Uncovering Robot Joint-Level Controller Actions from Encrypted Network Traffic

Cheng Tang(✉), Diogo Barradas, Urs Hengartner, and Yue Hu

University of Waterloo, Waterloo, Canada
{c225tang,dbarrada,urs.hengartner,yue.hu}@uwaterloo.ca

Abstract. This study examines the privacy risks faced during the teleoperation of robots which are controlled via commands exchanged through encrypted network communications. From the perspective of an adversary able to eavesdrop network traffic, we explore the potential to infer sensitive robotic actions by analyzing the metadata of encrypted traffic, such as packet timing, size, and direction. We investigate this threat in realistic setups, using a smartphone's Inertial Measurement Unit (IMU) to control a robotic arm via three joint-level modalities—position, velocity, and torque—to perform four distinct actions. First, by leveraging state-of-the-art traffic analysis attacks to examine robot teleoperation traffic, we uncover that an adversary can accurately identify the actions performed by the robot. Second, by prototyping a defense against such attacks, we analyze the performance and privacy trade-offs across the robot control modalities. Our findings highlight the importance of integrating privacy considerations into robotics APIs, and encourage further exploration of factors influencing the security of teleoperated systems.

Keywords: Encrypted traffic analysis · Privacy · Robot teleoperation

1 Introduction

Current research is pushing towards the adoption of teleoperated robots, particularly robot arms, in everyday settings, including homes [1] and production lines [4]. In this setting, the robot and the controller (i.e., the computer on which the control software is run) are typically connected to a network, often the internet [2,12], and the network traffic exchanged between the controller and the robot can typically be encrypted for preventing a network adversary to read or tamper with the commands sent by the controller.

Despite these safeguards, observers can still try to infer the content of encrypted messages by analyzing communication metadata, such as the size, direction, and timing of network packets. For instance, inference techniques have previously succeeded in identifying the websites users visit through encrypted tunnels [5,16] that aim to hide one's browsing patterns. However, this threat remains underexplored in robot teleoperation scenarios. If these inference techniques were as effective on encrypted robot/controller traffic as on web traffic,

Published by Springer Nature Switzerland AG 2025
L. Nemec Zlatolas et al. (Eds.): SEC 2025, IFIP AICT 745, pp. 121–135, 2025.
https://doi.org/10.1007/978-3-031-92882-6_9

adversaries could gain unauthorized access to private information which, in sensitive settings like home care, could reveal information about an individual's care regimen and health status (such as medication dispensing or physical therapy).

The *first objective* of our research is to showcase the practical feasibility of inferring robot actions from encrypted network traffic patterns. We leverage an intrinsic property of robotic teleoperation traffic: the unique network traffic pattern generated by each command and its associated feedback. For example, the completion time of a motion can be reflected by observing the frequency between network packets with specific lengths. Even with encryption, certain characteristics of this traffic pattern, such as the timing between packets, may allow an observer to differentiate between commands.

To experiment with these inference attacks, we employ a smartphone as the teleoperation interface [13,20] (similarly to [11]). We use the phone's Inertial Measurement Unit (IMU)'s gyroscope and accelerometer to control a collaborative robot arm. The smartphone is connected to a computer that is connected to the same network as the robot and commands the motions to the robot via three types of joint-level controllers: position, velocity, and torque. For instance, the end-effector (EE) of the robot arm follows the movements of the smartphone - if the smartphone is moved up and down, the robot arm's EE also moves up and down. For robot arms, all control strategies essentially boil down to joint-level controllers, which set specific values (e.g., positions, velocities, or torques) at joint level depending on the desired outcome. For instance, torque control might be more desirable when physical interactions with human users may occur, while position control might be prioritized for precision in movement.

The *second objective* of our research is to identify how different control strategies impact the privacy of teleoperated robotic systems. We posit that different joint-level controllers influence the type and granularity of data inferable from encrypted network traffic. For example, a controller adjusting joint positions predictably may make action inference easier from network metadata, compared to a torque-based control scheme that produces more variable traffic patterns. To the best of our knowledge, we are the first to study the prediction of robot actions from encrypted joint-level controller traffic.

In our experiments, we execute four distinct movements 100 times each using each of the three controllers and collect the encrypted network traffic transferred between the computer and the robot. We analyze different features of this network traffic, using them to train and evaluate different classifiers.

