s(t) \times k$, where the scaling factor $s(t)$ is set to 1 for $0 \leq t < t_0$ and to $1/c$ for $t \geq t_0$. This is to simulate a step-change in the pressure. *ms-embsys-app* at the *ms-com* end senses this step-change through the haptic data path. It then sends commands over the kinematic data path to nullify this change. Proportional Integral (PI) controller equations are used for this purpose [37]. Using a logger tool in *ss-embsys-app*, we record the step response profile of the haptic data; see Figure 5. We define t_r as the time taken by the step response to rise from $(1/c)\ units$ at t_0 to $(1/c + 0.9(1 - 1/c))\ units$. Overshoot is defined as the peak percentage fluctuation in the step response relative to $(1 - 1/c)\ units$.

Both rise-time (t_r) and overshoot depend on the TCPS component and network characteristics such as latency, jitter, and packet drops, e.g., both increase with network latency. Further, any TCPS application can withstand rise-times and overshoots up to specific values depending on operator hand speed and accuracy requirements. For instance, a certain rise-time may suit an application with slow operator hand movement, but may not for another application with faster operator hand movement. Similarly, an overshoot value may be tolerable for an application requiring less accuracy, but may not be for another application requiring more accuracy. In a nut shell, step response parameters such as rise times and overshoots can be used to judge a TCPS implementation's stability. Further, since they are affected by both networking and non-networking component characteristics, they can also aid in the construction of a one-stop metric to evaluate and compare TCPS implementations (see SectionV-F).

Note: Step response curve characteristics of a TCPS vary widely with PI controller parameters and step-change amplitude. Further, certain settings and combinations of these parameters may mask possible issues in a TCPS. For instance, setting a low PI controller constant or controller responsiveness will result in a step response curve that does not change its characteristics with variations in packet drops or latencies in the TCPS network. We address this concern by determining optimal values for PI controller parameters and step-change amplitude such that the experiments yield a step response curve that is maximally sensitive to networking and non-networking parameter changes of the TCPS under test.

IV. Testbed Implementation

This section describes how we implemented TCPSbed and its components in our lab.

A. Hardware Resources

We used desktop PCs running Ubuntu OS to host IP-based testbed components *embsys-app*'s, *ms-com*, *ms-ei*, *emu*, *srv*, *ss-ei*, and *ss-com*. To connect these IP-based components, the UDP transport protocol is used. PSoC based development boards are used to realize embsys components, *ms-embsys* and *ss-embsys* [38]. The embsys boards communicate to their communication counterparts, i.e., *ms-com* and *ss-com*, using a full-duplex, $1\,\text{Mbps}$, UART link.

B. API Implementation

In order to implement the APIs in Figure 3, we use Python scripts for the IP-based testbed components and a C script for the *embsys* components. The use of Python enables the APIs to be easily extended and ported to different systems. A downside of using Python is that it results in higher processing times for the APIs that code complex algorithms. However, we propose such APIs be coded in C, compiled and called within Python. Such a procedure will preserve all the advantages of Python while ensuring that the processing times of these APIs are within the requirements.

APIs corresponding to different flows in Figure 3 are run in parallel. In IP-based testbed components, this is done by placing the APIs corresponding to each flow in a separate loop and running these functions as different threads using Python's multiprocessing package.

We use Python's Open-CV library for implementing video flow [39]. The library APIs are used to capture video frames from the camera connected to *ss-com*. The frames are then encoded in JPEG to reduce size. Further, they are serialized using the Python pickle library and transmitted to *ms-com*. At *ms-com*, the received data are deserialized, decoded and displayed on a connected LCD screen. For high-definition video transmission, we split each frame into multiple sub-frames before transmission.

For implementing the audio flow, we use Python's PyAudio library [40]. The library APIs are used to capture audio frames from the microphone connected to *ss-com*. These audio frames are serialized and transmitted to *ms-com*. At *ms-com*, the audio frames are reconstructed and played on a connected speaker.

For the kinematic flow, we have implemented kinematic and inverse-kinematic algorithms to capture the operator's kinematic movements and to drive the robot joints to recreate the operator pose, respectively. We designed kinematic algorithms for a wearable tactile glove and a standard computer mouse [41]. There is a need to support a computer mouse to accommodate users who do not own an electronic glove to control the remote robot. To support a computer mouse to track the operator's movement, we use *ms-embsys-app*. The state machine in this embsys-app maps the movements of a computer mouse in the X-Y plane to a virtual hand movement in X-Y-Z space. For this, the algorithm maps the X-Y movements of the mouse to X-Y movements of the hand wrist. Y movement of the mouse, while the left button is in the pressed state, is mapped to the Z movement of the wrist.

C. Edge Intelligence

We use Python scripts to create the edge intelligence components *ms-ei* and *ss-ei*, and to interface them with the rest of the testbed components. Both *ms-ei* and *ss-ie* take kinematic and haptic data as inputs. As output, *ms-ei* generate haptic data and *ss-ei* generate kinematic data. In the edge intelligence components, we read kinematic, and haptic data at periodic intervals from their network sockets. The read values are then fed as arguments to API, *predict()*. This API processes the inputs it receives and returns the forecasted kinematic data in *ms-ei* and the forecasted haptic data in *ss-ei*. In edge intelligence components, we configure the input network sockets in non-blocking mode to allow edge intelligence components to read and predict outputs at frequencies higher than the kinematic and haptic transmit frequencies of their preceding testbed components. As a result, often, the data read will be null.

In our work, to demonstrate the use of edge intelligence with TCPS, we have coded a zero-order-hold predictor in *predict()* APIs of the edge intelligence components. In *ss-ei*, if the kinematic data read and fed to *predict()* is null, the zero-order hold predictor predicts the current kinematic value as the last received non-null kinematic value from *srv*. The *predict()* sets the current kinematic value to the kinematic

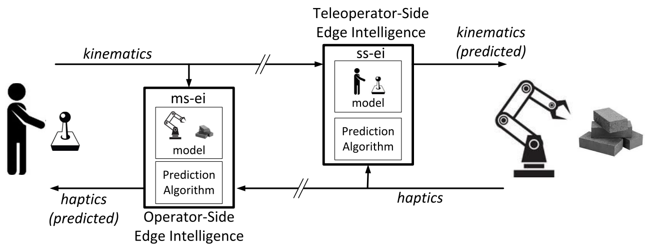

Fig. 6. Placement of edge intelligence components in a TCPS that incorporates a model-based prediction approach.

data it receives if the received data is not null. Although the zero-order hold predictor may appear as a naive prediction algorithm, we can use it to counteract packet losses in a TCPS network. For instance, if there exist packet losses in the last-mile link connecting *ss-ei* and *ss-com* of a TCPS, we can run the zero-order hold predictor at high frequencies to duplicate data packets and counteract packet losses. In Section V-E, we show with the help of an experiment how *ss-ei*, which runs a zero-order hold predictor, can counteract packet losses in a lossy last-mile TCPS link.

1) Modifying Predict(): We can modify *predict()* to code the desired prediction algorithms. For example, we can code a linear prediction algorithm in the *predict()* API of *ss-ei*, as follows. In *ss-ei*, if the kinematic data read and fed to the *predict()* is *null*, then *predict()* outputs the current kinematic value as a linear function of a fixed number of previously received non-null kinematic values from *srv*. If the received data is not *null*, *predict()* sets the current kinematic value to the received kinematic data.

