Information Sciences 576 (2021) 329–344

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Novel denial-of-service attacks against cloud-based multi-robot systems

Yuan Xu [a,b,c], Gelei Deng [d], Tianwei Zhang [d,*], Han Qiu [e], Yungang Bao [a,b,c]

[a] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Beijing, China
[b] University of Chinese Academy of Sciences, Beijing, China
[c] Peng Cheng Laboratory, Shenzhen, China
[d] Nanyang Technological University, Singapore, Singapore
[e] Telecom Paris, Palaiseau, France

ARTICLE INFO

Article history:
Received 30 January 2021
Received in revised form 3 June 2021
Accepted 20 June 2021
Available online 24 June 2021

Keywords:
Multi-robot Systems
Cloud-robotics
Denial-of-Service attacks

ABSTRACT

The development of robotics technology is accelerated by the strong support from cloud computing. Massive computation resources and services from the cloud make modern multi-robot systems more efficient and powerful. However, the introduction of cloud servers to multi-robot systems can also incur potential Denial-of-Service (DoS) threats, where an adversary can utilize the shared cloud resources to degrade or bring down the robot systems. In this paper, we conduct a comprehensive study about this security issue. By analyzing different attack vectors in cloud-robotic platforms, we propose three new DoS attacks, which manipulate the network resources, micro-architecture resources, and function parameters respectively. We conduct extensive evaluations and case studies to demonstrate the feasibility and severity of our techniques. We alert the robotics community to these catastrophic attacks on the safety and performance of cloud-robotic systems, and encourage building better defenses for higher reliability, in addition to automation and intelligence.

1. Introduction

The advance in Artificial Intelligence has promoted the rapid development of robotics technology. A variety of robots and autonomous systems are introduced to alter our lifestyle in a revolutionary manner. They can significantly improve our life quality and working efficiency: drones are adopted to deliver packages and supplies; manipulators can be used to assemble products in smart manufacturing, and perform surgery in hospitals; unmanned ground vehicles (UGVs) are widely adopted to perform dangerous jobs at the battlefields, fire and earthquake sites. The size of the global robotics market is anticipated to reach USD $189.36 by the year of 2017, with an annual growth rate of 13.5% [1].

Development and deployment of robot apps require the support from the cloud computing techniques. First, a robot device generally has limited battery capacity and computation capability. It cannot host complicated AI applications and process the tasks promptly. Hence, it is always recommended to offload partial or all robot workloads to the cloud. The powerful cloud resources can guarantee the real-time demands for various tasks, and increase the battery life. Second, cloud providers offer many services, e.g., big data analytics, machine learning framework, database storage, etc. Developers can leverage

* Corresponding author.
E-mail address: tianwei.zhang@ntu.edu.sg (T. Zhang).

https://doi.org/10.1016/j.ins.2021.06.063
0020-0255/

these services to build robot apps with great ease or analyze the mass data [2]. Due to these two reasons, it becomes a popularity to develop, test and run robot apps in cloud-robotic eco-systems. Many cloud-robotic services and platforms have been released to fulfill those functionalities, such as AWS RoboMaker [3], Rapyuta [4], Davinci [5], etc.

Cloud computing also facilitates the implementation of Multi-Robot Systems (MRS). In some complex scenarios, one single robot is not enough to complete the tasks within the demanded deadline. Thus, a couple of robots can be connected as a MRS to work on the tasks together. These robots can be the same type (homogeneous MRS) or different types with different functionalities (heterogeneous MRS). This collaboration can effectively reduce the task completion time. A MRS usually requires a centralized service to coordinate the involved robots. This service collects information from all the robots to make decisions, and sends to each robot the instructions for the next round. A common practice is to deploy such coordination service on the remote cloud. This enables to build a very large-scale MRS even across different locations.

Although cloud computing brings great convenience to the robot systems, it can also open new attack surface for adversaries to affect the entire MRS. There are three reasons that can exacerbate the vulnerabilities of a cloud-based MRS. First, a MRS is a hierarchically distributed system. It consists of multiple robots, with each one running one or more apps. Each robot app consists of multiple processes (a.k.a. nodes), with each one focusing on one specific function. Second, robot nodes are selectively offloaded to the remote cloud server for computation acceleration. They can share the same cloud resources, including operating systems, network bandwidth, and hardware components (memory sub-system, I/O devices, etc.). Nodes can interfere with each other even they are not on the same robot, or directly connected. Third, some robot nodes may not be trusted. On one hand, public robot repositories like Robot Operating System (ROS) [6] allow third-party developers to upload and share their function nodes. These repositories do not perform any security inspection, so an adversary can easily broadcast malicious nodes to other users. On the other hand, a lot of public robot nodes from ROS contain software bugs according to the Robot Vulnerability Database [7]. Most of these bugs are still not fixed, and can be exploited by an adversary to intrude into the vulnerable nodes.

The above facts can create a new attack chain in a cloud-based MRS: an adversary can take control of one robot node, abuse the cloud resource, and interfere with the operations of other nodes, and finally compromise the entire MRS. Such Denial-of-Service (DoS) attacks in the context of cloud-robotics are rarely explored in prior works. We are particularly interested in two question: (1) *how much damage can one malicious node bring to the entire system via its interaction with the cloud?* (2) *What techniques can an adversary leverage to maximize the damage?* Answers to these questions can help us better understand the security of the cloud-based MRS.

In this paper, we present a systematic study towards the above security threats. First, we build an analytic model to disclose the performance characteristics of multi-robot systems and workloads. Based on this model, we identify a critical execution path and a series of function nodes that can determine the performance of the entire system. Then we identify a couple of attack strategies that can affect the execution of the target workloads. Second, we design three novel DoS attacks, where an adversary can use just one malicious node to compromise the entire cloud-robotic platform. In a *network flooding attack*, a malicious node can send a large amount of network packets to flood the network devices, which can extensively interfere with the operations of other critical nodes within the same local network. In a *micro-architecture contention attack*, an adversarial node offloaded to the cloud server can abuse the shared hardware resources to degrade the performance of co-located nodes. In a *parameter manipulation attack*, an adversary can adjust the parameters of the controlled node to change its execution behaviors. This can also increase the workload of the function nodes, and delay the entire system's execution.

We implement the above attack techniques against the ROS system and workloads. Extensive evaluations are conducted from two perspectives. First, we measure the impact of the attacks on the individual critical node (Section 5). Results indicate that these techniques can cause very long latency and high packet drop rates. Besides, they can also significantly increase the processing time of the critical node, and reduce the maximum velocity of the robots. Second, we provide two end-to-end case studies (Section 6). We simulate a multi-robot system for an exploration task using Gazebo. Simulation results show the proposed DoS attacks can incur accidents to threaten the safety of the robot, or increase the total mission completion time by multiple times.

The rest of this paper is organized as follows. Section 2 introduces the background, including standard robot platforms, workloads, cloud-robotic systems, and our threat model. Section 3 presents our performance analysis and modeling of robot workloads, which inspires us to propose three possible attack strategies. We design three attack techniques based on these strategies in Section 4. We perform extensive evaluations in Section 5, followed by two case studies in Section 6. We provide discussions in Section 7, summarize the past works in Section 8 and conclude in Section 9.

2. Background & threat model

2.1. Robot platform

In this section, we describe the structure of robot apps and development cycles. We use the Robot Operating System (ROS) [6] as an example, which has been widely used in the research community and industry, such as Dji Matrice 200 drone [8], PR2 humanoid [9] and ABB manipulator [10].

The ROS platform provides two types of services for robot app developers. First, it offers a set of *robot core libraries*, which serve as the middleware between robot apps and hardware. They are responsible for hardware abstraction, message passing

and also provide device drivers for various sensors and motors. Second, the ROS platform also maintains a large quantity of *robot code repositories* (a.k.a. repos) to support different types of functions, e.g., localization, path planning, path tracking, etc.

Fig. 1 illustrates the lifecycle of robot app development and operation. First, the developer decomposes the design of the target app into some functions. Some of them are core functions (white ellipses) that need to be customized by the developer. Others are non-core functions (black ellipses) which can be directly downloaded from the ROS code repos (①). Then the developer installs the ROS core libraries to integrate these functions as an app workflow (②) and deploys the app to the robot (③). Each function is abstracted as a ROS node and connected as a Directed Acyclic Graph (DFG). They exchange messages through the ROS *Topics*, which are many-to-many named buses that store the robot or environment states. The communication follows the *publish-subscribe* messaging protocol: nodes can subscribe to a topic to obtain relevant data, or publish data to a topic.