Contributions. We deliver the following main contributions:

- We investigate encrypted network traffic patterns from various angles, including the effects of command frequency (i.e., the frequency at which the robot is requested to perform a movement as part of an action) on packet sizes and arrival times, as well as the time it takes to complete commands.
- We assess the effectiveness of several traffic pattern recognition methods and identify reliable techniques for deducing robot actions from encrypted traffic.
- We explore the consequences of our research for the design of new privacy-preserving robot teleoperation APIs.

2 Related Work

2.1 Analysis of Encrypted Network Traffic

In communications established across the network, encryption protocols (e.g., TLS) ensure that data remains confidential and secure against unauthorized tampering. Despite the use of encryption, it is still possible to infer privacy-sensitive information pertaining to these data exchanges through the analysis of the *communication metadata*; characteristics such as the size, timing, and frequency of network packets comprising encrypted communications reveal patterns that allow external observers to infer communications' contents [7].

Traffic *fingerprinting* is a sub-category of network traffic analysis. In a fingerprinting attack, an adversary's goal is to re-identify the occurrence of a certain event in the network, based on the similarity to past events labeled by the adversary, i.e., a set of traffic "fingerprints" [5]. These attacks have been widely researched in the context of website fingerprinting [10,21], where an adversary wishes to uncover which website a client is accessing through an encrypted tunnel (e.g., Virtual Private Networks (VPNs) [21] or Tor [10]) that hides the website's IP address from a local network observer.

2.2 Fingerprinting Encrypted Robotic Teleoperation Traffic

Teleoperated robots are controlled remotely via network commands, facing privacy challenges that bear a notable resemblance to those encountered in website fingerprinting. Specifically, traffic fingerprinting techniques in robotic teleoperation can expose fine-grained aspects of robot operations, ranging from movement patterns to user interactions, thereby posing a considerable privacy risk to users. Shah et al. [14] demonstrated that robot movements can be identified from encrypted traffic using deep neural network-based analysis. Adversaries can extract detailed operational sequences of robots, potentially revealing sensitive information, or unauthorized replication of tasks. In earlier work [17], we classified robot actions using higher-level APIs like Cartesian commands, achieving high accuracy through signal processing and traffic statistics analysis. However, these studies did not explore classifying daily actions and tasks using joint-level controllers, the most fundamental type of robotic arm controllers, thus leaving a critical gap in existing research.

As it stands, not all tasks can use high-level descriptions, especially in dynamic environments like homes or care facilities where human-robot interaction may occur. Unlike industrial robots, robots in these settings must adjust in real-time, needing to target delicate handling, human contact, or complex environments. For example, in rehabilitation, a robot assisting with arm lifting needs to continuously adapt its torque and position based on the patient's movements, requiring low-level control for smooth and safe interaction. In these cases, high-level controller commands need to be translated into precise joint-level position/velocity/torque commands that call for real-time sensor feedback. While Shah et al. [14] inferred low-level motions from encrypted traffic, they did not explore the inference of high-level actions through the composition/analysis of low-level motions, raising the

Fig. 1. The smartphone sends signals to the robot control software on a computer via cable, and the computer communicates with the robot over the same network.

question of whether and to which degree such relationships can be established. In our work, we show that it is possible to predict high-level actions from encrypted network traffic using joint-level controllers.

3 Experimental Setup

3.1 Hardware and Software Configurations

Our experimental setup leverages a smartphone equipped with an accelerometer, gyroscope, and magnetometer, which we use to teleoperate a commercial collaborative robot arm (Sawyer from Rethink Robotics) as depicted in Fig. 1.

Controllers Used in Teleoperation. We implemented teleoperation using the three types of joint-level controllers available on the Sawyer: position, velocity, and torque control, commonly used in commercial robot arms to move robot arms, where position and velocity control are desired for precision tasks and torque control for tasks requiring interactions with humans and/or delicate objects. The IMU data are collected from the smartphone to estimate its position and velocity, then, depending on the desired controller, translated into the robot arms' joint position, velocity, and torques for the arm to move with the smartphone. Ultimately, we used a single phone to collect network traffic generated via teleoperation (see Sect. 3.3). Despite variations in sensor performance, our goal is to capture valid trajectories. Minor sensor differences do not affect overall motion, ensuring coherent trajectories for analysis.

Teleoperation Software and Secure Communication. The control of Sawyer is facilitated through API[1] that enables the generation of command patterns for effective robot operation. To establish an encrypted communication channel between the robot and the computer, we used the built-in OpenVPN features of a commercial-off-the-shelf router. Commands are processed on this computer, running Ubuntu 20.04 LTS, which communicates with the robot via ROS (Robotic Operating System) Noetic, a standard framework in robotics. The robotic arm and computer are connected to the same router via Ethernet, ensuring reliable data transmission and real-time command execution/feedback.