We can also modify *predict()* to code a model-based predictor that uses physical models of devices and environments in a TCPS application to predict outputs [12]. Figure 6 shows the placement and input/output structure of the edge intelligence components in a typical TCPS that uses model-based prediction. The edge intelligence component at the teleoperator-side, *ss-ei*, uses a model of the operator and operator-side human-computer interface to predict kinematic commands from haptic data. The edge intelligence component at the operator-side, *ms-ei*, predicts haptic feedback from kinematic data using a model of the robot and teleoperator-side environment. It is also possible to predict audio and video feedback. However, for many use cases of TCPS, predicting haptic feedback alone may suffice. These are the cases in which the TCPS performance measures, such as the control-loop stability, are more sensitive to haptic feedback latency than audio and video feedback latencies [10].

We remark that, in practice, model-based predictions can often become involved as the models of the environment objects can consist of a large count of parameters that need to be estimated at run-time. Learning so many model parameters at run-time is a complex problem that requires running artificial intelligence or machine learning techniques in *predict()*. In contrast, linear prediction methods are simpler to implement and require less computational resources than model-based prediction methods. Time series predictors, due to their simplicity, suit many applications. However, they result in higher prediction errors in the presence of a more unreliable network.

2) Possible Hardware Options: In practice, the hardware to host edge intelligence components must be decided based on the complexity and computational requirement of the prediction algorithms used. For simple zero-order hold or low-order linear predictors, generic IP-based computing boards, such as Raspberry Pi or generic PCs, may suffice as done in this work. However, hosting edge intelligence components that use full-blown model-based predictors on generic computing hosts will add to the end-to-end latency of the TCPS application. In some cases, the delay may be significant enough to question the purpose of having edge intelligence in the first place. For these cases, it is appropriate to host the edge intelligence components in specialized computing hardware, known as edge computing devices, capable of accelerating machine learning and artificial intelligence prediction algorithms.

Edge computing devices come in two flavours, stand-alone accelerators and single-board computers with integrated accelerators. Stand-alone accelerators are modules that are plugged into a generic computer to enhance their ability to execute machine learning and artificial intelligence algorithms. Movidius NCS is one such example [42]. Single-board computers with integrated accelerators usually run a light Linux operating system with necessary drivers for the on-board accelerators, e.g., Nvidia Jetson [43]. Since many of these edge computing devices support Linux and Python, they can support *ms-ei* and *ss-ei* components of TCPSbed, which are also implemented using Python in a Linux environment.

D. Tools

We have implemented the following two tools to characterize the latency and the step response behaviour of a TCPS.

1) *Latency Characterizer*: The tool reads the raw data generated in the latency characterization experiments to summarizes the result with statistical features and graphs relating to latency, inter-packet delay, and packet drops.
2) *Step Response Analyzer*: The tool reads the raw data generated in the step response experiments and measures the rise time and the overshoot of step curves. Further, it classifies the step response profiles as good or bad, depending on whether the rise time and the overshoot are within given specifications to find the fraction of good step curves, which measures the reliability of the TCPS.

E. Network Simulation

TCPSbed supports both ns-3 and Mininet for simulating network topologies. When using ns3, the testbed components are run on real hosts and are interconnected using the component *emu*, which runs a simulated ns-3 network. On the other hand, when using Mininet, both the hosts and networks are simulated in Mininet.

V. Testbed Evaluation

In this section, first, we demonstrate how to use TCPSbed to deploy an end-end TCPS. Then, through a series of experiments evaluate the testbed performance, features and tools.

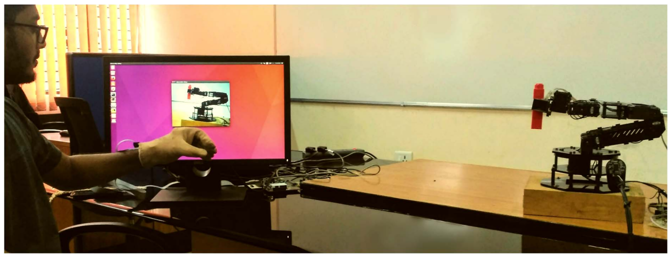
Fig. 7. Demonstration of a TCPS application using TCPSbed — a human operator wears a tactile glove and controls a remote-side robotic arm to pick and place objects.

A. Deploying an End-to-End TCPS Application

The objective of this experiment is to demonstrate how to deploy a TCPS application on TCPSbed. We choose the application described in [41] — a human operator wears a tactile glove and controls a remote-side robotic arm to pick and place objects.

To deploy the above application on TCPSbed, APIs in Figure 3 are updated to match the corresponding algorithms in [41]. A custom-made tactile glove similar to the one described in [41] is used. The tactile glove uses a combination of IMU sensors and flex sensors to track the operator's hand. The tactile glove also uses eccentric rotating mass motors to apply vibrotactile haptic feedback [44].

At the teleoperator-side, pressure sensors are mounted on the robot's (PhantomX [45]), gripper wings to sense the pressure when the robot's arm holds an object. All the four supported data-paths, namely, kinematic, haptic, audio and video, are enabled. For streaming and displaying audio and video data, *ss-embsys-app-microphone* and *ms-embsys-app-speaker* for audio, and *ss-embsys-app-camera* and *ms-embsys-app-display* for video are used. TCPSbed components *ms-com*, *ms-embsys-app*'s and *srv* are hosted on PC-1 and *ss-com* and *ss-embsys-app*'s on PC-2. A point-to-point gigabit Ethernet link connected the two PCs.

Figure 7 demonstrates the above TCPS application using TCPSbed.

B. Isolating TCPS Component and Link Latencies

Unlike traditional latency measurement approaches using ping or traceroute, TCPSbed's latency characterization method is capable of evaluating latencies of both networking and non-networking components in a TCPS application. This experiment's objective is to demonstrate how to conduct a latency evaluation of a TCPS application using TCPSbed and infer the results.

The experimental setup has the same testbed configuration as Section V-A. Additionally, to conduct latency experiments, *Test-PC* is configured to issue echo commands targeting various echo points in the forward kinematic data-path at intervals of $10ms$. By measuring the time it takes for the echo commands to return, the latencies associated with the echo points are determined. Further, the process is repeated multiple ($=N$) times to find the average latencies associated with the

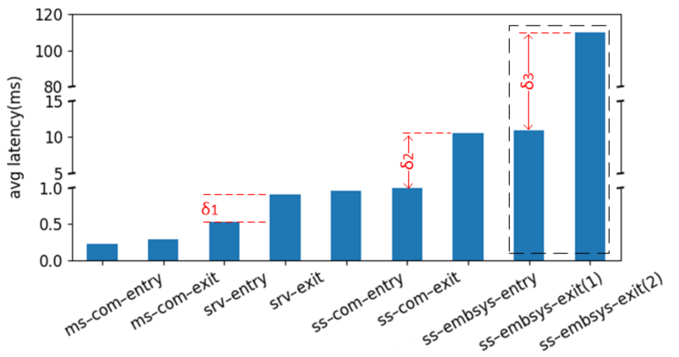

Fig. 8. Average latencies measured for different echo points. For all the latency bars, the margin of error is within 5%.

echo points. N is selected to keep the error of measurement within 5% at 95% confidence interval.

Figure 8 shows the measured average latencies for the various entry/exit echo points of the testbed components.