The robot app exposes a set of interfaces as ROS *Services* to end users for interaction, e.g., launching tasks, adjusting function parameters. Each service is implemented by the *Remote Procedure Call* (RPC) protocol. Once the robot receives instructions from the user (④), it starts to execute the tasks within the environment (⑤). It will inform the user when all tasks are completed (⑥).

2.2. Robot workloads

A common robot workload can be abstracted as: *navigating the robot to a given destination based on the environmental information captured from various sensors*. So different workloads always follow a standard pipeline, as shown in Fig. 2. This computation pipeline is composed of three major processing stages: PERCEPTION, PLANNING and CONTROL [11]. Each node in Fig. 2 represents a type of functional computation. The solid arrows denote that the connected two nodes communicate in the subscriber/publisher mode, and the dashed arrows represent a client/server paradigm.

Perception: This stage perceives data from the sensors, processes them to extract estimated states of the environment and the robot. It usually consists of two computation nodes: *Localization* is responsible for determining the robot's position; *Costmap Generation* is for modeling the robot's surroundings with costmaps to maintain the navigation information of the robot.

Planning: This stage is responsible for determining the long-range actions of the robot based on high-level goals specified by users. It also consists of two nodes: *Path Planning* (PP) identifies the shortest path from the start position to each destination in a known map; *Exploration* searches for user-defined accessible regions in absence of costmaps.

Control: This stage processes the execution action and forwards these motion commands to the actuators in the control subsystem. The *Path Tracking* node produces velocity commands to follow the planned path given the costmap. The *Velocity Multiplexer* node selects the velocity from multiple commands based on user-defined priorities.

Although these function nodes can be implemented by different libraries, according to whether the map is known, all robot workloads can be classified into two typical categories:

Navigation with a map. *CostmapGen* uses existing map data to create a costmap of its surroundings (①②) and *Localization* estimates the robot's position from the sensor data (③④). Based on the position and the costmap, *Path Planning* generates an efficient collision-free path to the destination (⑤). *Path Tracking* follows this path and outputs the best action from simulating multiple trajectories with different velocities to guarantee feasibility and robustness (⑥), such as obstacle avoidance and oscillation. Considering robot's kinematics and dynamics, some other velocity commands (e.g. safe controller, joystick) are all forwarded to *Velocity Multiplexer* with different priorities, and the final velocity command is sent to the actuators with the highest priority (⑦).

Exploration without a map. To navigate the robot in an unknown area, *Localization* executes the Simultaneous Localization and Mapping (SLAM) algorithm to infer the robot's position in absence of a map. Then, *Exploration* selects a position

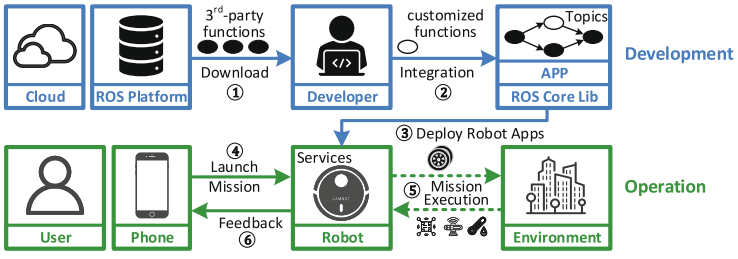


Fig. 1. The lifecycle of robot app development (blue parts) and operation (green parts).

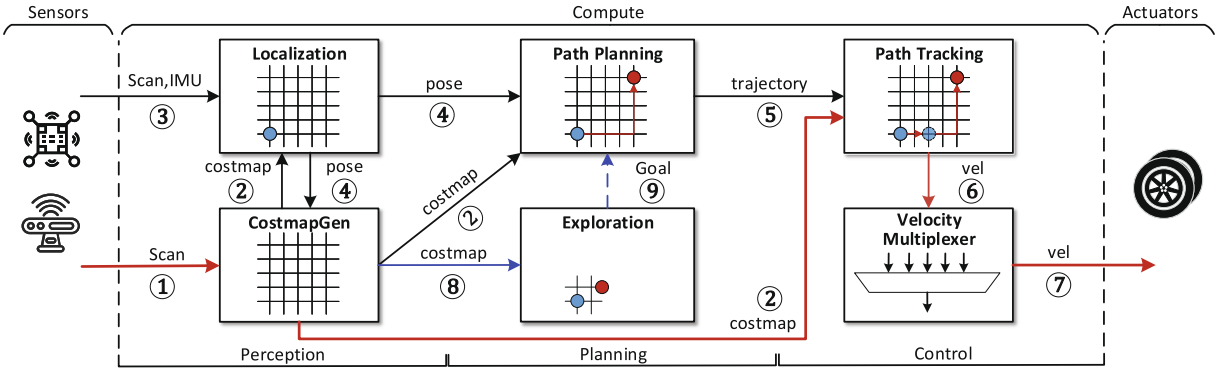


Fig. 2. The standard pipeline for common robot tasks. The workflow with a map is depicted in black lines. The workflow without a map contains all operations in black lines and some new operations depicted in blue lines. The velocity-dependent path is depicted in red lines.

in the frontier of known map as a destination and sends the goal to *Path Planning* (⑧⑨). By repeating this process of costmap update and exploration, the map of the environment will be expanded by publishing the boundary between the "known" and "unknown" regions, until the entire area has been mapped.

2.3. Cloud offloading and resource sharing

The onboard computers of common robots suffer from limited resources and poor computation capability. Hence, it becomes popular to offload the robot workloads to the cloud, and leverage the cloud resources to accelerate the computation and reduce the energy cost [12,13]. Specifically, the developer launches some virtual machines (VMs) on the cloud servers, and deploys the ROS environment inside the VMs. Then the developer migrates some selected function nodes to the VMs. At runtime, each node receives information (e.g., environmental states) from the local robot, and performs the computation. Then it sends the results back to the robot. The local MRS and cloud server is connected by a wireless router.

In a normal cloud-based MRS, one VM is launched corresponded to one robot. Multiple VMs can be located on the same cloud server. A hypervisor is introduced to virtualize and manage the hardware resources such that different VMs can simultaneously share the same micro-architectural components (e.g., CPU pipeline, caches, memory controller, DRAM) while their logical execution and memory are strictly isolated. However, the resource sharing among VMs can lead to severe Denial-of-Service attacks in the cloud environment [14,15].

In the context of cloud-robotic systems, function nodes from the same or different robots can cause severe contention on the network and cloud resources, which can impair the performance of the MRS workloads (Fig. 3). Specifically, (1) different robots communicate with the VMs via the same wireless router. As a result, network packets from different nodes can generate fierce competition on the receiver queue of the router when the communication is intensive. This can delay the nodes' transmission efficiency. (2) Different VMs on the same physical cloud server can interfere with each other via the shared hardware resources, even they are logically isolated by the hypervisor. They contend for the CPU cores, memory systems and I/O devices. This can slow down the nodes' processing speed. More seriously, a malicious node can attempt to misuse

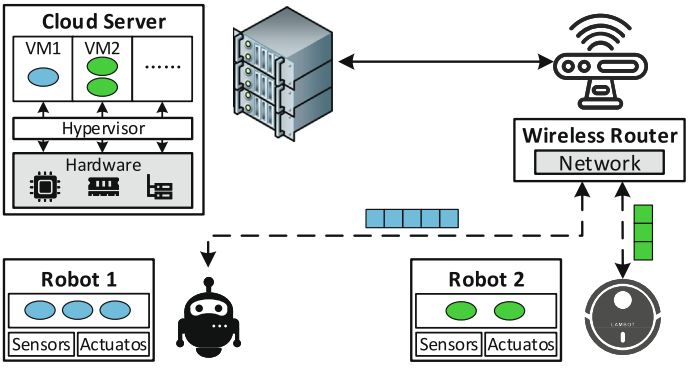


Fig. 3. The offloaded nodes share multiple layers of cloud resources.

the shared resources to exacerbate the severity of resource contention for Denial-of-Service attacks, which we aim to explore in this paper.

2.4. Threat model and assumptions

We consider a cloud-based MRS system, which involves multiple robots, and each robot runs many function nodes. They collaborate to complete a given workload (e.g., exploration, navigation). We assume the underlying OS and ROS core libraries in each robot are trusted: the communication and isolation mechanisms are correctly enforced. How to protect the security of the ROS core libraries [16,17], attack wireless networks [18–20], steal private data [21–23] and defeat the threats from the physical world [24–26] are orthogonal to our work. Some functions nodes are executed on the remote cloud server and the computation results are transmitted back to the robot devices. We assume that all offloaded nodes execute in the same VM or different VMs co-located in the same physical cloud server. This is reasonable since the communication latency among these nodes can be minimized. We assume the cloud computation and its communication with the local robots are also well protected by cryptography.