[1] https://rethinkrobotics.github.io/intera_sdk_docs/5.1.0/index.html.

3.2 Threat Model

Our threat model assumes a network adversary who is able to intercept the encrypted network traffic flowing between the computer and the robotic arm (e.g., by wiretapping a physical Ethernet link). The adversary aims at identifying the (potentially privacy-sensitive) actions being performed by the robot but is unable to break the cryptographic assumptions that would allow it to reveal the actual content of the encrypted communication channel.

To uncover the actions performed by the robot, the adversary will resort to encrypted traffic analysis techniques based on the inspection of the communication metadata exchanged between the controller and the robot, e.g., the timing, command frequency, and size of network packets exchanged in the communication, as well as the overall volume of exchanged data. Similar to the website fingerprinting literature, we assume that the adversary can replicate the operational setup of their target – in this case, including network conditions, the use of the same robot models, and the use of the same encryption techniques to establish a secure communication channel between the controller and the robot.

In particular, the adversary will attempt to re-identify the occurrence of a set of actions by the target robot as follows: First, the adversary generates a comprehensive database of network traces ("fingerprints") by executing a set of actions that closely resemble the ones that may be executed by the target robot in the future. Then, the adversary waits for the target robot to perform an action whose information is exchanged via the encrypted tunnel, collects the robot's traffic, and attempts to match it with one of the fingerprints in its database. This matching is performed via machine learning [5] and deep learning [10,16] techniques, which help automate and enhance the fingerprinting's effectiveness.

Traffic fingerprinting typically focuses on two different scenarios (referred to as *closed-* and/or *open-world* [16]), depending on the information made available to the adversary. In this work, we assume a *closed-world* scenario, where an adversary is given the knowledge about a limited set of actions a robot is able to perform (also known as the *monitored* set). While the set of monitored actions is limited, it may be sufficient to allow an adversary to infer a robot's activity in specific contexts. For instance, one may assume that a caregiving robot would perform a few pre-determined actions for helping a patient with reduced mobility, but that a different set of actions would be more relevant for a visually impaired patient. Should the adversary know which condition afflicts a patient, they could reasonably create a database of actions expected to be performed by the robot.

3.3 Dataset Collection

Choice of Robotic Actions. We collected network traces [18] while the robot performed *four* distinct actions: (a) picking up and placing an object; (b) pouring water; (c) pressing a key, and; (d) turning on a switch. This set of actions was selected based on the assumption that the robot may be operating in a private setting, where the four selected actions are highly representative of recurring

daily routine actions. While these actions may seem basic, they may reveal user habits, leading to the exposure of private information and user profiling.

Among the four actions, *pour water* is mostly similar to *pick and place*, while *press key* is mostly similar to *turn on switch*. We deliberately chose two pairs of actions that could be considered dissimilar to one another, while internally similar within each pair. This approach allows for exploring how subtle differences in the network traffic generated by joint-level commands can help distinguish between similar actions, as well as demonstrating clear distinctions between different types of movements. This selection helps us evaluate the effectiveness of fingerprinting across varied, yet closely related, robot behaviors (Sect. 5.2).

Robotic Actions' Trajectories. To ensure our traces provide a robust basis for assessing fingerprinting performance, we collected each action sample following a *different trajectory*. In essence, action trajectories are generated via the execution of several sub-tasks (e.g., move to a given location, press/lift an object, etc.) that are chained together and executed at different instants in time. Each sub-task may be configured to operate at different robot movement speeds, to assume disparate movement waypoints and object locations, and other task-specific parameters (e.g., pouring angles). In addition, each trajectory may involve additional variability due to minor variations in sensor readings (see Sect. 3.1).

For each action class, we collected and interpolated 100 sample motion trajectories using the phone, each lasting 10 to 30 s, totaling 400 data samples. One traffic sample encompasses the entire network data exchanged between the robot and the controller PC when completing an instance of the desired action.

Per-Controller Data Acquisition. We used the interpolated motion samples to operate the robot using our three different controllers: position, velocity, and torque. During these operations, we captured the network traffic exchanged between the robot and the computer. This process resulted in 400 network traffic trace samples for each type of controller, amounting to a total of 1200 network traffic trace samples. We acquired the network traces by running the Linux `tcpdump` utility on the robot's controller computer, which allows us to record the encrypted data exchanges between the robot and the controller.

