Response Time Analysis for RT-MQTT Protocol Grounded on SDN

Ehsan Shahri[1]
Department of Electronics, Telecommunications and Informatics (DETI),
University of Aveiro, Portugal
Institute of Telecommunications, Campus de Santiago, Aveiro, Portugal

Paulo Pedreiras
Department of Electronics, Telecommunications and Informatics (DETI),
University of Aveiro, Portugal
Institute of Telecommunications, Campus de Santiago, Aveiro, Portugal

Luis Almeida
Research Center in Real-Time and Embedded Computing Systems (CISTER), Porto, Portugal
Faculty of Engineering, University of Porto (FEUP), Portugal

Abstract

The current industry trend is to replace the use of custom components with standards-based Commercially available Off-The-Shelf (COTS) based hardware and protocols. Furthermore, the emergence of new industrial paradigms, such as Industry 4.0 and the Industrial Internet of Things, sets additional requirements regarding e.g. scale, transparency, agility, flexibility and efficiency. Therefore, in these domains, application layer protocols such as Message Queuing Telemetry Transport protocol (MQTT) are gaining popularity, in result of their simplicity, scalability, low resource-usage and decoupling between end nodes. However, such protocols were not designed for real-time applications, missing key features such as determinism and latency bounds. A recent work proposed extending MQTT with real-time services, taking advantage of Software Defined Networking (SDN) to manage the network resource. These extensions allow applications to specify real-time requirements that are then captured by a resource manager and used to reserve the necessary resources at the network layer. This paper shows that such MQTT extended architecture is analyzable from a worst-case timing perspective. We derive a system model that captures the real-time features and we present a response-time analysis to assess the schedulability of the real-time traffic. Finally, we validate the analysis with a set of experimental results.

2012 ACM Subject Classification Networks

Keywords and phrases Real-time systems, OpenFlow, fixed-priority non-preemptive scheduling, response time analysis, MQTT

Digital Object Identifier 10.4230/OASIcs.NG-RES.2023.5

Funding This work is funded by Portuguese national funds through FCT/MCTES and, when applicable, co-funded by European Community funds, under projects IT-UIDB/50008/2020-UIDP/50008/2020 and CISTER-UIDB/04234/2020, as well as the FCT scholarship PD/BD/137388/2018.

1 Introduction

Real-time computing systems have been widely employed in industrial domains for many years, from robotics [9] to industrial control [26, 24] and industrial automation [14]. Recent trends, pushed by Industry 4.0 and the Industrial Internet-of-Things (IIoT), have brought Information and Communication Technologies (ICT) into industrial operations towards

[1] Corresponding author

increased scalability, transparency, agility, flexibility and efficiency. However, while improving these properties, ICT have typically disregarded their own timing behavior, conflicting with the real-time requirements found in industrial operations, namely quick response times and high predictability and stability. Another difficulty that came with ICT is their typical high overhead that may conflict with deployments within embedded resource-constrained hardware, commonly found in industrial settings. Moreover, Industry 4.0 and IIoT imply an increase in the complexity and heterogeneity of the data exchanges, both in terms of nature and requirements, putting together low and high data-rate flows, low and high bandwidth flows, and critical and non-critical flows.

The Message Queuing Telemetry Transport protocol (MQTT) [20] is one of the most popular application-layer protocols within the scope of the Internet-of-Things. It was developed already with a perspective of applying ICT in operations and it is finding a growing use within the IIoT, too. Main factors that contribute to its popularity include its simplicity, low footprint, scalability and effective publisher-subscriber messaging model. Standard MQTT relies on TCP/IP networks that provide ordered and lossless bi-directional channels. However MQTT misses support for real-time requirements, having Quality-of-Service (QoS) policies that only address message delivery. This limitation has been tackled by the scientific community [21, 16, 7, 18]. Specifically the work in [19] proposes an architecture called RT-MQTT that extends MQTT with real-time services, allowing applications to define real-time requirements that are translated to network reservations which are enforced using Software Defined Networking (SDN), particularly using the OpenFlow protocol.