- The latency values depend on the location of the echo point in the data-flow path. The farther the echo point from the *tactile-operator*, the higher its latency will be. This is evident in Figure 8. In the figure, latency values increase from left to right because echo points to the left are closer to the *tactile-operator* than to the echo points to the right.
- The difference in latency values corresponding to the entry/exit echo points for a given testbed component corresponds to the component latency. E.g., δ_1 corresponds to the latency of *srv*.
- The difference in latency values corresponding to the entry/exit echo points corresponding to two different, but adjacent testbed components, corresponds to the latency of the interface connecting the two components. E.g., δ_2 is the latency of the UART interface connecting *ss-com* and *ss-embsys*.

Estimating Robot Actuation Delay: In Figure 8, the latency that corresponds to the echo point *ss-embsys-exit(1)* does not include the actuation delay of the robot, because, the echo commands targeting *ss-embsys-exit(1)* are sent back as soon as the actuation commands embedded in the echo commands are extracted and sent to the robot's joint motors. However, if the echo commands are held until the robot moves to the commanded position, which is the case with *ss-embsys-exit(2)*, then the actuation delay of the robot can be measured as the difference in latency between the echo points *ss-embsys-exit(1)* and *ss-embsys-exit(2)*. In Figure 8, $\delta_3 = 110\,\text{ms}$ corresponds to the actuation delay of the robot in moving its end-effector position by $1\,\text{cm}$ back and forth. In the experiment, a VREP model of the PhantomX robot is used. The VREP model is interfaced with *ss-com* using *ss-embsys-app-vrep*.

C. Latency Distributions

Objective of the following experiment is to demonstrate the ability of TCPSbed's latency characterization method to capture latencies experienced by kinematic data packets during a telesurgical operation in the presence of simulated and real Internet traffic.

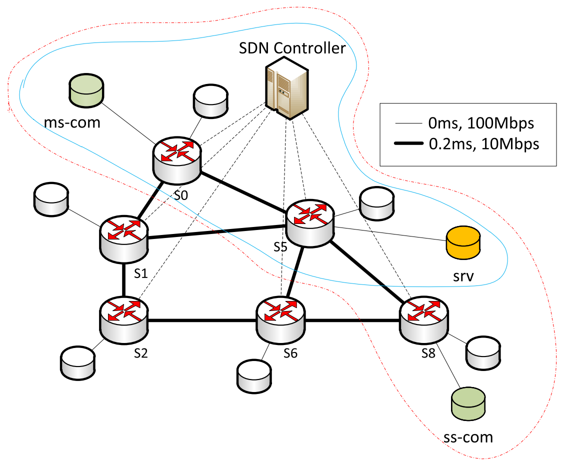

Fig. 9. The network topology simulated in Mininet. The topology corresponds to the top-left portion of the popular USNET 24-node topology. The full network is simulated for the experiments in Section V-D. The network portion circled in blue is used for the experiments in Section V-C. The network portion circled in dotted red is used for the experiments in Section VI-C. We run the Mininet simulations on a server with the following configuration: Processor: Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, Memory: 16 GB, and, with less than 5% and 10% CPU and RAM utilization, respectively.

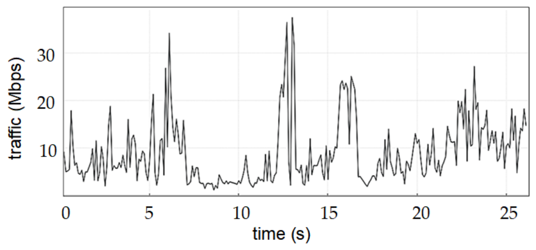

Fig. 10. Traffic simulated by replaying the PCAP file *bigflows.pcap* at replay speed of 1x using Linux tool *tcpreplay*.

1) Latency Evaluation in the Presence of Simulated Internet Traffic: For the experiment, Mininet is used to simulate the network topology circled in blue in Figure 9. Component *ss-com* is run on the same host that runs *srv*. The hosts coloured in white in the circled network portion send data to each other to simulate external traffic. The hosts simulate external traffic by replaying back the Packet Capture (PCAP) file *bigFlows.pcap* [46] using Linux tool *tcpreplay* [47]. *The PCAP is a capture of real network traffic on a busy private network's access point to the Internet* [46]. We use two different replay speed settings for simulating traffic: 1x and 5x. Setting the replay speed to 1x replays the traffic at the same speed at which it is captured in the PCAP file. Setting the replay speed to 5x replays the traffic five times faster than the speed at which it is captured in the PCAP file. Figure 10 shows the simulated traffic at the replay speed of 1x captured using Wireshark [48].

We use TCPSbed's latency characterization method to measure the latency experienced by kinematic packets from the

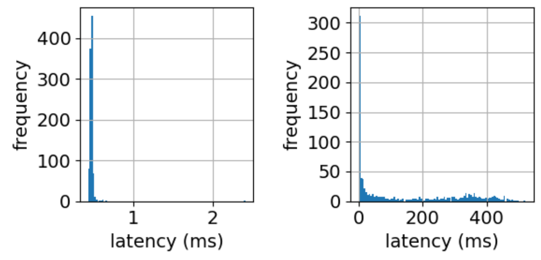

Fig. 11. Distributions of the latencies experienced by kinematic packets traversing through the testbed components without (left-side plot) and with (right-side plot) external traffic.

TABLE I

LATENCIES EXPERIENCED BY KINEMATIC PACKETS TRAVERSING THROUGH THE TESTBED COMPONENTS FOR DIFFERENT INTER-SWITCH LINK BANDWIDTHS AND TCPREPLAY SETTINGS

Tcpreplay Settings		Inter Switch Link Bandwidth (Mbps)	Latencies (ms)
Enabled	Replay Speed		Min,Max,Avg,Std Dev
No	-	10	0.34,0.98,0.41,0.02
Yes	1x	10	0.36,625.39,142.28,156.73
Yes	1x	100	0.35,1.47,0.42,0.02
Yes	5x	100	0.35,5.33,0.42,0.99

entry point of *ms-com* to the exit point of *ss-com*. The kinematic packets used are from [49] that correspond to a surgeon performing a suturing operation using the da Vinci Surgical System. For this, we embed echo commands to each kinematic packet read from the da Vinci Surgical System dataset to target the echo point *ss-com-exit*. These kinematic packets are then inserted into the kinematic data path using *Test-PC*, as shown in Figure 4.

Figure 11 shows the distribution of latencies experienced by the kinematic packets without and with simulated external traffic for inter-switch link bandwidth of $10\,\mathrm{Mbps}$ and PCAP replay speed of 1x. Table I lists the latencies experienced by the kinematic packets for different inter-switch link bandwidths and tcpreplay settings.

2) Latency Evaluation in Presence of Real Internet Traffic Across an Intercontinental Link: For this experiment, we used two PCs. We placed PC-1 and PC-2 in two universities: Indian Institute of Science (Bangalore, India) and Delft University of Technology (Delft, The Netherlands). PC-1 hosted *ms-com*, and *srv* and PC-2 hosted *ss-com*. We then used TCPSbed's latency characterization method to measure the latencies experienced by kinematic packets traversing from the entry point of ms-com to the exit point of *ss-com*. Figure 12 shows the result.

Latencies experienced by kinematic packets have a long tail in the presence of external traffic. The tail is a result of the link bandwidth not being large enough to accommodate the peak traffic rates. Since TCPS applications are sensitive to latencies, long latency tails are a concern. Further, critical TCPS applications demand stringent bounds also on end-to-end latency. Traditional best-effort networks cannot guarantee such bounds on latencies without overprovisioning the link bandwidth.