4 Characterization of Robot Actions Traffic

In our testbed, the command and feedback messages necessary for teleoperation are encrypted by VPN software, preventing adversaries from reading their content. However, different actions might produce distinct network patterns. Commands requiring more parameters may generate larger packets; different actions may require different command sequences, resulting in varying packet sizes; and different actions might require command and feedback exchanges at different frequencies, affecting packet inter-arrival times. These possibly disparate traffic patterns could allow an adversary to match them to specific actions.

To better visualize the existence of such patterns in teleoperation traffic, Fig. 2 presents a set of example network traces collected during the execution of different actions. (The traces comprising each action/controller combination

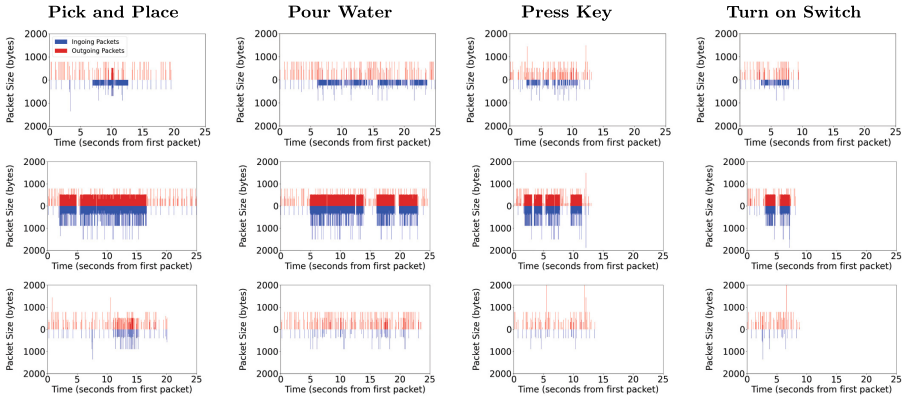


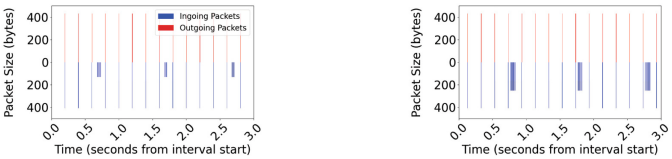
Fig. 2. Examples of four actions' packet lengths over time. (First row – position controller; second row – velocity controller; third row – torque controller).

were found to be rather stable.) Each row of the figure depicts the packet sizes and timing relationship for the position, velocity, and torque controllers. In this figure, "incoming" refers to the packets sent by the controller and received by the robot, while "outgoing" refers to packets sent in the reverse direction.

Different Controller/Action Combinations Produce Disparate Traffic Patterns. A closer analysis of Fig. 2 allows us to pinpoint two major observations: a) row-wise, each controller originates a different traffic pattern across each of the actions, and; b) column-wise, for the same action, each controller originates a different network traffic pattern. Overall, this means that each action performed by each different controller will exhibit a different traffic pattern in the network, potentially allowing traffic fingerprinting techniques to identify the traces produced by a specific controller/action combination.

Consider, for instance, the first row of Fig. 2, which showcase the traffic patterns produced by the position controller over each of the four actions. For each action, we can see that not only each action is performed in a different amount of time, but also that the frequency of packets exchanged between the robot and its controller is quite disparate. For instance, for the *pick and place* action, there is a high concentration of incoming packets from around 7 s to 12 s in the trace, while incoming packets for the *pour water* action form four individual clusters after 5 s into the trace, where these packets are more concentrated. The traffic patterns generated by other controllers given the same action follow a similar trend, i.e., the length of the packet exchange, as well as the frequency of packets exchanged at different points in the trace is indicative of the action being performed. Now, consider also the first column of Fig. 2, showcasing the traffic patterns for the *pick and place* action across three different controllers. We can observe that each controller will also produce a different pattern.

Packet Sizes Reflect the Number of Joints Being Commanded. Figure 3 depicts a comparative analysis of command transmissions when controlling a



(a) Command one joint. (b) Command all joints.

Fig. 3. Packet lengths over time when commanding a different number of joints.

single joint versus all seven joints of the robot. Note that sending one command to a given joint essentially requires the controller to specify a value for one key in a dictionary. Thus, the more joints are controlled simultaneously, the more information needs to be conveyed by the controller to the robot. Indeed, we can observe from the figure that packet lengths experience a noticeable increase when all seven joints are being controlled. This trend is also consistent across position, velocity, and torque control modes. Interestingly, this means that network traffic analysis techniques can also focus on these particularities to infer whether simpler/more complex actions are being performed, depending on the number of joints used by the robot at any given time. For instance, in a simple horizontal motion of the arm, only one joint needs to be moved. In this case, the controller would send smaller packets for commanding the movement of a single joint.