Despite the referred efforts to reconcile real-time requirements with MQTT, formal traffic schedulability analysis is still lacking. In this paper we build on RT-MQTT and detail and formalize its system model and provide a response-time analysis for critical real-time traffic that assumes fixed-priority non-preemptive packet transmissions. The analysis and its implementation in a practical setup are validated with a set of experimental results that also allow assessing the tightness of the response-time analysis. Our aim is to show the analyzability of RT-MQTT. To the best of our knowledge, this is the first schedulability analysis for MQTT-based networks.

The rest of the paper is structured as follows. Section 2 discusses traffic schedulability analysis for multi-hop networks. Section 3 describes the OpenFlow switches as the networking elements in RT-MQTT. Section 4 introduces the multi-hop network architecture and system model. The response-time analysis is presented in Section 5 and validated with experiments in Section 6. Section 7 concludes the paper and discusses future work.

2 Related Work

Using ICT in the industrial operations domain came with several challenges, such as timing analysis. Particularly, analyzing the worst-case timing behavior of ICT protocols is frequently unfeasible. For this reason, we did not find a concrete schedulability analysis for real-time traffic on MQTT, which is not a surprise given the lack of protocol support. However, as we show in this paper, the extensions of RT-MQTT leverage the real-time network capabilities of SDN and make the framework analyzable.

Worst-case response-time analysis methods for real-time multi-hop networks fall in three main categories [28, 17]: Network Calculus (NC) [10], Trajectory approach [13] and holistic analysis [23]. NC [30, 29] considers that network elements and arrival flows are characterized by a service curve and arrival curve, respectively. Provided with this information, NC allows computing the maximum delay that each flow can suffer at each network element, as well

as the maximum size of the waiting queues and the corresponding departure curves. NC provides deterministic results, but requires the determination of arrival and service curves, usually in the form of bounds, which introduces some degree of pessimism.

The Trajectory approach [22, 25] computes the latest starting time of a packet on the last node visited, then moving backwards through the sequence of nodes visited by the packet (i.e. the packet trajectory), identifying the preceding packets and busy periods that affect the latency of the packet in each node. This approach is the tightest among the three categories, in general, but is is more complex to implement and validate, particularly when considering unconstrained routing schemes in which message flows can cross multiple times.

The holistic analysis [8] is meant for distributed systems and it computes the minimum and maximum response-time of the tasks and uses this jitter to compute the minimum and maximum response-times of the messages they generate. This jitter is then used to reassess the response time of the tasks triggered by the messages. The whole process is repeated until the response times of tasks and messages converge. This method can be simplified if we focus on the network, only, particularly for a multi-hop switched network. At each hop, we use release jitters to compute both the response times of the messages and their jitters, which will be used as release jitters of the next hop. Thus, the response times are computed in just one pass, from source to destination. This approach is more pessimistic than the trajectory approach, as it considers worst-case scenarios on every node that often cannot occur in operation. However, it is simpler to implement and verify, reason why it was adopted in this work as a first approach to show the real-time analyzability of RT-MQTT.

3 OpenFlow Switch Structure

The core network elements of RT-MQTT are OpenFlow switches that forward packets (a.k.a frames) in an SDN environment and can be implemented in software or hardware. SDN decouples the network data and control planes, the former being implemented in the switch to process packets, while the latter is implemented in a separate SDN controller that makes high-level packet handling decisions and configures switches accordingly. OpenFlow is the de facto protocol used for switch-controller interactions (a.k.a. southbound interface). Figure 1 shows the OpenFlow pipeline in the switches that receives frames at ingress ports and sends them to a set of flow tables installed by a controller. Each flow table contains a set of Flow Entries with: i) a priority to sort matching; ii) filters to identify incoming frames; iii) associated instructions; and iv) fields for statistics.

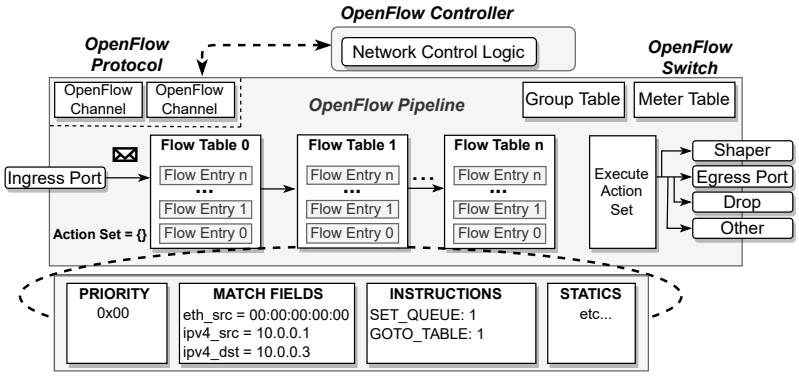


Figure 1 Overview of the OpenFlow pipeline.