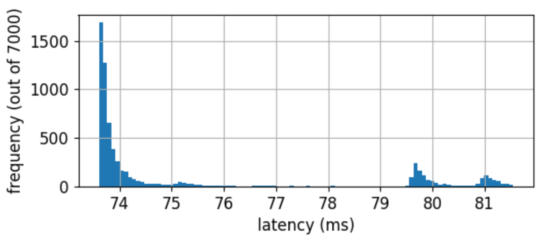

Fig. 12. Distribution of latencies experienced by kinematic packets over an intercontinental link.

TABLE II

SET AND MEASURED CODE BLOCK LATENCIES FOR DIFFERENT CODE BLOCK LATENCY SETTINGS AND TRAFFIC CONDITIONS

Code Block Latency (ms)			
Set Value	Measured (at T_b=0)	Measured Value (at T_b=500 Kbps)	Measured Value (at T_b=750 Kbps)
1	1.08	1.07	1.01
3	3.09	3.15	3.09
5	5.08	5.15	5.08
10	10.09	10.16	10.11
15	15.09	15.11	15.10

D. Accuracy of Latency Measurements

The objective of this experiment is to evaluate the measurement error with TCPSbed's latency characterization method in a sample TCPS setting. For the experiment, the network in Figure 9 is simulated in Mininet. We use a code block in *srv* to simulate different code latencies. We then measure these latencies using the TCPSbed's latency characterization method and compare them with the original setting to determine the measurement error. We experiment with three different external traffic conditions. For simulating the external traffic every host coloured in white in the network transmits data to every other host coloured in white at a constant bit rate of T_b. We run the experiment for three different T_b: $0\,\mathrm{Kbps}$, $500\,\mathrm{Kbps}$ and $750\,\mathrm{Kbps}$. For the latency measurements, we send 5000 echo commands targeting the entry and exit points of the code block. We set the transmit interval of the echo commands to $20\,\mathrm{ms}$.

Table II shows the set and measured latency of the code block for different code block latency settings and external traffic conditions. Figure 13 plots the error in latency measurement. We observe that measurement error is within 10% for all cases.

E. Performance of Edge Intelligence

To demonstrate how edge intelligence can improve performance in TCPS applications, we devise and perform the following set of experiments. In these experiments, through an *embsys-app* at *ms-com*, we alternatively drive the gripper

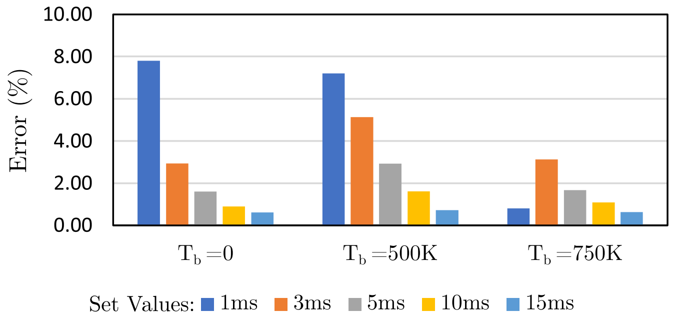

Fig. 13. Error in latency measurement for different set values of code block latencies and traffic conditions.

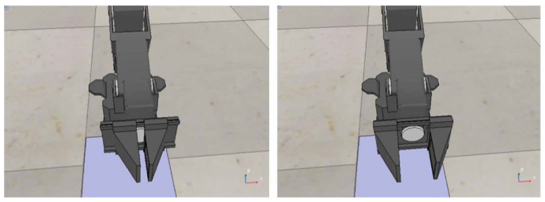

Fig. 14. Left and right figures show the closed and open positions, respectively, of the PhantomX robot gripper.

position of the remote-side robot in VREP between two states — open and close. See Figure 14.

For the experimental setup, we configure the testbed as in Figure 2 with support for Mininet. In Mininet, we simulate four hosts to run testbed components and use point-to-point links to connect these hosts. We use the first host to run *ms-embsys-app* and *ms-com*, second host to run *srv*, third to run *ss-ei* and the last one to run *ss-com* and *ss-embsys-app*. Throughout the experiment, we disabled the *ms-ei*.

In the experiments, we switch the robot's gripper position between the open and close states at an interval of 100 ms. In the first experiment, we simulate a lossless link connecting different testbed components. The leftmost graph in Figure 15 shows how the signal corresponding to the gripper position at the robot-side varies over time. As we expect, the gripper position alternates between a lower and higher value corresponding to the close and open states at an interval of 100 ms. In the second experiment, we model the last-mile link, the link connecting *ss-ei* and *ss-com* as a lossy link and every other link as lossless links to model TCPS applications that use wireless technologies for the last-mile link. For the experiment, we consider a packet-loss percentage of 50. Although high, we choose this value to easily visualize the effect of the lossy last-mile link on gripper positions. The center graph in Figure 15 shows how the gripper position at the robot-side varies over time. We see that unlike the leftmost graph, in the centermost graph, the gripper position does not switch at regular intervals; this is because of the missing packets at the robot-side due to the lossy link.

In the third and final experiment, we use the teleoperator-side edge intelligence module, *ss-ei*, to prevent the last-mile link losses from affecting the gripper positions. In *ss-ei*, we run a simple forecast algorithm, a zero-order hold predictor that runs at a frequency of 1 KHz. The zero-order hold predictor running at high frequencies duplicates data packets in the last-mile link and counteracts any packet losses. We see that unlike in the center graph in Figure 15, in the rightmost graph, which is the case with edge intelligence, the gripper position switches at regular intervals as expected from a lossless last-mile link.

Enabling edge intelligence modules, even if they run simple prediction algorithms such as a zero-order hold predictor, can be useful in many TCPS applications. TCPSbed with support for operator-side and teleoperator-side edge intelligence components will enable the developers to add edge intelligence support to their TCPS applications with ease. Developers using TCPSbed can concentrate on designing prediction algorithms instead of spending time on integrating edge intelligence in TCPS.

F. Step Response Experiments

We claim in Section III-B that the step response curve parameters such as rise-time and overshoot are affected by the TCPS component and network characteristics. To demonstrate this behaviour using TCPSbed, we conduct the following step response experiments. For the experiment, we use three PCs. We run *ms-com* and *srv* on PC-1, *emu* on PC-2, and *ss-com* on PC-3. *emu* simulates an ns-3 network with configurable RTT and packet drop values. For conducting the step response experiments, we interface the PI controller to *ms-com* by running *ms-embsys-app-PIController* in PC-1. Further, to simulate step disturbance, we use *ss-embsys-app-stepInput*. *ss-embsys-app-stepInput* is interfaced to *ss-com* and is run on PC-3. We ensured that the PC's selected are sufficiently fast to introduce negligible impact on the resultant step responses.

Figure 16 shows the step response curves for different network RTT and packet drops. These results confirm that RTT and packet drops affect the step response curves. Any application prescribes limits on rise times and overshoots. It is now possible to determine the maximum RTT and packet drop rate to keep the overshoot and rise time below the prescribed values. For instance, for the implemented TCPS and the sample PI controller coefficients in use, packet drops must not exceed 30% if the maximum allowed overshoot is 20% (see *graph 3* in Figure 16).