However, we assume that only one node in one robot is untrusted. It aims to perform Denial-of-Service attacks against the cloud resources, which can further affect the entire MRS workload. Our assumption is based on two observations. First, public robot platforms like ROS are open for everyone to contribute. Developers can upload and share their code repos, without any security inspection. As a result, a malicious developer can insert malware to a repo and publish it to the platforms for users all over the world to download. This threat has been highlighted in the design document of ROS2 Robotic Systems Threat Model [27]: "*third-party components releasing process create additional security threats (third-party component may be compromised during their distribution)*". Second, a lot of public functions in the robot platforms contain software bugs [17,28–30]. According to the Robot Vulnerability Database [7], up to the date of writing, 17 robot vulnerabilities and 834 bugs (e.g., no authentication, uninitialized variables, buffer overflow) have been discovered in the repos of 51 robot components, 37 robots and 34 vendors in the ROS platform. Most of them are still not addressed yet. An adversary can easily exploit those bugs to compromise the function node. Our goal is to identify the possible consequences a malicious node can bring to the entire MRS.

3. Performance analysis of robot workloads

As described in Section 2.3, the adversarial goal is to obtain the control of one malicious node in the target MRS to carry out Denial-of-Service attacks. The malicious node can leverage the contention on the network and cloud resources to cause huge performance loss of the MRS workloads. Thus, it is critical to figure out the intricate relationships between resource contention and MRS performance. In this section, we analyze the MRS workload to identify the key factors that can impact the system performance. We then identify several possible attack strategies in the cloud-robotic scenario.

3.1. MRS performance analysis

We assume that a MRS system has I robots, cooperating to execute one workload. The workload is decomposed into several tasks, and each task is dispatched to a robot. For each robot i, it runs $N + M$ computation nodes to execute the allocated task. N nodes are executed on the embedded computer of the robot and M nodes are offloaded to the cloud servers. We use the local task completion time T_i^L to denote the total time each robot i spends on its task, and the global workload completion time T^G to denote the total time all the robots complete the MRS workload. Since all the robots work in a parallel way, the global workload completion time T^G depends on the longest local task completion time (Eq. (1)):

$$T^G = \max(T_1^L, T_2^L, \ldots, T_I^L) \tag{1}$$

The local mission completion time T^L of each robot consists of two parts: the standby time T_s and moving time T_m (Eq. (2)).

$$T^L = T_s + T_m \tag{2}$$

Standby time T_s measures the time a robot suspends during a task. It exists as the computation capacity cannot meet the task's requirements. When the on-board computation capacity is lower, the robot needs a longer processing time t_p, and has to stay standby for a longer time. Thus, the processing time t_p can be denoted by the sum of three parts, i.e. the processing time of the nodes in the robot t_p^R, the processing time of the nodes in the cloud server t_p^C, and the network latency t_c (Eq. (3)).

$$T_s \sim t_p = t_p^R + t_p^C + t_c \tag{3}$$

Moving time T_m measures the time a robot moves along the path P. It is highly dependent on the maximum velocity v_{max} of a robot: the faster the speed is, the less time it will spend on moving (Eq. (4)). The maximum velocity v_{max} is determined by the obstacle avoidance constraint [31]. As shown in Fig. 4, the robot needs three steps to ensure that the current maximum velocity can prevent it from hitting obstacles. First, it detects obstacles at a distance of d from its sensors. Second, the

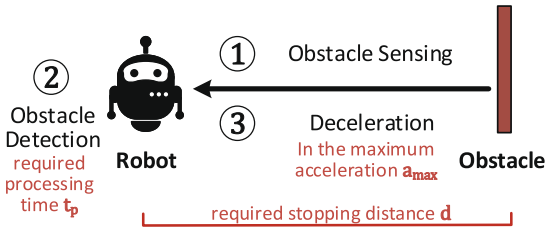


Fig. 4. The process of the obstacle avoidance.

function nodes in the robot spend t_p processing time to recognize the obstacles in front of it. Finally, the robot slows down the speed conforming the maximum acceleration limit a_{max}. If the robot can stop right before it hits the obstacle, the current velocity is safe and available. The maximum of all available velocities is the maximum velocity v_{max}.

$$T_m \sim P/v_{max} = P/[a_{max}(\sqrt{t_p^2 + 2d/a_{max}} - t_p)] \tag{4}$$

From Eqs. (1) to (4), we can observe that the maximum velocity is the key factor to determine the MRS performance. Thus, we introduce the concept of Velocity-Dependent Path (VDP) to help us better understand the computation characteristics of robot tasks. VDP denotes the longest velocity-dependent execution flow path. The total processing time of all the nodes along VDP determines the robot's maximum velocity (Eq. (4)), and thus the local task and global workload completion time. It is the performance bottleneck of a MRS workload.

Recall the standard pipeline of a robot workload in Fig. 2, once receiving the laser data, the *CostmapGen* node detects and marks the obstacles in the costmap. Then *Path Tracking* generates a collision-free path, sends the adjusted velocity to *Velocity Multiplexer* and further forwards to the motors. So the execution path along *CostmapGen*, *Path Tracking* and *Velocity Multiplexer* is VDP in the robot task (red line in Fig. 2).

3.2. Possible attack strategies

From the above analysis, in order to compromise the performance of a cloud-based MRS workload, the adversary can try to delay the execution time of VDP. We identify several possible attack strategies to achieve this goal.

Network Contention. To ensure the performance of the MRS under the wireless environment, communication between the cloud server and local robots usually adopts the UDP protocol. As a result, the adversarial node can flood the network bandwidth and resources with a large quantity of useless UDP messages. This can also affect the latency and packet loss rate, causing safety issues when the robot moves at a fast speed.

Micro-architecture Contention. If the adversarial node is co-located with the critical nodes on the same cloud server, it can generate malicious contention on the micro-architectural units to deprive the resource usage of the critical nodes, and increase their computation time. This is feasible if they share the hardware resources, even they are in different VMs.

Direct Delay. If the adversary is able to directly affect the execution of the computation nodes along VDP via the user interface, he can potentially increase the processing time of the nodes. Then this strategy can decrease the maximum velocity, and increase the local task and global workload completion time.

4. Novel DoS attacks against the cloud-based MRS

Inspired by the three strategies proposed in Section 3.2, we design three new DoS attacks against the cloud-robotic systems. Each of them can incur significant performance degradation, task failure, or safety accidents.

4.1. Network flooding attack

This attack is based on the strategy of network resource contention. It occurs when the malicious node is deployed on a local robot. Then the adversary can configure this node to send a large amount of network packets to the nodes on other robots or the cloud. The cloud server has wired connection with the router, which has larger bandwidth than the wireless network. Thus, the adversary's behavior will first cause traffic congestion in the wireless router.

As shown in Fig. 5(a), function nodes exchange messages (gray squares) based on the topics. Each topic is a named bus and strongly typed by the message type. If a node is interested to receive this type of data, it can subscribe to this topic and becomes a *subscriber*. On the contrary, a node can also publish data to a relevant topic and becomes a *publisher*. Both the publisher and subscriber are unaware of the nodes they are communicating with. Once receiving a message from the topic, the subscriber triggers its callback function and uses the received data for computation, e.g., path planning and map

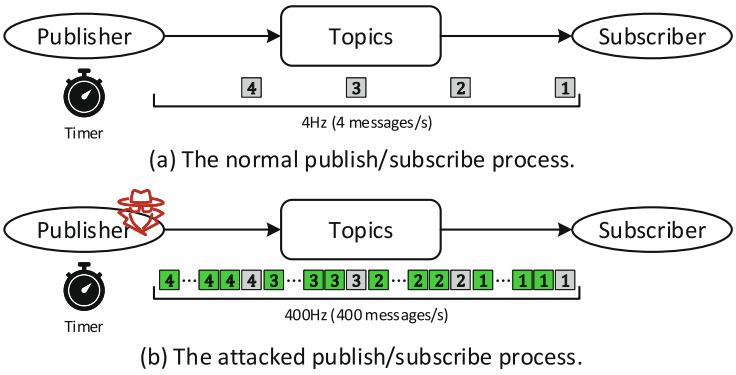


Fig. 5. The process of network flooding attack.

generation. To reduce the computation overhead, the publisher commonly implements a timer to limit the publishing rate. For example, a publishing rate of 4 Hz means the publisher can send 4 messages per second.