When a packet matches the filters of a given flow entry, the switch performs the associated instructions. Once the packet reaches the last table or is not directed to a subsequent one by the matched entry, the switch executes the current action set that may send the packet to a group table or forward it to an OpenFlow egress port. Conversely, if a packet does not match any flow entry, it is handled as a table-miss. The actions executed in this case are also configurable including dropping the packet, forwarding to a subsequent table, or sending to the controller on a packet-in message via the control channel. If the table-miss flow entry does not exist, unmatched packets are dropped by default. Sending the controller a flow request (packet-in message) for every unknown packet can overwhelm the controller since it has to determine the forwarding path and forwarding rules for every new packet and then install them in the flow tables in all the involved switches. Finally, each egress port features several prioritized queues to enforce traffic segregation for time-sensitive flows.

4 RT-MQTT Network Architecture

The RT-MQTT network architecture (Figure 2) is based on the MQTT application layer protocol supported by OpenFlow switches. MQTT permits the use of multiple brokers for fault-tolerance and load balancing purposes, allowing the system to scale and tolerate broker failures. Similarly, Openflow also features mechanism that allow networks to scale and tolerate faults [2]. However, scalability and fault-tolerance mechanisms are out of the scope of this paper and will not be further addressed.

The RT-MQTT topology comprises an OpenFlow controller (OF-Controller), OpenFlow switches (OF-Switches), an MQTT broker, a real-time network manager (RT-NM) and MQTT clients (IIoT nodes). The OF-Controller is connected to the OF-Switches, having a global view of the network and storing all information in the OF-DataBase (OF-DB). RT-MQTT allows applications to explicitly specify real-time requirements in the User Properties of their MQTT packets. The RT-NM intercepts all MQTT messages to extract possible real-time requirements data. It is logically placed between MQTT clients and the broker, desirably executing in the same node. These requirements are subsequently conveyed to the OF-Controller that processes them and manages the flow tables of the OF-Switches to create corresponding real-time channels.

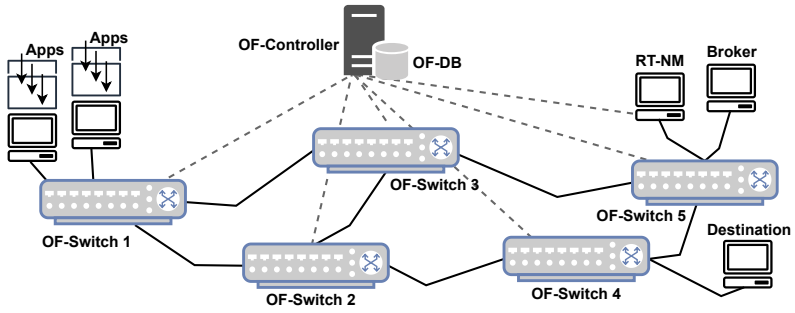


Figure 2 High-level RT-MQTT system architecture.

4.1 Message Model

RT-MQTT classifies packet flows (a.k.a messages) in `real-time`, or time-sensitive, and `non-real-time`, such as normal MQTT messages and general background traffic. Client nodes are assumed to run an operating system with minimal real-time capabilities (we

use real-time enabled Linux with regular network stack) and can generate real-time and non-real-time traffic concurrently. We model each real-time message m_i as sporadic. The message set Γ, composed of N messages, is defined as follows:

$$\Gamma = \{m_i(C_i, PS_i, D_i, T_i, P_i, S_i, DS_i, \mathcal{L}_i, n_i), i = 1...N\} \quad (1)$$

The semantics of the parameters are the following:

$\boldsymbol{i}$: message index used as identifier;
$\boldsymbol{C_i}$: total transmission time of the message;
$\boldsymbol{PS_i}$: maximum packet size among the packets that compose m_i;
$\boldsymbol{D_i}$: deadline, maximum allowed time between transmission and reception of a message;
$\boldsymbol{T_i}$: minimum interval between consecutive message source publications, with $D_i \leq T_i$;
$\boldsymbol{P_i}$: message priority;
$\boldsymbol{S_i}$: source node;
$\boldsymbol{DS_i}$: destination node;
$\boldsymbol{\mathcal{L}_i}$: set of links that m_i passes through, including local-links and inter-links;
$\boldsymbol{n_i}$: number of links that m_i crosses, i.e., $n_i = |\mathcal{L}_i|$.