TCPS Quality Assessment: We can model a TCPS as a control system with multiple control loops, each associated with a different feedback modality such as audio, video and haptic. Depending on the use case, one or more of these control loops will be critical. Analyzing the performance of such critical control loops will provide an insight into the overall quality of TCPS. Our proposed step response analysis method will be useful here; in particular, we may use the resultant step response parameters such as rise times, overshoots and undershoots to construct a one-stop metric to evaluate and compare TCPS performances. However, such a study requires that each TCPS outputs a unique step response profile. We believe that generating such unique step response profiles is possible by tuning the responsiveness of the reference controller used in step response experiments. A possible way to tune the controller would be to produce step responses with the shortest rise time for specific overshoot and undershoot limits.

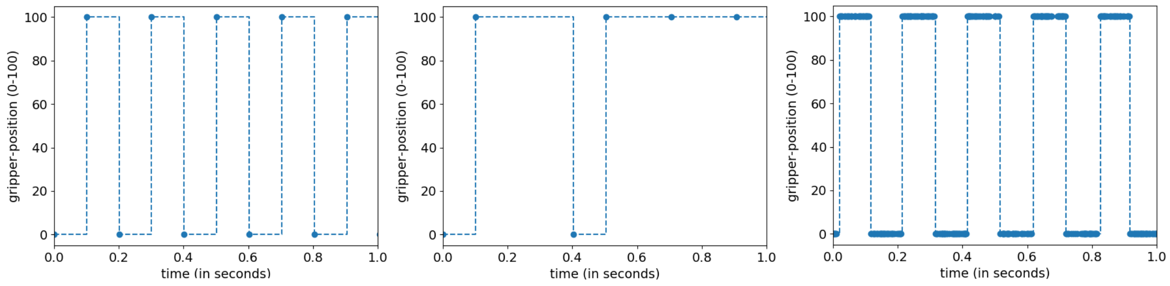

Fig. 15. Figures show how the gripper position signal passed to the PhantomX robot varies over time for different experimental settings. The leftmost figure is for the case where the last-mile link is lossless. The center figure is when the last mile link suffers a 50% packet loss, and the rightmost figure is when the last mile link suffers a 50% packet loss and with teleoperator-side edge intelligence.

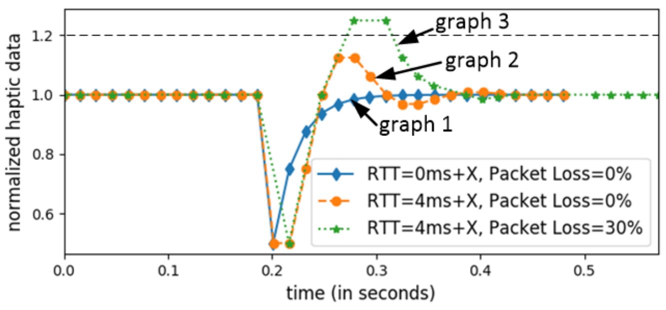

Fig. 16. Step response for different RTT and packet drop percentages. X represents the component of RTT that is not part of emulation. The change in pressure ($1 \longrightarrow 0.5$) occurs at $\approx 0.2\,\text{s}$.

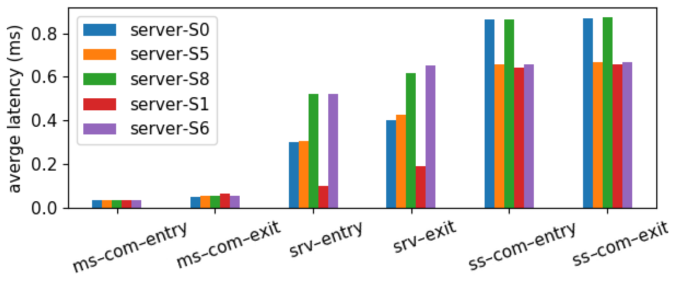

Fig. 17. Average latency corresponding to different echo points and for different server locations. For all the latency bars in the graph, the margin of error is within 5%.

Recently, after publication of our initial work [1], we have extended the step response characterization of TCPS further. In [5] and [50], we standardize step response characterization of TCPS. We also use step response characterization to formulate a metric called Quality of Control (QoC). We use QoC as an indicator to grade TCPS performances. Towards this, we extensively evaluate and characterize the step response method.

VI. Testbed Applications

In this section, we demonstrate a few potential applications of TCPSbed. Specifically, we illustrate how to use TCPSbed to determine optimal placement of TCPS components and to analyze the effect of interrupt coalescence and buffering in network device drivers and network nodes on TCPS latencies.

A. Effect of Server Placement

In many TCPS applications, we may have the option to place *srv* at different network locations. Traditional network tools such as ping and traceroute will not be able to determine which of these locations is appropriate for placing *srv* for the following reasons. First, these tools can measure only network latencies. They will not be able to evaluate the non-networking latencies, such as the overhead involved between *srv* application to network interfaces or the *srv* processing times. Second, these tools may not capture the latencies of the path which the application might take. Third, these tools cannot evaluate the performance during a live TCPS run, which is necessary to study the effect of external traffic on TCPS performance.

To circumvent the above issues with traditional latency measurement approaches, we propose using TCPSbed's latency characterizer. This experiment's objective is to demonstrate how to use TCPSbed's latency characterizer to determine the best network location to place *srv*.

For the experiment, the network, as shown in Figure 9, is simulated in Mininet. *ms-com* is run in the host connected to switch S_0 and *ss-com* in the host connected to switch S_8. We used TCPSbed's latency characterization method to measure the average latencies experienced by kinematic packets corresponding to different echo points by running *srv* in different hosts connected to different switches.

Figure 17 shows the results. One can use these results to interpret the best placement strategy for *srv*. For instance, placing *srv* in hosts connected to switches S_1, S_6 and S_5 minimizes the end-to-end latency in comparison to placing it in hosts connected to S_0 and S_8.

Remark: In the figure, the difference in the entry and exit latencies of *srv* is significant in comparison to *ms-com* and *ss-com*, because, the kinematics and the inverse kinematics algorithms are run in *srv*.

B. Effect of Interrupt Coalescence and Buffering in the Network Device Drivers

In this experiment, we study how one flow in a TCPS affects others due to interrupt coalescence and buffering at network device drivers. In particular, we study how traffic

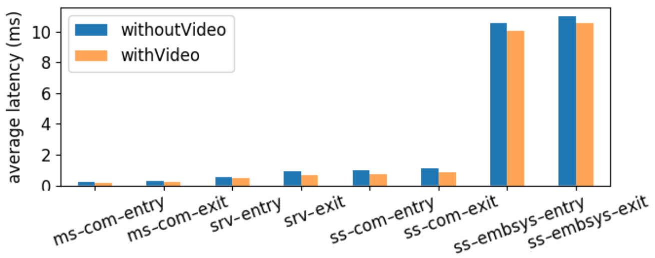

Fig. 18. Effect of enabling video on the average latencies of the kinematic data packets. For all the latency bars in the graph, the margin of error is within 5%.

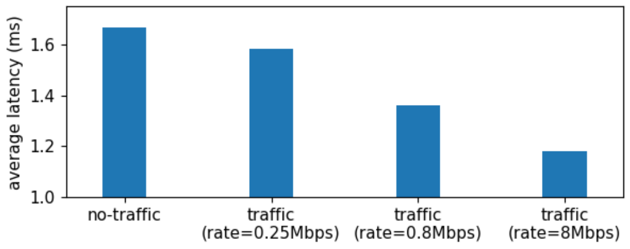

Fig. 19. How average latencies of the kinematic packets corresponding to the echo point *ss-com-exit* varies for different transmit rates of the simulated external traffic. Packet size of the external traffic is set to 1 KB. For all the latency bars in the graph, the margin of error is within 5%.

in the bandwidth-intensive video flow affects the latencies of packets in the kinematic flow. The experimental setup consists of two PCs, PC-1 and PC-2, connected by a point-to-point Ethernet link. *ms-com* and *srv* are run in PC-1 and *ss-com* in PC-2. We consider two cases. In the first case, the latencies of packets in the kinematic flow are measured without enabling the video flow. In the second case, the latency measurements are repeated with video flow. In both cases, we use an unmodified Linux kernel in hosts and use TCPSbed's latency characterization method to measure latencies. We transmit video in MJPEG format at a frame rate of 30 fps and with a JPEG frame size of 20 KB from the *tactile-teleoperator* to the *tactile-operator*.