The malicious node can become a publisher, and alter its timer to flood messages to the MRS. Fig. 5(b) shows the detailed process of such network flooding attack. Specifically, the malicious node sets its timer from 4 Hz to 400 Hz, which can increase the network bandwidth by 100 times. Every 0.25s, it generates 99 dummy messages (green squares), which have the same content as the normal message. These dummy messages will not cause any function errors to the robot workload. However, they can quickly fill up the receiving queue of the router, resulting in a slowdown of scheduling normal messages, or packet loss for the critical nodes. This can terribly affect the safety state of the robot and cause potential accidents.

4.2. Micro-architecture contention attack

This attack is based on the strategy of micro-architecture contention. It happens when the malicious node is deployed in a VM on the cloud server, sharing the same hardware resources with other nodes. The malicious node can try to consume more resources and affect the critical VDP. For instance, it can use the same strategy as the network flooding attack, by publishing a lot of dummy messages to certain topics. Then nodes subscribed to these topics will keep receiving those messages, and waste more computation resources to process them. Note that this strategy has very high effectiveness: one malicious publisher node can lead to multiple subscriber nodes to do useless computations, which can easily deplete the valuable resources and delay the critical operations of the nodes along VDP (e.g., *Path Tracking*).

Fig. 6 describes the detailed procedure of message processing in the cloud server. From the publisher, the sent messages are first serialized into the string type and copied to newly initialized memory regions in the user space. Then the publisher node issues the *sendto* system call to copy the serialized data from the user buffer to the kernel buffer. Since the destination address of these messages is the machine itself, the data are copied back to the user space again and deserialized into their original message type. This deserialization process also needs one copy action in the user space. Thus, a full closed loop of a publish/subscribe process costs four memory copies. The malicious node can leverage these frequent copy operations to increase the micro-architecture contention and increase the workload completion time.

4.3. Parameter manipulation attack

This attack is based on the strategy of direct delay. Each node exposes services for end users to adjust function parameters. Unfortunately, some parameters can be directly manipulated by the adversary to increase the computation cost of one function node. If the node is along VDP, then misconfiguration of such critical node can significantly affect the performance of the entire MRS. Below we give an example to maliciously configure the *Path Tracking* node.

As discussed in Section 3.1, acceleration of the computational-intensive nodes along VDP is crucial to reduce the task completion time. To navigate along a planned path in an obstacle-filled environment, the robot needs a costmap to represent the knowledge of geometric world and a path tracker to compute the optimal velocity. A costmap uses the static map, laser data and robot's footprint to generate multiple layers (e.g., static map layer, obstacle map layer, inflation layer) to store and update information about the obstacles. Based on the costmap and the global path generated from *Path Planning*, *Path Tracking* simulates multiple possible trajectories based on the current velocity. For each possible velocity, it performs forward simulation to generate M trajectories. To find the best path, it scores each trajectory using a cost function that incorporates many characteristics, including proximity to the goal, to the global path, to obstacles and oscillation. After discarding all illegal trajectories (e.g. colliding with obstacles), the trajectory with the highest score will be selected and the corresponding velocity is sent to *Velocity Multiplexer*. Thus, we conclude that the high computation burden for *Path Tracking* is derived from two

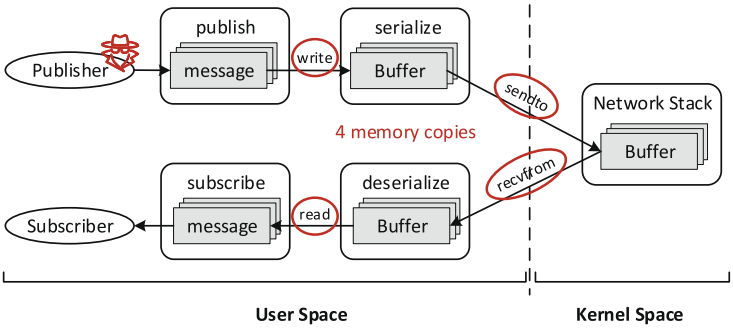


Fig. 6. The process of micro-architecture contention attack.

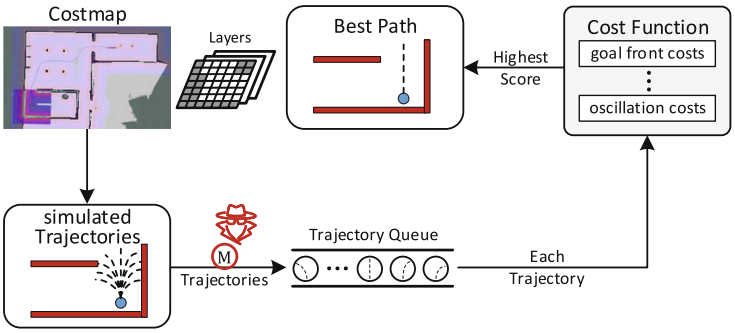


Fig. 7. Parameter manipulation attack in the *Path Tracking* node.

factors: the sequentially performed duplicated scoring work and the number of trajectories. The adversary can increase the number of trajectories M to cause a DoS attack on the *Path Tracking* node, and then the entire system (Fig. 7).

5. Evaluation

5.1. Experiment setup

We select two types of robots, and one remote server to set up our cloud-based MRS. Table 1 details the specifications of hardware resources. Specifically, we use the NVIDIA Jetson TX2 (NVIDIA Denver CPU@2.3 GHz with 8 GB of RAM) module and a laptop (Intel i5-8250U@1.6 GHz with 8 GB RAM) to simulate Robot 1 and Robot 2, respectively. These modules are widely used in commercial robots, such as Jackal [32], Jet [33] and Turtlebot2 [34]. Both of the two robots are connected to a wireless router (TP-LINK WDR8600) in our lab with a passive 5 GHz band wireless network. We set up a local server (Intel i7-7700 K CPU@4.2 GHz with 16 GB of RAM) and connect it to the router with a wired link. We configure this server to represent the configurations of different cloud servers general purpose instance with a frequency of 2.5 GHz, and high-frequency compute instance with a frequency of 3.1 GHz). We use VirtualBox 6.0 as the hypervisor and launch several VMs on the server to execute migrated computations. All robots and VMs run ubuntu 18.04 and ROS melodic. We choose two types of messages with different sizes in the experiment: an image message of 230 KB and an Inertial Measurement Unit (IMU) message of 0.33 KB. Thus, the traffic rates in different tests are the product of attack Hz and the size of the malicious image and IMU message.

Table 1
Cloud-based MRS specifications.

	Robot 1	Robot 2	Cloud Server
Module	NVIDIA Jetson TX2	Intel i5-8250U	Intel i7-7700 K
Frequency	2.3 GHz	1.6 GHz	4.2 GHz
Cores	2	4	4
Memory	8 GB	8 GB	16 GB
Example	Jackal [32], Jet [33] F1Epoch RACECAR [35]	ROCH [36], Spark [37]Turtlebot2 [34]	General-purpose instance (2.5 GHz) High-frequency compute instance (3.1 GHz)

5.2. Network flooding attack

We first measure the attack effectiveness caused by network contention. We assume a malicious publisher node exists in Robot 1, which sends a large amount of dummy image messages to a subscriber node in the remote server. At the same time, we deploy a victim node in Robot 2, which sends normal messages to the server, and receives the same messages from it. We measure the Round-Trip Time (RTT) of each message, and count the loss ratio of messages dropped during the communication.

Fig. 8 shows the distribution of RTT and message loss rate of the victim robot under various message flooding frequencies by the malicious publisher node. From Fig. 8(a), we observe that the average RTT of the received messages without the attack is 2.24 ms. When the adversary launches the network flooding attack, the RTT is significantly increased with the flooding rate. With a rate of 25 Hz, the average RTT of the victim message becomes 7.32 ms, and the maximum RTT is 193 ms, which indicates severe service degradation. The message loss of the victim robot is even worse since the remote communication is based on the UPD protocol (Fig. 8(b)): when the flooding rate reaches 20 Hz, about 35% messages are dropped due to the network contention in the wireless router. Such high drop rate can bring severe consequences to the safety of the MRS, which will be demonstrated in Section 6.2.