5 Evaluation of Fingerprinting Attacks

5.1 Traffic Analysis Attacks and Metrics

Classical ML-Based Attacks. These attacks work by analyzing a set of manually-engineered features extracted from encrypted network traffic. These features include summary statistics such as the average packet size and percentiles of inter-packet timing and are used to identify patterns that can distinguish different types of traffic despite encryption. We use three ML-based classifiers (initially proposed for website fingerprinting) that operate on summary statistics, including k-Fingerprinting [5], CUMUL [9], and k-Nearest Neighbors (kNN) [19]. We also devised our own classifier based on gradient-boosted decision trees (XGBoost), resorting to summary statistics focusing the frequency distribution of packet sizes and timing, as well as each action's communication volume.

Deep Learning Attacks. Unlike classical ML attacks which require manual feature engineering, deep learning-based attacks automatically extract features from network traces by using kernels as filters during the backpropagation process. These methods are generally more resource-intensive, requiring significant computational power and large training datasets. While acquiring such datasets is relatively straightforward in domains like website fingerprinting, it proves challenging in the context of robotic operations, where collecting large amounts of

real-world data requires extensive work [8]. Nevertheless, we explore these classifiers within our study, considering their success in network traffic classification.

We use two deep learning attacks based on Convolutional Neural Networks (CNNs), named deep fingerprinting (DF) [16] and Tik-Tok [10]. While both classifiers share the same neural network architecture, they use different input representations of the network data they are trained/tested on. Specifically, DF leverages an input vector formed by the *direction and size* of traffic (e.g., [+50, −50, +80] denotes a trace formed by an outgoing packet of size 50, an incoming packet of size 50, and an outgoing packet of size 80). Instead, Tik-Tok considers a *directional timing* input representation (e.g., [0, −0.02, +0.34] denotes a trace formed by the first outgoing packet that is sent to the network, followed by an incoming packet that arrives 0.02 s after the first packet was sent, and so on).

In addition to the "vanilla" inputs for both classifiers, we generated two alternative input representations. Specifically, we started by generating an alternative input representation for Tik-Tok, where we simply use raw packet timings (e.g., [0, 0.02, 0.54] denotes a trace where the second packet exchanged in the connection was sent after 0.02 s of the start of the connection). Then, for assessing the potential benefits of including all three of the metadata dimensions we consider for a network packet (direction, timing, and size), we crafted new input vectors containing these dimensions. We feed this hybrid feature vector into the generic DF/Tik-Tok architecture for building a new classifier we refer to as DF-Tok.

Lastly, we use another recent attack known as Robust Fingerprinting (RF) [15]. This attack aggregates the direction and timing information of packets (according to fixed-length time intervals) into a matrix and uses a CNN-based classifier.

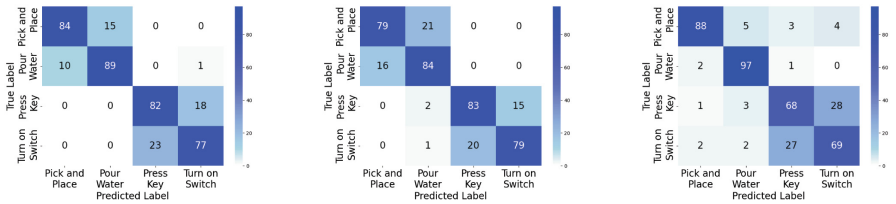
Classification Metrics and Setup. We evaluate the success of each attack through its *accuracy* and use *confusion matrices* to examine predictions across different classes. We used 10-fold cross-validation when experimenting with classical ML classifiers and, due to computational demands, used an 80% training/10% validation/10% testing splits when evaluating our deep learning classifiers. All results refer to the accuracy obtained on each classifier's testing set.

5.2 Attack Results

Table 1 depicts the accuracy of each attack when identifying the action performed by the robot under a given controller. These results showcase significant privacy risks (accuracy surpassing 80%), suggesting that an adversary could potentially identify the actions executed by the robot by using already existing fingerprinting attacks. Below, we describe our main findings.

Table 1. Accuracy of each attack in distinguishing between the four different actions across each of the controllers.