Figure 18 shows the result. We find that, with video, the average latency of kinematic packet reduces. The decrease in average latency is non-intuitive. However, we can explain this phenomenon as follows. The network drivers do not transfer IP packets to the upper layer for every short packet it receives. Instead, the received packets accumulate the received packets until they cross a threshold (or when a time-out happens) [51], [52]. When the video flow is enabled, the video packets being large cause the threshold to cross more frequently, resulting in the network driver to transfer packets to the upper layer more often, thereby reducing latency.

To confirm our hypothesis, we replicate the above behaviour by transmitting simulated external traffic instead of video. Our hypothesis that the unnecessary buffering of packets contributes to the decrease in average kinematic packet latency will be correct if, in experiments, we find that average latency tends to decrease with an increase in the external traffic bit rate. Figure 19 shows these results. As we expect, increasing the external traffic bit rate decreases the average latency. Note, however, that we expect this behaviour only up to a certain intensity, beyond which the external traffic will choke the buffers and cause the latency to shoot up.

Interrupt coalescence and buffering of packets in Linux hosts are a result of NAPI in network device drivers [51]. Since having low latency is critical in TCPS applications, we recommend disabling NAPI in network device drivers of all hosts along the TCPS transmit path.

C. Effect of Bufferbloat

From the previous experiment, we conclude that the buffering of packets in hosts is harmful to TCPS. In TCPS, apart from hosts, the buffering of the packets also occurs in the network infrastructure (e.g., switches and routers). This experiment's objective is to determine how the buffering in switches affects TCPS performance, particularly TCPS packet latencies.

Depending on the switch buffer size, link capacities, and the external traffic, TCPS packets can experience different latencies. Often higher traffic rates in combination with large buffer sizes can clog the buffers resulting in bufferbloat. When bufferbloat occurs, packets will experience high latencies for extended duration of time till a congestion control algorithm steps in and limits the flows and until the switches drain out the packets pending in their buffers. The vulnerability of bufferbloat is that it can be triggered even by a single misbehaving network flow. TCPS flows, being latency-sensitive and used in critical applications such as telesurgery, cannot tolerate such bufferbloat, even if it appears intermittently.

A possible solution to avoid bufferbloat, even in the presence of a misbehaving flow, is to limit the buffer sizes. However, limiting buffer sizes in switches can result in packet drops. For many latency-sensitive applications such as in TCPS, it may be preferable to experience intermittent packet drops rather than to experience bufferbloat, which can affect the packet latencies for an extended duration of time.

We perform the following experiment to demonstrate how buffer sizes in switches affect the latencies of TCPS packets. We use Mininet to simulate the network topology circled in red in Figure 9. The hosts coloured in white in the circled network portion are used to simulate external traffic at a constant bit rate of 4.7 Mbps to each other.

Figure 20 shows how the latency experienced by TCPS kinematic packets traversing from the operator to the teleoperator side varies with buffer sizes and with different TCPS transmit rates. As expected, higher transmit rates and larger buffer sizes result in higher packet latencies. Limiting buffer size in switches to the size of two IP packets ensures that even at a TCPS kinematic packet transmit interval of 1 ms, the latency is under control (≈ 1 ms). However, this setting also increases the packet drop percentage by $\approx 20\%$.

Bufferbloat can be avoided by constraining buffers in the network switches. However, the traditional methods of constraining switch buffers per-port will degrade the performance of non-TCPS flows such as video streaming and media downloads that demand large buffers. A potential solution is to isolate TCPS flows from non-TCPS flows and to constrain

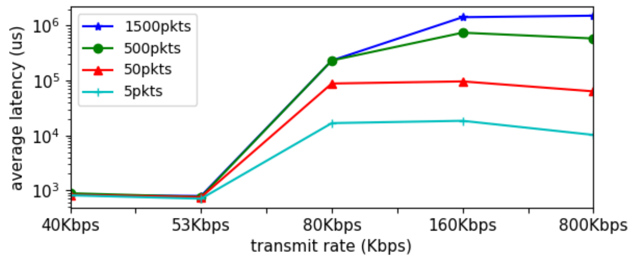

Fig. 20. Average latencies experienced by kinematic packets for different buffer sizes and TCPS packet transmit rates. For all the latency points in the graph, the margin of error is within 5 %.

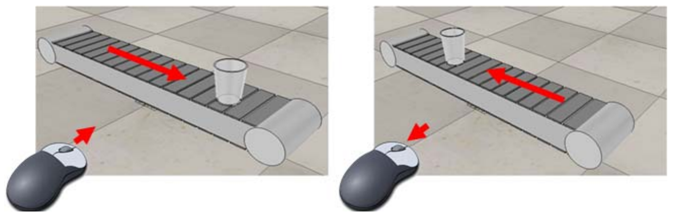

Fig. 21. The left and right figures capture snapshots of the clockwise and anticlockwise motion of the conveyor belt simulated in V-REP with a dummy glass object on the top. The operator uses the wheel of the computer mouse connected to *ms-com* to control the speed and the direction of the conveyor belt.

switch buffers per-flow such that fewer buffers are allotted for the TCPS flows [53], [54].

D. Support for Different TCPS Applications

TCPSbed supports implementation of different TCPS applications. We have already discussed use of TCPSbed to run a TCPS application where an operator controls a remote-side robot using a haptic glove (see Section V-A). In [55] we demonstrate use of TCPSbed to run a TCPS application where an operator controls a virtual robot simulated in V-REP using a computer mouse. In this section, as a proof of concept, we demonstrate use of TCPSbed to implement a non-robotic TCPS application where an operator controls the direction and speed of a remote-side conveyor belt using a computer mouse. Towards this, we use the experimental setup in Figure 9 with the following modifications. We simulate a conveyor belt in V-REP at *ss-com*, and we interface a computer mouse at *ms-com*. We enable the kinematic data path and update its APIs to map the mouse wheel movements to the speed and direction of the conveyor belt (see [55]). We demonstrate this application in Figure 21.

VII. Conclusion

We have designed and implemented a testbed, TCPSbed, for Tactile Internet-based Cyber-Physical Systems (TCPS). The main objective of TCPSbed is to enable quick prototyping and evaluation of TCPS applications. Our initial learnings from using TCPSbed can be summarized as follows: (i) in a TCPS, both networking and non-networking latencies, in particular, the delay associated with robot actuation, play a significant role. (ii) buffering of packets, at the switches or at the host, affect TCPS packet latencies. (iii) traditional best-effort networks in presence of external traffic cannot guarantee bounds on TCPS packet latencies. (iv) using edge intelligence components to predict kinematic and haptic signals at the network edges can significantly improve TCPS applications' performance even if the edge intelligence components consist of simple prediction algorithms. (v) step response curves of a TCPS implementation are affected by its components. Designing a meaningful metric to assess the quality of TCPS implementations using step response experiments is thus warranted.