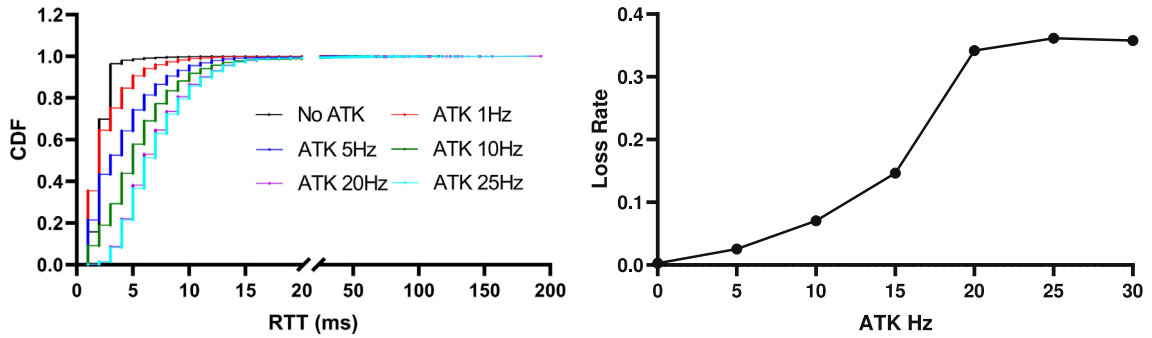
5.3. Micro-architecture contention attack

We launch a malicious publisher node in the remote server, which floods dummy image and IMU messages to normal subscriber nodes in the same machine, respectively.

We first demonstrate the malicious publisher node can remarkably increase the CPU utilization of the subscriber nodes, making them deprive of the CPU resources from other normal nodes. Fig. 9 shows the CPU utilization of a subscriber node under different flooding rates of these two messages. We use red, blue and green lines to denote the results with the CPU frequency of 2.5 GHz, 3.1 GHz and 4.2 GHz, separately. We can draw two observations from this figure. First, to occupy full CPU load, the malicious node needs to publish dummy messages at a higher rate when the cloud server adopts a higher frequency, which can process the operations at a faster speed. Second, since the IMU messages have a smaller size, the malicious publisher node is able to send the messages at a higher frequency. Consequently, the CPU utilization of the subscriber node is higher under IMU flooding (Fig. 9(a)) than under image flooding (Fig. 9(b)).

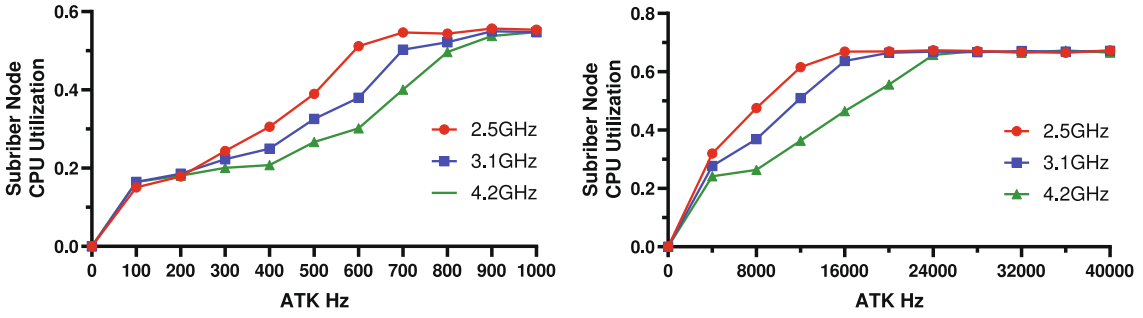
To dive deep into the CPU usage under these circumstances, we measure the CPU utilization of main functions, as shown in Fig. 10. We observe that the causes of heavy computation cost flooded by two types of messages are totally different. For the big-size image message, the cost originates from the four memory copy operations as discussed in Section 4.1. As shown in Fig. 10(a), the *memcpy* and *memset* functions are triggered by the serialization and deserialization in the subscriber process. The copy_to_user and *copy_from_user* are systemcall functions that copy data between the user and kernel spaces. However, when the message size is small (IMU), the computation overhead becomes the function call cost (Fig. 10(b)): the top-three functions that consume the most CPU resources are the built-in Python APIs to interpret bytecode, search attributes from the dictionary and execute calls. Thus, we conclude that the high-frequency function calls are the computation bottleneck for the flooding messages with small sizes. Since small-size messages are more effective in increasing the CPU utilization of the subscriber node, we will adopt this message type for the following experiments.

Next, we show such CPU contention can impact the processing speed and maximum velocity of the robot. We launch the malicious publisher node, N subscriber nodes, and the critical *Path Tracking* node on the same CPU core. The publisher node keeps flooding the IMU messages to the topic, and received by those subscriber nodes. We set N=5, 10, 15 and 20. Fig. 11(a) shows the processing time of *Path Tracking* versus various flooding rate. We can observe the processing time is increased linearly with the flooding rate until it becomes saturate, which denotes the case that the CPU utilization of this core reaches



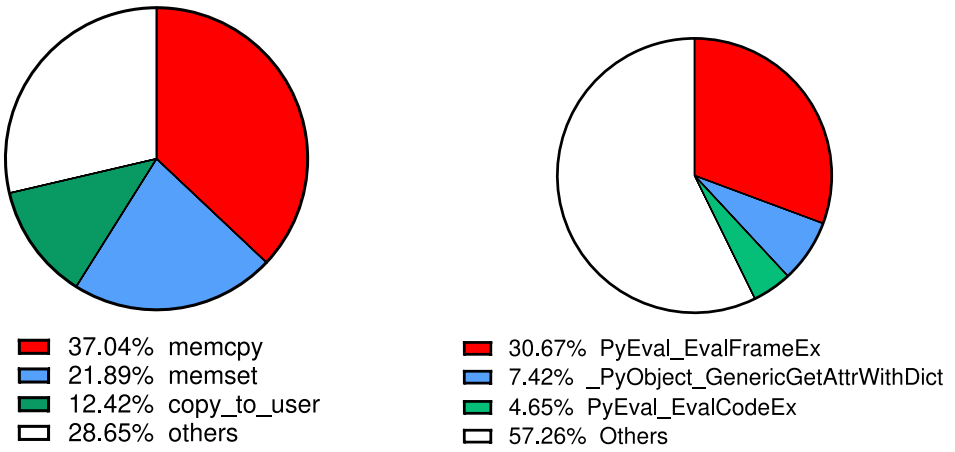
(a) RTT of the victim robot. (b) Message loss rate of the victim robot.

Fig. 8. Network flooding attack results.



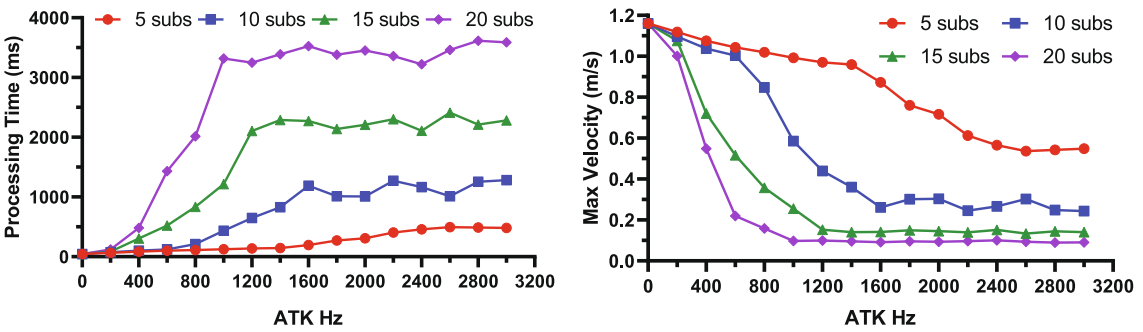
(a) Flooding image messages. (b) Flooding IMU messages.

Fig. 9. CPU utilization of the subscriber node.



(a) Flooding image messages. (b) Flooding IMU messages.

Fig. 10. CPU utilization of main functions in a subscriber node.



(a) Processing time of *Path Tracking*. (b) Maximum velocity of the robots.

Fig. 11. Impact of micro-architecture contention on the workload.

100%. Besides, a larger N can also leads to a longer processing time, due to the fair scheduling of the OS scheduler. When $N = 20$, the processing time of *Path Tracking* is more than 3700 ms, which indicates a severe DoS threat.

We also calculate the maximum velocity according to Eq. (4), as shown in Fig. 11(b). The maximum velocity is calculated from the processing time t_p, the maximum acceleration limit a_max and required stopping distance d. We set t_p as the velocity-dependent path time, a_max and d following the mechanical specifications of one common commercial mobile robot (turtlebot3 [38]). We can see for the worst case, the maximum velocity of the robot can be below 0.1 ms. This will remarkably increase the global workload completion time, which will be shown in Section 6.1.