Attack	Position (%)	Velocity (%)	Torque (%)
k-Fingerprinting	77.2	63.5	72.5
KNN	33.2	31.0	27.5
CUMUL	63.6	61.0	63.8
XGBoost	83.2	81.2	80.5
DF (direction, sizes)	67.5	62.5	48.8
Tik-Tok (direction, timing)	28.8	27.5	28.8
Tik-Tok (timing)	28.8	27.5	28.8
DF-Tok (direction, timing, sizes)	67.5	63.2	55.0
RF (direction, timing)	38.0	37.8	37.4

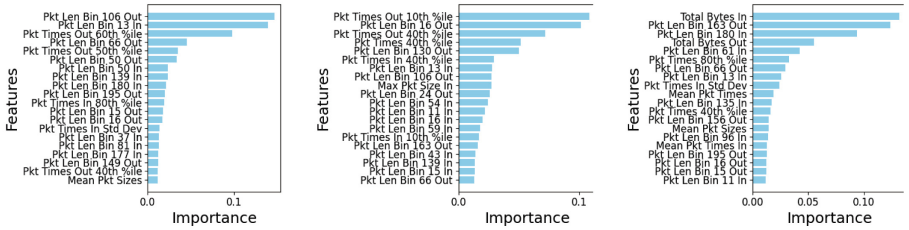


(a) Position Controller. (b) Velocity Controller. (c) Torque Controller.

Fig. 4. XGBoost's confusion matrices for the three controllers.

XGBoost Performs Best in Uncovering Robot Actions from Encrypted Network Traffic. Figure 4 displays the confusion matrices for the three controllers using the XGBoost classifier, which yielded the highest accuracy. The actions of picking and placing an object and pouring water tend to overlap in terms of their characteristic patterns, which can lead to these actions being more easily confused with one another during classification. Similarly, the actions of pressing a key and the tapping motion involved turning on a switch and exhibit a higher likelihood of being mixed up. This observation aligns with the intuitive understanding that actions sharing similar physical movements or operational characteristics are more challenging to distinguish based on traffic patterns.

The Performance of Deep Learning Attacks was Subpar When Compared to Those Based on Classical ML. Despite the theoretical advantages of CNNs, the performance of the deep learning attacks we considered may have been limited by several factors. First, these attacks were proposed for the website fingerprinting setting, where data collection is a rather easy process to automate and conduct at scale, whereas this is not the particular case for teleoperation. Second, while Tik-Tok and RF take packet timing into consideration, they forego packet sizing information, thus being unable to build important context about the trace by distinguishing between different packet types composing a trace (and



(a) Position Controller. (b) Velocity Controller. (c) Torque Controller.

Fig. 5. Top 20 feature importance for three controllers using XGBoost.

which may only be distinguishable according to their size). This may present a challenge to these classifiers, since, alongside the command messages we rely on to characterize each action, the controller and the robot exchange maintenance and feedback messages that are not useful for action identification. Indeed, DF-Tok's results suggest that timing information appears to provide only limited gains on top of DF with packet direction and size information. However, performance gains were evident when applied to the torque controller (∼6% increase).

In contrast, attacks that jointly consider statistics about both packet sizes and timing have demonstrated a compelling potential to unveil sensitive data. Classical ML attacks, including k-Fingerprinting or our XGBoost-based classifier, were effective in uncovering the actions executed by the teleoperated robot.

Different Controllers Produce Different Sets of Traffic Characteristics Useful for Action Identification. Fig. 5 showcases the top 20 most important features for classifying actions across the three different controllers, as determined by our XGBoost classifier. It can be observed that the position, velocity, and torque controllers are all influenced by the number of packets exchanged in a trace, featuring a mix of useful packet size ranges that are both shared and distinct across controllers. This is also true for some indicators of timing information between packets. Additionally, the total in/out bytes transmitted in the connection are relevant for the torque controller, but not for the other controllers. This is consistent with the third row of Fig. 2, where it can be seen that different actions tend to produce traffic sequences of varying length (e.g., the tap action is relatively short, under 10 s, whereas the pour water action is longer, over 20 s), suggesting that longer action sequences also generate a larger amount of packets, providing useful information to the classifier.

Some Actions Share Common Feature Distributions. When delving into the per-class distribution of the most important feature for each controller, we identified an overlap in the feature distributions for some classes, revealing some of the inherent difficulties in distinguishing between the network traces of different actions. For instance, consider the *10th percentile of outgoing packets' times* feature, identified as the most important feature for distinguishing actions performed with the velocity controller (Fig. 5b). We observed a clear similarity

between the distribution of this feature among the samples of the *pick and place* and *pour water* actions (both ranging from 4 to 6 milliseconds), and also between *press key* and *turn on switch* (2 to 4 milliseconds), whereas there is an observable difference between these two groups of actions. This is further evidenced by the results shown in the confusion matrix of Fig. 4b.