As common in IIoT applications we consider single packet messages. Concerning the path, each element in $\mathcal{L}_i$ has a duplet $l =< x, y >$ representing a link l between node/switch x and node/switch y. A link between a node and a switch is called local-link, while the link between two switches is an inter-link. The direction of message transmission in that link is indicated by the sequence within the duplet. The set of links in the route of m_i is $\mathcal{L}_i = \{l_k | k = 1..n_i\}$. Each message has a defined priority assigned in ascending order (larger value of P_i means higher priority). Since MQTT generally relies on unicasting [15], we restrict the analysis to unicast streams, with only one destination port per message.

4.2 Scheduling Model

As common in current networks, packet transmission is non-preemptive. Thus, we use non-preemptive fixed priorities scheduling with FIFO strategy within each priority level. When a message arrives at an ingress port it is processed by the pipeline that defines the interaction with the flow tables in that switch and (in the normal case) places the packet in the output port queue determined by the message path and priority. Note that each output port corresponds to a link in the message path. Generally, message m_i can suffer two types of delays in any output queue, namely blocking and interference delays as explained next:

Blocking delay is the longest time that m_i may have to wait when it arrives at an output port and message m_j with lower priority ($P_j < P_i$) is already being transmitted in that port. This delay can be computed considering the longest packet among all lower priority messages that share the same output port in each switch. For simplicity of analysis we assume that the blocking delay can be as long as the transmission time of the longest configured packet length, i.e. the Maximum Transmission Unit (MTU). We also consider the MTU to be the same in all nodes.

Interference delay is the longest time that message m_i must wait due to the transmission of all messages m_j with higher or same priority ($P_j \geq P_i$) that share the same output port. The worst-case interference delay requires that all messages with similar priority arrive just before m_i arrives and that all higher priority messages arrive with a pattern defined in the next section.

Figure 3 shows the scheduling model of RT-MQTT with its logical channels and the corresponding physical links between source and destination nodes, conveying real-time and non-real-time traffic concurrently. Once the logical channel is established, the route is determined by a sequence of physical links, each one with origin at a specific output port of a particular switch.

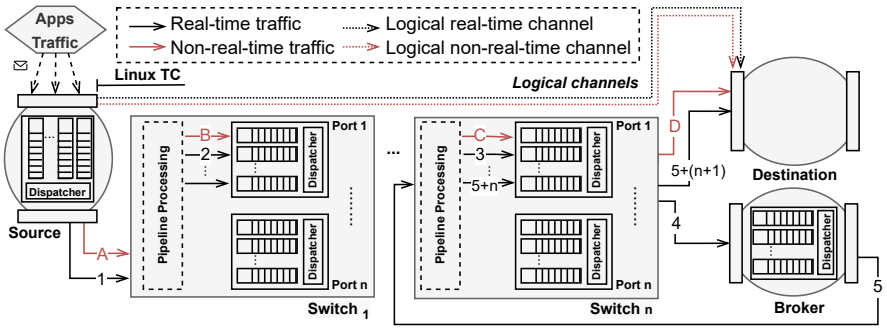


Figure 3 Scheduling model of RT-MQTT.

5 Response Time Analysis

To analyze RT-MQTT we follow a response time analysis method for fixed priorities. We consider the response time of a real-time message as being composed of three parts: (i) the response time from the `source` to the `broker`; (ii) the response time inside the `broker`; and (iii) the response time from the `broker` to the `destination`. While the response times of parts (i) and (iii) are communication delays, (ii) is a computational delay. Both communication and computation delays are interdependent. The holistic approach [23] solves this interdependence iteratively to find the global worst-case response times but, in this case, it requires knowing the computational structure of the broker, which is generally unknown.