We envision that TCPSbed will find its usage in all stages of TCPS development. One can use TCPSbed's latency and stability analyzer tools to perform suitability analysis of networking and non-networking components for prototyping TCPS applications. In the prototyping stage of a TCPS, TCPSbed's tools can be used to co-optimize the hardware, software and algorithms delays to meet the requirements of the application. Further, TCPSbed's edge intelligence components can be used to experimentally determine the right prediction algorithm to apply for a given network scenario. Finally, during the production stage of a TCPS, TCPSbed can be used as a platform to calibrate several of the TCPS components.

In the same vein as many open wireless networking testbeds, we have built TCPSbed modularly, which enables others to modify it as per their needs. Our intercontinental experiment shows that we can use TCPSbed remotely and can port it to different locations. We have also made TCPSbed's source code open through our GitHub page *TactileInternet* [55].

Acknowledgment

The authors thank Belma Turkovic for installing several *TCPSbed* components at TU Delft, which enabled their international experiment. They also thank Deepak Panchapakesan for assisting in the implementation of several TCPSbed components.

References

[1] K. Polachan, T. V. Prabhakar, C. Singh, and F. A. Kuipers, "Towards an open testbed for tactile cyber physical systems," in *Proc. 11th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2019, pp. 375–382.

[2] O. Holland *et al.*, "The IEEE 1918.1, 'tactile internet' standards working group and its standards," *Proc. IEEE*, vol. 107, no. 2, pp. 256–279, Jan. 2019.

[3] G. P. Fettweis, "The tactile Internet: Applications and challenges," *IEEE Veh. Technol. Mag.*, vol. 9, no. 1, pp. 64–70, Mar. 2014.

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

[5] K. Polachan, P. T. V, C. Singh, and D. Panchapakesan, "Quality of control assessment for tactile cyber-physical systems," in *Proc. 16th Annu. IEEE Int. Conf. Sens., Commun., Netw. (SECON)*, Jun. 2019, pp. 1–9.

[6] K. Polachan, B. Turkovic, T. V. Prabhakar, C. Singh, and F. A. Kuipers, "Dynamic network slicing for the tactile Internet," in *Proc. ACM/IEEE 11th Int. Conf. Cyber-Phys. Syst. (ICCPS)*, Apr. 2020, pp. 129–140.

[7] K. Polachan, C. Singh, and T. V. Prabhakar, "Decentralized dynamic gate scheduling of IEEE 802.1 Qbv time aware shaper and a TSN simulator for tactile cyber-physical systems," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 45–53.

[8] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 460–473, Mar. 2016.

[9] J. J. LaViola, "A discussion of cybersickness in virtual environments," *ACM SIGCHI Bull.*, vol. 32, no. 1, pp. 47–56, Jan. 2000.

[10] E. Steinbach *et al.*, “Haptic communications,” *Proc. IEEE*, vol. 100, no. 4, pp. 937–956, Feb. 2012.

[11] A. Nasrallah *et al.*, “Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research,” *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.

[12] D. van den Berg *et al.*, “Challenges in haptic communications over the Tactile Internet,” *IEEE Access*, vol. 5, pp. 23502–23518, 2017.

[13] M. Maier and A. Ebrahimzadeh, “Towards immersive tactile internet experiences: Low-latency FiWi enhanced mobile networks with edge intelligence [invited],” *J. Opt. Commun. Netw.*, vol. 11, no. 4, p. B10, 2019.

[14] Coppelia Robotics. (2018). *V-REP Virtual Robot Experimental Platform*. [Online]. Available: https://www.coppeliarobotics.com/

[15] NS-3. (2018). *NS-3 Consortium*. [Online]. Available: https://www.nsnam.org

[16] Mininet. (2018). *An Instant Virtual Network on your Laptop (or Other PC)*. [Online]. Available: https://mininet.org/

[17] X. Mai, J. Chen, Y. Wang, S. Bi, Y. Cheng, and N. Xi, “A teleoperation framework of hot line work robot,” in *Proc. IEEE Int. Conf. Mechatronics Automat. (ICMA)*, Aug. 2018, pp. 1872–1876.

[18] W. Pryor *et al.*, “Experimental evaluation of teleoperation interfaces for cutting of satellite insulation,” in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2019, pp. 4775–4781.

[19] E. Mendoza and J. P. Whitney, “A testbed for haptic and magnetic resonance imaging-guided percutaneous needle biopsy,” *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3177–3183, Oct. 2019.

[20] W. J. Book, H. Lane, L. J. Love, D. P. Magee, and K. Obergfell, “A novel teleoperated long-reach manipulator testbed and its remote capabilities via the internet,” in *Proc. IEEE Int. Conf. Robot. Automat.*, Apr. 1996, pp. 1036–1041.

[21] S. Hayati, T. Lee, K. Tso, P. Backes, and J. Lloyd, “A testbed for a unified teleoperated-autonomous dual-arm robotic system,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 1990, pp. 1090–1095.

[22] L. S. D. Pecly, M. L. O. Souza, and K. Hashtrudi-Zaad, “Model-reference model-mediated control for time-delayed teleoperation systems,” in *Proc. IEEE Haptics Symp. (HAPTICS)*, Mar. 2018, pp. 72–77.

[23] F. Gringoli, R. Klose, M. Hollick, and N. Ali, “Making Wi-Fi fit for the tactile internet: Low-latency Wi-Fi flooding using concurrent transmissions,” in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2018, pp. 1–6.

[24] J. A. Cabrera, R. S. Schmoll, G. T. Nguyen, S. Pandi, and F. H. Fitzek, “Softwarization and network coding in the mobile edge cloud for the tactile internet,” pp. 350–363, Feb. 2019.

[25] H. Cao, S. Gangakhedkar, A. R. Ali, M. Gharba, and J. Eichinger, “A 5G V2X testbed for cooperative automated driving,” *IEEE Veh. Netw. Conf. (VNC)*, Dec. 2016, pp. 1–4.

[26] V. Gokhale, K. Kroep, V. S. Rao, J. Verburg, and R. Yechangunja, “TIXT: An extensible testbed for tactile internet communication,” *IEEE Internet Things Mag.*, vol. 3, no. 1, pp. 32–37, Apr. 2020.

[27] SolarWinds. (2017). *Network Performance Monitor*. [Online]. Available: https://www.solarwinds.com/network-performance-monitor#features

[28] Visualware. (2017). *VisualRoute*. [Online]. Available: https://www.visualroute.com/

[29] A. Hamam and A. E. Saddik, “Evaluating the quality of experience of haptic-based applications through mathematical modeling,” in *Proc. IEEE Int. Workshop Haptic Audio Vis. Environ. Games (HAVE)*, Oct. 2012, pp. 56–61.

[30] M. Al Jaafreh, A. Hamam, and A. El Saddik, “A framework to analyze fatigue for haptic-based tactile Internet applications,” in *Proc. IEEE Int. Symp. Haptic, Audio Vis. Environ. Games (HAVE)*, Oct. 2017, pp. 1–6.

[31] A. Tatematsu, Y. Ishibashi, N. Fukushima, and S. Sugawara, “QoE assessment in haptic media, sound and video transmission: Influences of network latency,” in *Proc. IEEE Int. Workshop Tech. Committee Commun. Qual. Rel. (CQR)*, Jun. 2010, pp. 1–6.