6 Towards Effective Fingerprinting Defenses

6.1 Requirements for Teleoperation Traffic Analysis Defenses

In designing software APIs for teleoperated robotic systems, several strategies can be considered to enhance privacy. As an inspiration, the literature on website fingerprinting is already prolific with the development of defenses to scramble traffic patterns so as to make fingerprinting attacks unsuccessful [6].

Accommodating Real-Time Constraints. Ideally, defense mechanisms should avoid compromising a robot's operational effectiveness. For instance, while strategies involving the padding of command packets to the same size and forcing packet delays might deter an adversary from identifying the robotic actions being performed, the bandwidth and latency overheads imposed by such strategies should not compromise the correct and timely operation of the robot. In contrast to web traffic, robot teleoperation is often ruled by real-time requirements which are not amenable to fingerprinting defenses that introduce latency. In general, if any data at the joint control level is delayed even by milliseconds, this would cause either the robot to stop (if such a feature is implemented on the onboard computer) or the controller could become unstable.

Zero-Delay Defenses. Defenses that avoid introducing delays have been recently explored in the website fingerprinting literature. For instance, FRONT [3] foregoes bandwidth savings, altering traffic patterns by introducing dummy packets. Another strategy could entail sending a set of commands to all joints, at a fixed rate, even in the absence of updates. This could mask the number of joints involved in motion execution (an information leak we underlined in Sect. 4).

6.2 Prototyping a Command Size Padding Defense

We prototyped a simple simulator for a zero-delay defense against robot traffic fingerprinting, applying it on the traces we have previously collected (Sect. 3.3).

Since our earlier feature analysis (see Fig. 5) suggested that packet sizes are amongst the most important features for classification, we built a defense that artificially pads joint-level command packets to their nearest multiple of n bytes. We applied the defense in several configurations, including n = 200B, 400B, 800B, or 1800B. Given that the largest command packets we observed are in a size range of 1600–1700B, the 1800B padding configuration essentially turns all command packets into the same size. Thus, this modification removes any useful information that could be extracted from the size of individual command packets

Table 2. XGBoost attack accuracy and bandwidth overhead (% of extra transmitted bytes) on padded command packets.

Padding (B)	Position		Velocity		Torque	
	Acc. (%)	OH (%)	Acc. (%)	OH (%)	Acc. (%)	OH (%)
200	80.7	121.1	72.9	132.9	79.7	120.7
400	79.7	136.1	72.5	164.5	79.3	137.1
800	78.5	182.2	73.4	252.2	78.7	182.6
1800	78.0	300.6	72.9	495.3	78.8	302.2

alone. Padding increases bandwidth consumption (see Table 2), but does not affect packet transmission time nor adds extra packets to the link.

We assess the ability of the XGBoost classifier to distinguish actions protected by the defense by training and testing the classifier on the same (but now defended) subsets of the data used to train/test the original attacks in Sect. 5.2. The results of this experiment, shown in Table 2, reveal that padding command traffic alone appears to be largely ineffective for preventing the fingerprinting of position and torque controllers, even when padding all command packets to the same size, imposing a maximum ∼8% accuracy degradation when classifying actions using the velocity controller. According to our feature importance analysis, this may be explained by the fact that, while the size of individual command messages is padded to the same length, the communication volume and amount of command packets exchanged in each direction of the communication still reveal sufficient information for the classifier to identify different actions. Overall, these findings call for a comprehensive exploration of defense mechanisms that can withstand fingerprinting while ensuring the robot's operational effectiveness.

7 Considerations on Generalizability

Robot Characteristics. Different robots may use different joint-level control APIs and control modes, as well as different networking protocols to exchange data. This may result in different command formats and structure of associated messages/network packets. While we find that the Sawyer robotic arm is vulnerable to traffic fingerprinting (and likely, other ROS-based robots as well), we aim to conduct future work that extends our study to a wider set of robots.

Sets of Actions. Additional sets of actions included in our dataset may result in differences on the ability of adversaries to correctly fingerprint robotic actions. However, while admittedly far from comprehensive, our work suggests that even relatively similar actions (e.g., *turn on switch* and *press key*) can be successfully distinguished through traffic analysis.