Although there is some work in the literature addressing the real-time behavior of the broker, e.g. applying offset-based response time analysis [12], we opted for leaving it outside our current work and focusing on network delays, only. Moreover, the response time analysis for parts (i) and (iii) is similar, just switching the source in (i) with the broker in (iii) and the broker in (i) with the destination in (iii). Therefore, hereinafter we focus only on part (i), i.e., developing a response time analysis for the traffic from source nodes to the broker.

5.1 Response Time Analysis From Source to Broker

We analyze the full path from source to broker as a series of links, each contributing with additional delay. The total response time is then obtained by adding the response times of all individual links in the message path plus the time taken by the switches to move messages across, through their pipelines. We refer to this last component as the switching delay. For the sake of simplicity of analysis, we consider worst-case conditions in all links and switches, at the expense of additional pessimism.

5.1.1 Single link response time analysis

In each link, messages are serialized in the output port of its source, be it a source node or an OF-Switch. As discussed in the previous section, each output port enforces fixed-priorities non-preemptive packet scheduling, with FIFO handling within the same priority level. To analyze this model we apply the conventional computation of response time without preemption in uniprocessors based on cumulative delays [4, 5, 6].

A crucial aspect of this analysis is determining the critical instant, i.e., the message release pattern that leads to the worst-case interference that a message can suffer. In our case, we use as critical instant a synchronous pattern in which a message m_i is released immediately after the release of a lower priority message with maximum size (maximum blocking) and immediately after all equal priority messages and together with all higher priority messages considering their maximum release jitter (maximum interference).

Non-preemptive transmissions are subject to the push-through effect according to which a message m_i can delay higher priority messages through blocking that in turn will generate higher interference in the following instances of m_i itself. For this reason, the worst-case response time of m_i in a given output port may occur at instances beyond the one that is released at a critical instant, within the so-called occupied period. The number of instances following a critical instant that have to be checked to determine the worst-case response time is given by $Q_i = \lceil (w_i + J_i)/T_i \rceil$, where J_i is the release jitter and w_i is the length of the level-i busy period. This period is computed as in Equation 2 using fixed point iteration, starting with $w_i^0 = B_i + C_i$ and ending when $w_i^{n+1} = w_i^n$. The summation term labeled `hep` represents the total interference due to invocations of higher and equal priority messages released strictly before the end of the busy period.

$$w_i^{n+1} = B_i + \sum_{\forall\, j \in hep(i)} \lceil \frac{w_i^n + J_j}{T_j} \rceil C_j \tag{2}$$

The so-called level-i occupied period starts at the critical instant and extends until the following level-i idle period, i.e., when the queues of priorities P_i and higher of the output port become empty. This period includes the q previous instances of m_i and it represents the maximum delay that the next instance of m_i can suffer before starting transmission. Equation 3 gives an upper bound to its length $v_i(q)$ and can also be solved through fixed-point iteration with a possible initial value $v_i^0(q) = B_i + qC_i$ and ending when either $v_i^{n+1}(q) = v_i^n(q)$ or when $v_i^{n+1}(q) + C_i - qT_i > D_i - J_i$ in which case the deadline cannot be guaranteed.

$$v_i^{n+1}(q) = \; B_i + qC_i + \sum_{\forall\; j \in hep(i)} (\lfloor \frac{v_i^n(q) + J_j}{T_j} \rfloor + 1) C_j \tag{3}$$

The worst-case response time of an instance preceded by q instances can then be obtained with $RT_i(q) = v_i(q) + C_i - qT_i$. Equation 4 gives us the worst-case response time for m_i.

$$RT_i = \; max_{q=0,1\ldots Q_i-1}(v_i(q) + C_i - qT_i) \tag{4}$$

5.1.2 Switching delay calculation

As referred before, the switching delay affects all messages crossing a switch even without blocking or interference by other messages. Figure 4 [27] helps understanding the switching delay of OpenFlow switches, particularly implemented in software, as in our system.