5.4. Parameter manipulation attack

Finally, we evaluate the parameter manipulation attack, where the adversary tampers with the parameters of *Path Tracking* to affect the execution of VDP. To increase the processing time of the *Path Tracking* node, the adversary can add more simulated trajectories by setting bigger values for two parameters: *vx_samples* and *vth_samples*. The *vx_samples* parameter specifies the number of velocity samples in the x dimension (i.e., forward direction) while the *vth_samples* function specifies the number of velocity samples in the θ dimension. For each sample of the linear velocity in *vx_samples* and angular velocity in *vth_samples*, the *Path Tracking* node simulates a specific trajectory for scoring. In total, it needs to generate *vx_samples* $\times$ *vth_samples* trajectories.

Fig. 12(a) shows the processing time of the *Path Tracking* node when we increase *vx_samples* and *vth_samples* simultaneously from 30 to 300. We can clearly observe the processing time is increased by 103.5×, 97.6×, 110.6× under the 2.5 GHz, 3.1 GHz and 4.2 GHz CPU frequencies, respectively. We also calculate the maximum velocity according to Eq. (4), as shown in Fig. 12(b). We can observe the maximum velocity decreases from 1.12 m/s to 0.04 m/s due to the increase of the two parameters. This indicates a severe service availability threat, as demonstrated in Section 6.1.

6. Case studies

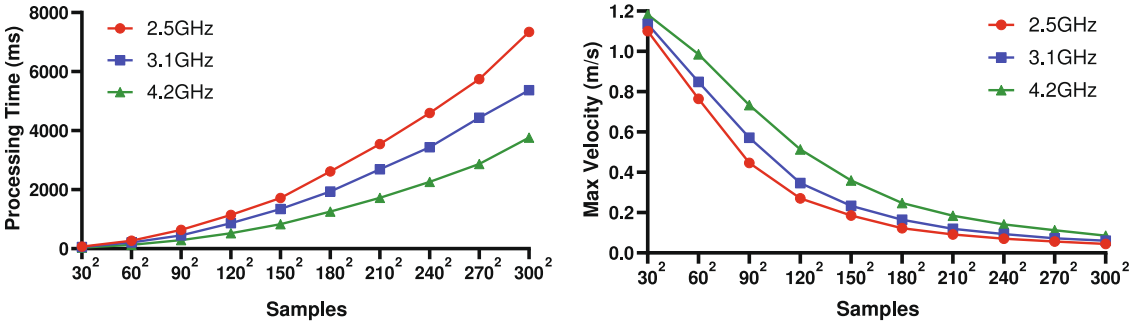
In Section 5, we show our attack techniques are able to increase the node's message drop rate, and reduce the maximum velocity. In this section, we provide two case studies to show these attacks can incur severe consequences to the MRS.

6.1. Case 1: Increasing workload completion time

The micro-architecture and parameter manipulation attacks can significantly increase the processing time of the *Path Tracking* node, and reduce the maximum velocity below 0.05 m/s. We show that such a small velocity can affect the global workload completion time.

We use the ROS platform and Gazebo simulator to implement an MRS, consisting of three robots. They collaborate on a map exploration task. A coordination service is deployed in the server, which assigns the frontiers of unknown area to robots for exploration based on the calculated information gain. Each robot keeps exploring the assigned area, updating the global map and its position. Fig. 13 shows the map to be explored. The default maximum velocity of a robot in Gazebo is 0.55 m/s. We assume all three robots (red square) are affected by the DoS attack, and their maximum velocities are decreased at the same time.

Fig. 14 shows the relationship between the workload completion time and the maximum velocities of the three affected robots. We can observe that when the velocities of the affected robots are higher than 0.25 m/s, the workload completion time is hardly altered. However, when the velocities are reduced below 0.25 m/s, which can be achieved by our DoS attacks,



(a) The increase of the processing time. (b) The reduction of the maximum velocity.

Fig. 12. Attack results caused by the changes of *vx_samples* and *vth_samples*.

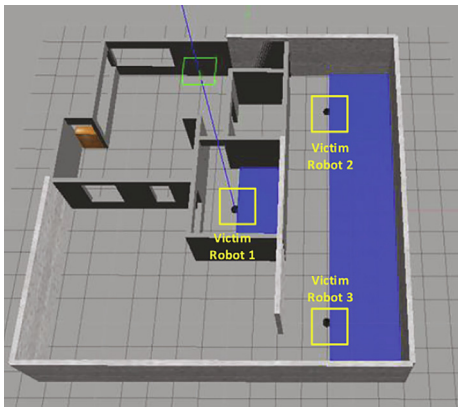


Fig. 13. The simulated scenario of MRS exploration workload in Gazebo.

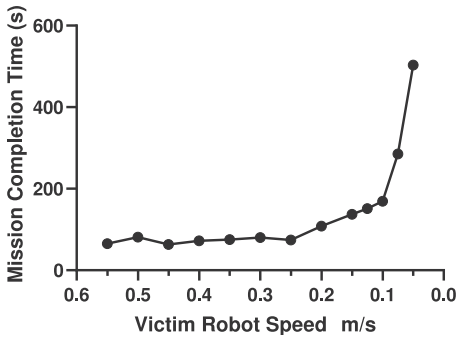


Fig. 14. The workload completion time of the MRS exploration workload.

the corresponding workload completion time is sharply increased. This validates that our attack techniques can severely affect the performance of the entire exploration task.

6.2. Case 2: Incurring accidents

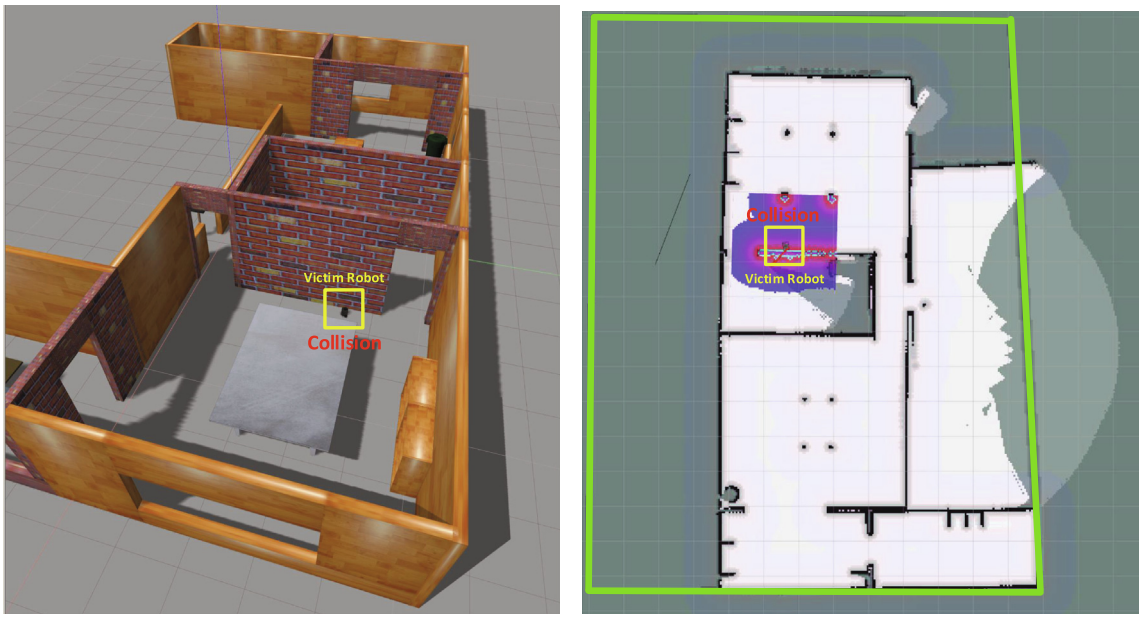
The network flooding attack can increase the communication latency and packet drop rates. This can incur severe safety problems in MRS. We use this case study to demonstrate such consequence.

As Fig. 15 shows, we use the Gazebo and Rviz simulator to implement and visualize a cloud-based exploration task. Specifically, a robot is dispatched to explore an unknown area (green square) in the map using the same offloading configuration as the previous case. When it has explored almost 90% of the map and plans to enter the only unknown zone, an adversary launches a network flooding attack. As discussed in Section 5.2, about 35% of the robot's messages are dropped due to the network contention in the wireless router. Such a high drop rate causes a crash when the robot turns left to enter the room. This is because the dropped packages make the robot move at an unsafe maximum velocity. As shown in Fig. 4 and Eq. (4), the maximum velocity is determined by the current stopping distance d. When the current generated maximum velocity message is dropped, the robot moves at the previous maximum velocity. However, the stopping distance is shortened when it moves during the attack period. As a result, the previous maximum velocity does not give enough time for the robot to reduce the speed to 0 m/s before it hits the obstacle.