Network Conditions. Different network conditions may affect the traffic patterns observed on the communication between a robot and its controller. For

instance, degraded conditions may cause packets to be dropped and retransmissions to occur, altering packet delivery rates or causing the exchange of additional messages. We aim to extend our study to more complex networking setups.

8 Conclusions

Our study uncovered significant privacy risks in robot teleoperation, particularly by analyzing encrypted network traffic data of joint-level controller actions exchanged between a collaborative robot arm and the computer that runs the controller. One possible step to address this issue is to integrate traffic scrambling methods in API design for robots, while balancing the integration of these privacy-preserving techniques with the efficiency of robot operations.

Acknowledgments. This work was supported in part by a seed grant jointly awarded by the University of Waterloo's Cybersecurity and Privacy Institute and RoboHub. We thank RoboHub for providing us with access to the Sawyer robotic arm.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Cabrera, M.E., Bhattacharjee, T., Dey, K., Cakmak, M.: An exploration of accessible remote tele-operation for assistive mobile manipulators in the home. In: Proceedings of IEEE RO-MAN (2021)
2. DeMarinis, N., Tellex, S., Kemerlis, V.P., Konidaris, G., Fonseca, R.: Scanning the internet for ROS: a view of security in robotics research. In: Proceedings of ICRA (2019)
3. Gong, J., Wang, T.: Zero-delay lightweight defenses against website fingerprinting. In: Proceedings of USENIX Security (2020)
4. González, C., Solanes, J.E., Munoz, A., Gracia, L., Girbés-Juan, V., Tornero, J.: Advanced teleoperation and control system for industrial robots based on augmented virtuality and haptic feedback. J. Manuf. Syst. **59** (2021)
5. Hayes, J., Danezis, G.: k-fingerprinting: a robust scalable website fingerprinting technique. In: Proceedings of USENIX Security (2016)
6. Mathews, N., Holland, J.K., Oh, S.E., Rahman, M.S., Hopper, N., Wright, M.: SoK: a critical evaluation of efficient website fingerprinting defenses. In: Proceedings of IEEE S&P (2023)
7. Nguyen, T.T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Commun. Surv. Tutor. **10**(4) (2008)
8. Open X-Embodiment Collaboration, et al.: Open X-embodiment: robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864 (2023)
9. Panchenko, A., Lanze, F.: Website fingerprinting at internet scale. In: Proceedings of NDSS (2016)
10. Rahman, M.S., Sirinam, P., Mathews, N., Gangadhara, K.G., Wright, M.: Tik-Tok: the utility of packet timing in website fingerprinting attacks. Proc. Priv. Enhanc. Technol. **2020**(3) (2020)

11. Rodriguez, D., Perez, C., Jagersand, M., Figueroa, P.: A comparison of smartphone interfaces for teleoperation of robot arms. In: Proceedings of CLEI (2017)
12. Romeo, L., Petitti, A., Marani, R., Milella, A.: Internet of robotic things in smart domains: applications and challenges. Sensors **20**(12) (2020)
13. Sankar, G., Djamasbi, S., Li, Z., Xiao, J., Buchler, N.: Systematic literature review on the user evaluation of teleoperation interfaces for professional service robots. In: Proceedings of HCI International (2023)
14. Shah, R., Ahmed, C.M., Nagaraja, S.: Can you still see me?: identifying robot operations over end-to-end encrypted channels. In: Proceedings of ACM WiSec (2022)
15. Shen, M., Ji, K., Gao, Z., Li, Q., Zhu, L., Xu, K.: Subverting website fingerprinting defenses with robust traffic representation. In: Proceedings of USENIX Security (2023)
16. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep fingerprinting: undermining website fingerprinting defenses with deep learning. In: Proceedings of ACM CCS (2018)
17. Tang, C., Barradas, D., Hengartner, U., Hu, Y.: On the feasibility of fingerprinting collaborative robot traffic. arXiv preprint arXiv:2312.06802 (2023)
18. Tang, C., Barradas, D., Hengartner, U., Hu, Y.: Network traces for Sawyer's joint-level controller actions (2025). https://doi.org/10.5281/zenodo.15007304
19. Wang, T., Cai, X., Nithyanand, R., Johnson, R.: Effective attacks and provable defenses for website fingerprinting. In: Proceedings of USENIX Security (2014)
20. Wu, L., Alqasemi, R., Dubey, R.: Development of smartphone-based human-robot interfaces for individuals with disabilities. IEEE Robot. Autom. Lett. **5**(4) (2020)
21. Xue, D., et al.: OpenVPN is open to VPN fingerprinting. In: Proceedings of USENIX Security (2022)