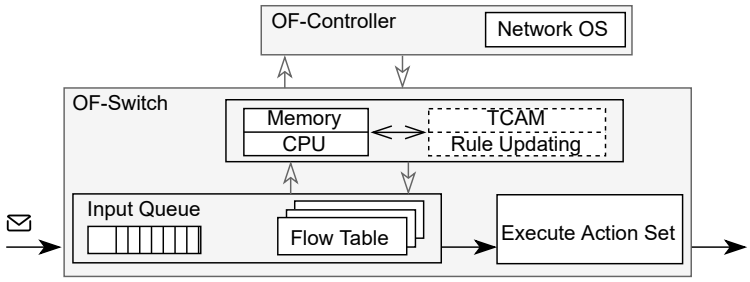


Figure 4 The switching delay in an OF-Switch.

We consider the switching delay as resulting from three components, the time to manage the input queues (essentially memory operations), the time to process the flow tables and the time to execute the action set. We refer to the switching delay affecting m_i as SD_i. The switching delay must be bounded for a bounded response time, but this bound can be different for each switch because it depends on the input load, on the number of flow tables and on the complexity of the action set. These aspects have been studied in the literature. For example, the switching delay increases with increasing flow arrival rate since it requires more bandwidth from the local CPU in the SDN switch [27]. The switching delay may also increase with decreasing packet size because it typically promotes higher arrival rate again imposing higher processing burden on the switch CPU [3]. The forwarding table is another source of overhead. The longer it is the higher the switching delay. In practice, the SD is normally measured and then used in the analysis.

5.1.3 Response time calculation algorithm

Algorithm 1 shows the computation of the worst-case response time for m_i for the total route from source to broker, taking as input the network topology and the message set.

Algorithm 1 WCRT calculation for m_i.

Input: G, Γ
Output: $RT_i^{TotalRoute}$

/* A. Compute delays and jitter at source node: */
$RT_i^{TotalRoute} = ResponseTimeCalc(i, 1)$
$J_i^{acc} = J_i + J_{i,1}^{QP}$
$k = 2$
/* B. Compute delays and jitter for each switch: */
while $k \leq n_i$ **do**
    $SD_{i,k} = SwitchingDelayCalc(i, k)$
    $J_i^{acc} = J_i^{acc} + J_{i,k}^{SD}$
    $RT_{i,k} = ResponseTimeCalc(i, k)$
    $J_i^{acc} = J_i^{acc} + J_{i,k}^{QP}$
    $RT_i^{TotalRoute} = RT_i^{TotalRoute} + RT_{i,k} + SD_{i,k}$
    $k = k + 1$
end while

The algorithm has two parts. The first part (A) processes the output link of the source node (corresponding to k=1), including the computation of the response time ($ResponseTimeCalc(i, 1)$ according to Equation 4) and output jitter (J_i plus $J_{i,1}^{QP}$ which is the response time jitter). These values are used to initialize two accumulators that will allow building the response time for the total route ($RT_i^{TotalRoute}$) and the release jitter (J_i^{acc}) that will affect each subsequent link along the path. The second part in the algorithm (B) is very similar to the first one with the only difference that it also computes the switching delay introduced by the switch under analysis and the additional jitter it may cause. This is repeated from the second to the last link in the path of m_i.

6 Performance Assessment

We validate the analysis presented before with an empirical study using the Mininet emulation framework applied in multiple scenarios of different complexity. For consistency, we also focus on the publishers side, measuring the time intervals from the publication instant (writing to the respective socket) to the respective reception at the broker and comparing with the corresponding analytical response times.

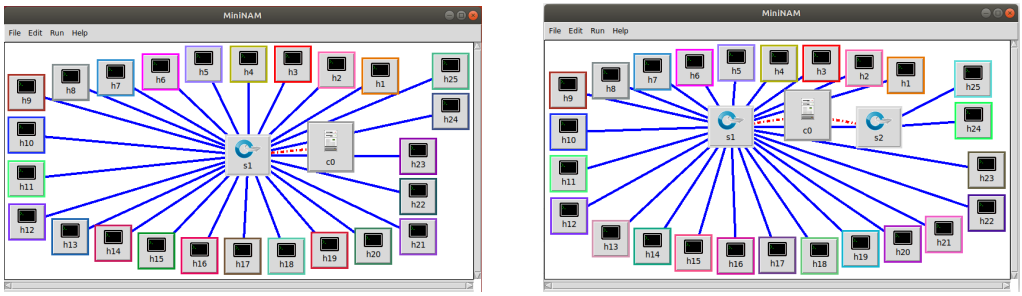
6.1 Emulation Setup

We used the Mininet virtual network emulator, version 2.3.0d6 `http://mininet.org/`, together with Eclipse Mosquitto [11] (v2.0.10) and Eclipse Paho MQTT library to create MQTT clients and broker. Mininet is executed on a laptop computer featuring a 4.9 GHz Intel Core i7 processor and 16 GB of RAM. The SDN controller is the RYU OF-Controller [1] and it is executed on the same laptop computer.