7. Discussion and future work

7.1. Possible defense

Network Flooding Defense. One traditional method to mitigate the UDP flood DoS attack is to limit the response rate of ICMP packets. However, this method is not suitable for MRS apps since it would also filter out legitimate packets. Another defense method is fingerprint learning [39], which checks whether the same offset locations of various UDP packets contain the same content. Unfortunately, most of the ROS packages contain the unique package ID and time information at the application layer. Although the data among malicious packets and normal packets are the same, the contents of the UDP packets



(a) The simulated scenario in gazebo. (b) The visualization of the exploration workload in rviz.

Fig. 15. The exploration workload under network flooding attack.

are different. This makes the fingerprint learning approach ineffective. If multiple wireless routers are allowed, the robot can use some AP selection methods to switch the wireless router when it detects the package loss increases. However, the adversary can also launch the network flooding attack to all the wireless routers that it can connect to. The best defense solution we propose is to implement a monitor process to record the original publishing rate of each publisher and detect whether this value is changed. If the rate suddenly increases, it resets the rate to the original value.

Micro-architecture Contention Defense. The most effective strategy to alleviate micro-architecture contention is to partition the hardware resources physically or temporally for different entities. For instance, different approaches have been designed to partition the shared CPU caches [40,41] and memory [42]. These solutions can be applied to our scenario, and isolate the cloud resources for different ROS nodes. An alternative solution is to monitor the runtime behaviors and resource consumption of each node, and identify any anomalous activities caused by certain nodes. For instance, Hardware Performance Counters have been widely adopted by researchers to reflect the resource fairness and DoS attack detection in multi-tenant clouds [14,43,44]. These solutions are expected to defeat our proposed attack as well, which will be evaluated in our future work.

Parameter Manipulation Defense. The parameter manipulation attack is robot-specific, and there are very few works to discuss the defense solutions. One possible method is to set a threshold to limit the legal choice of performance-related parameters. But this may make the robot fail to work in the complex environment because it needs more computations to deal with. The best defense solution is to design an adaptive algorithm to automatically adjust the parameter according to the complexity and uncertainties of the environment. One typical example is the Adaptive Monte Carlo Localization (AMCL) algorithm [45]. This laser-based algorithm uses an adaptive particle filter to track the pose of a robot against a known map and automatically adjust the number of particles at runtime, preventing the adversary from manipulating the parameters.

7.2. Attack effectiveness on ROS2

This paper mainly focuses on ROS with the melodic version, since ROS is still the main choice for the commercial products of many companies (e.g., Dji Matrice 200 drone [8], PR2 humanoid [9] and ABB manipulator [10]). Recently ROS2 [46] was released, which provides more security measures. It implements a DDS/RTPS communication system to replace the simple TCP/UDP-based pub/sub communication in ROS. Hence, ROS2 offers a rich variety of Quality of Service (QoS) policies that

allow users to tune communications between nodes and possibly mitigate the above three types of DoS attacks. As future work, we will extend our evaluation to ROS2 or a hybrid design of ROS/ROS2.

7.3. Attack effectiveness with other technologies

As discussed in Section 5.3, the heavy computation cost in the micro-architecture contention attack depends on the size of flooding message. For big-sized messages, one possible defense method is to use the kernel-bypassing technique such as DPDK [47] and netmap [48]. However, this method can only reduce part of the contention because more than half of the cost is derived from *memcpy* and *memset* functions, triggered by the serialization and deserialization in the subscriber process. These two functions are not relevant to the kernel-bypassing technique.

8. Related works

DoS attacks in the cloud. DoS attacks in the cloud scenario has been extensively studied. Some works [49–51] proposed network-based attacks, which can deplete the network bandwidth or network device resources. I/O-based attacks were evaluated in [52–54] to degrade the performance of virtual machines. Some attacks [55] compromised the hypervisor scheduler to steal the CPU usage of victim VMs. [56] designed the resource-freeing attack, where the adversary can steal one type of resource from the co-located victim VM by increasing this VM's usage of other types of resources. The performance degradation due to memory resource contention was explored in [57–59], enabling an adversary to perform cross-VM DoS attacks on the memory sub-system [14,15].

DoS attacks in robotic systems. A quantity of works also focus on the DoS attacks in various robotic devices and systems. For instance, the impacts of DoS attacks on drones were evaluated in [60,61]. How to protect the autonomous vehicles from DoS attacks was discussed in [62,63]. Similarly, some works also explored to enhance the resilience of multi-robot systems against DoS attacks [64–66]. A few studies designed solutions to build secure communication protocols in ROS to mitigate DoS threats [67,68].

However, there are very few works considering the DoS threats in the cloud-robotic context. Our work bridges this gap by systematically analyzing the characteristics of cloud-robotic systems and workloads, and proposing novel DoS attack techniques. Considering the cloud-based MRS is a popular trend to meet the increased demands of high automation, intelligence and reliability, our study can shed light on the security protection of cloud and robot systems.

9. Conclusion

In this paper, we present the first study towards the DoS threat of cloud-based multi-robot systems. We propose three novel attack techniques to target different layers of the MRS. They leverage the shared resources among different function nodes to incur malicious contention and resource depletion. We perform comprehensive evaluations and case studies to show the attacks can cause severe safety and performance issues. We encourage researchers and practitioners from the robotics community to seriously consider this threat when designing new multi-robot systems. As future work, we will explore possible defense solutions to enhance the resilience of cloud-robotic systems against DoS attacks.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported in part by Key-Area Research and Development Program of Guangdong Province (NO.2020B010-164003), the National Natural Science Foundation of China (Grant No. 62090020), Youth Innovation Promotion Association of Chinese Academy of Sciences (2013073), and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030200), Singapore MoE AcRF Tier 1 RG108/19 (S) and NTU-Desay Research Program 2018-0980.

References

[1] Robotics technology market, https://www.alliedmarketresearch.com/robotics-technology-market (2020).
[2] M. Qiu, Z. Ming, J. Wang, L.T. Yang, Y. Xiang, Enabling cloud computing in emergency management systems, IEEE Cloud Comput. 1 (4) (2014) 60–67.
[3] Amazon aws robomaker, https://aws.amazon.com/robomaker/ (2019).
[4] D. Hunziker, M. Gajamohan, M. Waibel, R. D'Andrea, Rapyuta: The roboearth cloud engine, in: International Conference on Robotics and Automation (ICRA), 2013.
[5] B.L. Rajesh Arumugam, Vikas Reddy Enti, X. Wu, K. Baskaran, F.K. Foong, A.S. Kumar, D.M. Kang, W.K. Goh, Davinci: A cloud computing framework for service robots, in: Symposium on Cloud Computing (SoCC), 2013.
[6] Open source robot operating system, http://www.ros.org/ (2019). .