[32] N. Sakr, N. D. Georganas, and J. Zhao, “A perceptual quality metric for haptic signals,” in *Proc. IEEE Int. Workshop Haptic, Audio Vis. Environ. Games*, Oct. 2007, pp. 27–32.

[33] C. G. Corrêa, D. M. Tokunaga, E. Ranzini, F. L. S. Nunes, and R. Tori, “Haptic interaction objective evaluation in needle insertion task simulation,” in *Proc. 31st Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, Apr. 2016, pp. 149–154.

[34] R. Chaudhari, E. Steinbach, and S. Hirche, “Towards an objective quality evaluation framework for haptic data reduction,” in *Proc. IEEE World Haptics Conf.*, Jun. 2011, pp. 539–544.

[35] MathWorks. (2018). *Step Info*. [Online]. Available: https://in.mathworks.com/help/control/ref/stepinfo.html

[36] B. Messner and D. Tilbury. (2018). *Control System Analysis*. [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section% =SystemAnalysis

[37] B. Messner and D. Tilbury. (2018). *Introduction: PID Controller Design*. [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section% =ControlPID

[38] *Cypress Semiconductors*. (2017). PSoC. [Online]. Available: https://www.cypress.com/products/32-bit-arm-cortex-m3-psoc-5lp

[39] OpenCV. (2017). *OpenCV*. [Online]. Available: https://opencv.org/

[40] PyAudio. (2017). *PyAudio*. [Online]. Available: https://pypi.python.org/pypi/PyAudio

[41] A. N., A. S. M., K. Polachan, T. V. Prabhakar, and C. Singh, “An end to end tactile cyber physical system design,” in *Proc. 4th Int. Workshop Emerg. Ideas Trends Eng. Cyber-Phys. Syst. (EITEC)*, Apr. 2018, pp. 9–16.

[42] Intel. *Intel Movidius Neural Compute Stick*. Accessed: Nov. 5, 2021. [Online]. Available: https://movidius.github.io/ncsdk/index.html

[43] Nvidia. *Nvidia Jetson*. Accessed: Nov. 5, 2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano

[44] Adafruit. (2017). *Vibrating Mini Motor Disc*. [Online]. Available: https://www.adafruit.com/product/1201

[45] (2020). *PhantomX Reactor, Trossen Robotics*. Accessed: Nov. 5, 2021. [Online]. Available: https://www.trossenrobotics.com/p/phantomx-ax-12-reactor-robot-arm.aspx

[46] Tcpreplay. *Bigflows.Pcap*. [Online]. Available: https://tcpreplay.appneta.com/wiki/captures.html

[47] *Tcpreplay Man Page*. Accessed: Aug. 22, 2021. [Online]. Available: https://tcpreplay.appneta.com/wiki/tcpreplay-man.html

[48] *Wireshark Go Deep*. Accessed: Aug. 31, 2021. [Online]. Available: https://www.wireshark.org/

[49] Y. Gao *et al.*, “JHU-ISI gesture and skill assessment working set (JIGSAWS) a surgical activity dataset for human motion modeling,” *Model. Monitor. Comput. Assist. Intervent. (MCAI)*, 2014, p. 3.

[50] K. Polachan, J. Pal, C. Singh, and P. T V, “Quality of control assessment for tactile Internet based cyber-physical systems,” 2019, *arXiv:1910.08743*.

[51] L. Foundation. *Napi*. Accessed: Nov. 5, 2021. [Online]. Available: https://wiki.linuxfoundation.org/networking/napi

[52] K. M. Salehin, V. Sahasrabudhe, and R. Rojas-Cessa, “Remote measurement of interrupt-coalescence latency of internet hosts,” in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, Jun. 2017, pp. 1–9.

[53] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, “Approximating fair queueing on reconfigurable switches,” in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Apr. 2018, pp. 1–16.

[54] R. Kundel, J. Blendin, T. Viernickel, B. Koldehofe, and R. Steinmetz, “P4-codel: Active queue management in programmable data planes,” in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2018, pp. 1–4.

[55] *Github—Tactileinternet/TCPSbed: A Modular Testbed for Tactile Internet Based Cyber-Physical Systems (v1.0)*. Accessed: Aug. 31, 2021. [Online]. Available: https://github.com/TactileInternet/TCPSbed

Kurian Polachan received the M.Tech. and Ph.D. degrees from the Department of Electronic Systems Engineering, Indian Institute of Science, Bengaluru, in 2014 and 2021, respectively. He has over five years of industry experience working at Cypress Semiconductors. He was a Senior Design Engineer from 2014 to 2015 and a Senior Applications Engineer from 2008 to 2012. His research interests span touch sensing technologies, time-sensitive networking, and tactile cyber-physical systems.

Joydeep Pal received the B.Tech. degree in electronics and communication engineering from Delhi Technological University (formerly DCE) in 2016. He is currently pursuing the Ph.D. degree with the Department of Electronic Systems Engineering, Indian Institute of Science, under the guidance of Dr. T. V. Prabhakar and Dr. Chandramani Singh. He has worked on the IoT systems for home automation. His research focuses on tactile internet systems.

Chandramani Singh received the M.E. and Ph.D. degrees in electrical communication engineering from the Indian Institute of Science, Bengaluru, India, in 2005 and 2012, respectively. From 2005 to 2006, he was with ESQUBE Communication Solutions Private Ltd., Bengaluru. From 2012 to 2013, he was a Research Engineer with TREC, a joint research team between INRIA Rocquencourt and ENS de Paris, Paris, France, and a Post-Doctoral Research Associate with CSL, University of Illinois at Urbana–Champaign, Champaign, IL, USA, from 2013 to 2014. He is currently an Assistant Professor with the Department of Electronic Systems Engineering (ESE), Indian Institute of Science. His interests include communication networks, data centers, and smart grids.

T. V. Prabhakar (Member, IEEE) received the M.Sc. (Engg.) degree from the Indian Institute of Science (IISc), Bengaluru, and the Ph.D. degree from TU Delft, The Netherlands. He is currently the Principal Research Scientist with the Department of Electronic Systems Engineering, IISc, where he is involved in networked embedded systems research. He is also with the Zero Energy Networks Laboratory, IISc, and specializes in building ultralow power embedded hardware and software network stacks. His research interests include RFID, device ID PUFs, energy harvesting and power management algorithms, tactile CPS infrastructure, digital twin, the IoT data management, speech decoding on embedded systems, and condition monitoring.

Fernando A. Kuipers (Senior Member, IEEE) received the Ph.D. degree *(cum laude)* from the Delft University of Technology (TU Delft) in 2004. He was a Visiting Scholar with Technion—Israel Institute of Technology in 2009 and Columbia University, New York City, in 2016. He is currently a Full Professor and the Head of the Laboratory on Internet Science, TU Delft. His research interests include network optimization, network resilience, quality of service, and quality of experience and address problems in software-defined networking, tactile internet, the Internet of Things, and critical infrastructures. His work on these subjects includes distinguished papers at IEEE INFOCOM 2003, Chinacom 2006, IFIP Networking 2008, IEEE FMN 2008, IEEE ISM 2008, ITC 2009, IEEE JISIC 2014, NetGames 2015, and EuroGP 2017. He is a member of the ACM SIGCOMM Executive Committee and the Vice Chair of the IFIP Working Group 6.2 on Network and Internetwork Architectures. He co-founded the Do IoT Fieldlab and the PowerWeb Institute at TU Delft and is a part of the Board of the TU Delft Safety and Security Institute.