[7] Robot vulnerability database (rvd), https://github.com/aliasrobotics/RVD/ (2020).
[8] Dji onboard sdk, https://developer.dji.com/onboard-sdk/ (2020).
[9] Ros pr2 package, http://wiki.ros.org/Robots/PR2/ (2020).
[10] Ros abb package, http://wiki.ros.org/abb/ (2020).
[11] Siciliano, Bruno, O. Khatib, Springer handbook of robotics, Springer, Secaucus, NJ, USA: Sprinter-Verlag New York Inc, 2016.
[12] B. Kehoe, S. Patil, P. Abbeel, K. Goldberg, A survey of research on cloud robotics and automation, IEEE Trans. Autom. Sci. Eng. (T-ASE) 12 (2) (2015) 398–409.
[13] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, A.V. Vasilakos, Cloud robotics: Current status and open issues, IEEE Access 4 (2016) 2797–2807.
[14] T. Zhang, Y. Zhang, R.B. Lee, Dos attacks on your memory in cloud, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 253–265.
[15] T. Zhang, R.B. Lee, Host-based dos attacks and defense in the cloud, in: Proceedings of the Hardware and Architectural Support for Security and Privacy, 2017, pp. 1–8.
[16] B. Dieber, S. Kacianka, S. Rass, P. Schartner, Application-level security for ros-based applications, in: International Conference on Intelligent RObots and Systems (IROS), 2016.
[17] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, P. Schartner, Security for the robot operating system, IEEE Trans. Rob. Autonom. Syst. 98 (2017) 192–203.
[18] H. Sathaye, D. Schepers, A. Ranganathan, G. Noubir, Wireless attacks on aircraft instrument landing systems, in: USENIX Security Symposium (USENIX Security 19), 2019.
[19] R. Baker, I. Martinovic, Losing the car keys: Wireless phy-layer insecurity in ev charging, in: USENIX Security Symposium (USENIX Security 19), 2019.
[20] Z. Zhang, J. Wu, J. Deng, M. Qiu, Jamming ack attack to wireless networks and a mitigation approach, in: Global Communications Conference (GLOBECOM), 2008.
[21] D. Frassinelli, S. Park, S. Nürnberger, I know where you parked last summer: Automated reverse engineering and privacy analysis of modern cars, in: IEEE Symposium on Security and Privacy (S&P), 2020.
[22] S. Birnbach, R. Baker, I. Martinovic, Wi-fly?: Detecting privacy invasion attacks by consumer drones, in: Network and Distributed System Security Symposium (NDSS), 2017.
[23] W. Dai, M. Qiu, L. Qiu, L. Chen, A. Wu, Who moved my data? privacy protection in smartphones, IEEE Commun. Mag. 55 (1) (2017) 20–25.
[24] N.O. Tippenhauer, C. Pöpper, K.B. Rasmussen, S. Capkun, On the requirements for successful gps spoofing attacks, in: ACM Conference on Computer and Communications Security (CCS), 2011.
[25] H. Shin, D. Kim, Y. Kwon, Y. Kim, Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications, in: International Workshop on Cryptographic Hardware and Embedded Systems (CHES), 2017.
[26] Y. Son, H. Shin, D. Kim, Y.-S. Park, J. Noh, K. Choi, J. Choi, Y. Kim, Rocking drones with intentional sound noise on gyroscopic sensors, in: USENIX Security Symposium (USENIX Security 15), 2015.
[27] Ros 2 robotic systems threat model, https://design.ros2.org/articles/ros2_threat_model.html (2020).
[28] S. Rivera, A.K. Iannillo, R. State, Ros-immunity: Integrated approach for the security of ros-enabled robotic systems.
[29] R. White, G. Caiazza, C. Jiang, X. Ou, Z. Yang, A. Cortesi, H.I. Christensen, Network reconnaissance and vulnerability excavation of secure dds systems, in: EuroS&P Workshops, 2019.
[30] J.R. Mcclean, C. Farrar, A preliminary cyber-physical security assessment of the robot operating system (ros), in: Proceedings of SPIE, 2013.
[31] S. Liu, M. Watterson, S. Tang, V. Kumar, High speed navigation for quadrotors with limited onboard sensing, in: International Conference on Robotics and Automation (ICRA), 2016.
[32] Jackal, unmanned ground vehicle, https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (2020).
[33] Jackal – unmanned ground vehicle, https://developer.nvidia.com/jet-robot/ (2020).
[34] Turtlebot2, https://www.turtlebot.com/turtlebot2/ (2020).
[35] The f1epoch racecar robot, https://github.com/NVIDIA-AI-IOT/Formula1Epoch/ (2020).
[36] Roch – wheeled mobile robot, agv robot, http://www.soyrobotics.com/product/product_162_1.html (2020).
[37] Spark, http://wiki.ros.org/Robots/Spark (2020).
[38] Turtlebot3, https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (2020).
[39] Huawei antiddos v500r001 maintenance guide, https://support.huawei.com/enterprise/en/doc/EDOC1000128684/c62f79d0/udp-attack-defense-policy (2020).
[40] Improving real-time performance by utilizing cache allocation technology, http://www.intel.com/content/www/us/en/communications/.
[41] T. Kim, M. Peinado, G. Mainar-Ruiz, {*STEALTHMEM*}: System-level protection against cache-based side channel attacks in the cloud, in: 21st {USENIX}) Security Symposium {*USENIX*} Security 12), 2012, pp. 189–204.
[42] T.M.O. Mutlu, Memory performance attacks: Denial of memory service in multi-core systems, in: USENIX security, 2007.
[43] C. Delimitrou, C. Kozyrakis, Paragon: Qos-aware scheduling for heterogeneous datacenters, ACM SIGPLAN Notices 48 (4) (2013) 77–88.
[44] Y. Zhang, M.A. Laurenzano, J. Mars, L. Tang, Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers, in: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE, 2014, pp. 406–418.
[45] D. Fox, Kld-sampling: Adaptive particle filters, in: Conference on Neural Information Processing Systems (NeurIPS), 2001.
[46] Ros 2 documentation, https://docs.ros.org/en/foxy/index.html (2020).
[47] The dpdk project, https://support.huawei.com/enterprise/en/doc/EDOC1000128684/c62f79d0/udp-attack-defense-policy (2020).
[48] The netmap project, http://info.iet.unipi.it/luigi/netmap/ (2020).
[49] H. Liu, A new form of dos attack in a cloud and its avoidance mechanism, in: ACM Workshop on Cloud Computing Security, 2010.
[50] H.S. Bedi, S. Shiva, Securing cloud infrastructure against co-resident DoS attacks using game theoretic defense mechanisms, in: Intl. Conf. on Advances in Computing, Communications and Informatics, 2012.
[51] Z. He, T. Zhang, R.B. Lee, Machine learning based ddos attack detection from source side in cloud, in: 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud) IEEE, 2017, pp. 114–120.
[52] Q. Huang, P.P. Lee, An experimental study of cascading performance interference in a virtualized environment, SIGMETRICS Perf. Eval. Rev.
[53] Z. Yang, H. Fang, Y. Wu, C. Li, B. Zhao, H. Huang, Understanding the effects of hypervisor i/o scheduling for virtual machine performance interference, in: IEEE Intl. Conf. on Cloud Computing Technology and Science, 2012.
[54] R. Chiang, S. Rajasekaran, N. Zhang, H. Huang, Swiper: Exploiting virtual machine vulnerability in third-party clouds with competition for i/o resources, IEEE Trans. Parall. Distrib. Syst.
[55] F. Zhou, M. Goel, P. Desnoyers, R. Sundaram, Scheduler vulnerabilities and coordinated attacks in cloud computing, in: IEEE Intl. Symp. on Network Computing and Applications, 2011.
[56] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, M.M. Swift, Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense), in: ACM Conf. on Computer and Communications Security, 2012.
[57] D. Grunwald, S. Ghiasi, Microarchitectural denial of service: Insuring microarchitectural fairness, in: ACM/IEEE Intl. Symp. on Microarchitecture, 2002.
[58] D.H. Woo, H.-H.S. Lee, Analyzing performance vulnerability due to resource denial-of-service attack on chip multiprocessors, in: Workshop on Chip Multiprocessor Memory Systems and Interconnects, 2007.
[59] T. Moscibroda, O. Mutlu, Memory performance attacks: Denial of memory service in multi-core systems, in: USENIX Security Symp., 2007.
[60] G. Vasconcelos, G. Carrijo, R. Miani, J. Souza, V. Guizilini, The impact of dos attacks on the ar. drone 2.0, in: 2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), IEEE, 2016, pp. 127–132.

[61] G. Vasconcelos, R.S. Miani, V.C. Guizilini, J.R. Souza, Evaluation of dos attacks on commercial wi-fi-based uavs, Int. J. Commun. Networks Inf. Secur. 11 (1) (2019) 212–223.
[62] K.M.A. Alheeti, A. Gruebler, K.D. McDonald-Maier, A. Fernando, Prediction of dos attacks in external communication for self-driving vehicles using a fuzzy petri net model, in: 2016 IEEE International Conference on Consumer Electronics (ICCE) IEEE, 2016, pp. 502–503.
[63] D. Zhang, Y.-P. Shen, S.-Q. Zhou, X.-W. Dong, L. Yu, Distributed secure platoon control of connected vehicles subject to dos attack: theory and application, IEEE Trans. Syst. Man Cybern.: Syst.
[64] X. Sun, R. Nambiar, M. Melhorn, Y. Shoukry, P. Nuzzo, Dos-resilient multi-robot temporal logic motion planning, in: 2019 International Conference on Robotics and Automation (ICRA) IEEE, 2019, pp. 6051–6057.
[65] E.M. Amullen, S. Shetty, L.H. Keel, Model-based resilient control for a multi-agent system against denial of service attacks, in: 2016 World Automation Congress (WAC), IEEE, 2016, pp. 1–6.
[66] E.M. Amullen, S. Shetty, L.H. Keel, Secured formation control for multi-agent systems under dos attacks, in: 2016 IEEE Symposium on Technologies for Homeland Security (HST), IEEE, 2016, pp. 1–6.
[67] M. Mukhandi, D. Portugal, S. Pereira, M.S. Couceiro, A novel solution for securing robot communications based on the mqtt protocol and ros, in: 2019 IEEE/SICE International Symposium on System Integration (SII), IEEE, 2019, pp. 608–613.
[68] B. Breiling, B. Dieber, P. Schartner, Secure communication for the robot operating system, in: 2017 annual IEEE international systems conference (SysCon), IEEE, 2017, pp. 1–6.