In the emulation experiments, the QoS of all MQTT messages is set to 1 (deliver at least once). This QoS level favors reliability over timeliness given its positive acknowledge and retry mechanism, and it was used since fault-tolerance is important for many IIoT applications. However, this is not expected to have a significant impact on cabled Ethernet networks, given their low error rate.

The operational environment included heterogeneous data exchanges mimicking the diversity of industrial scenarios created with the *Distributed Internet Traffic Generator* (D-ITG) `http://traffic.comics.unina.it/software/ITG/` for TCP packets, *VLC media player* to generate audio/video streams and *vsftpd* to transfer files using the File Transfer Protocol (FTP) `https://linuxconfig.org/how-to-setup-and-use-ftp-server-in-ubuntu-linux`. These were all non-real-time traffic sources with bandwidth limited to 10 Mbit/s, 32 kbit/s, and 800 kbit/s for D-ITG, VLC, and vsftpd, respectively. All links are configured with 100 Mbit/s capacity as still commonly found in industry.

The experiments consider two network topologies with different levels of complexity, named `Single-Switch` and `Dual-Switch`, comprising 1 and 2 OF-Switches (Figure 5). For each topology we generate three different load-levels, labeled `A`, `B`, and `C`, involving publications in different real-time topics. The real-time messages were published with a nominal period chosen to cover the interval [2 15]ms and using a single Ethernet packet with maximum size (1500 bytes), leading to a transmission time of 123μs. The priority assignment was Deadline-Monotonic. Since we were interested in assessing the accuracy of the analysis, we considered schedulable message sets, only.



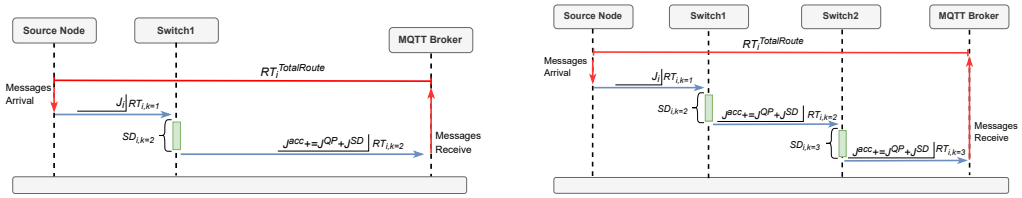
(a) Single-Switch network topology. **(b)** Dual-Switch network topology.

Figure 5 Network topologies used in Mininet.

Both topologies include the OF-controller (node `c0`), as well as the `RT-NM` and the MQTT broker (both in node `h24`), plus up to 20 MQTT publishers (nodes `h1` to `h20`), each publishing a single real-time message (m_1 to m_{20}, resp.) to the broker, favoring contention in the network and not in the nodes network stacks. In `Load-Level A` just 5 publishers are active (nodes $h1$ to $h5$), `Load-Level B` uses 10 publishers (nodes $h1$ to $h10$) and `Load-Level C` uses all the 20 publisher nodes. All configurations considered the non-MQTT traffic including D-ITG data from node `h21`, VLC streams from `h22` and *vsftpd* data from `h23`, all directed to node `h25` which is also the subscriber of all MQTT topics. On our measurement host, we use `libpcap` to timestamp the packets for each flow, measuring the associated jitters and the switching delays. For reduced interference, we first log the timestamps in memory and when the experiment is completed, the results are dumped to disk and processed, enabling us to calculate the switching delay and response times.

6.2 Experimental Results

The measurement points in the experiments are shown in Figure 6. The experiments tested all combinations of topology {Single-Switch & Dual-Switch} and load-levels {A, B, C}. Each combination was executed 1000 times with each publisher generating 100 messages per run.

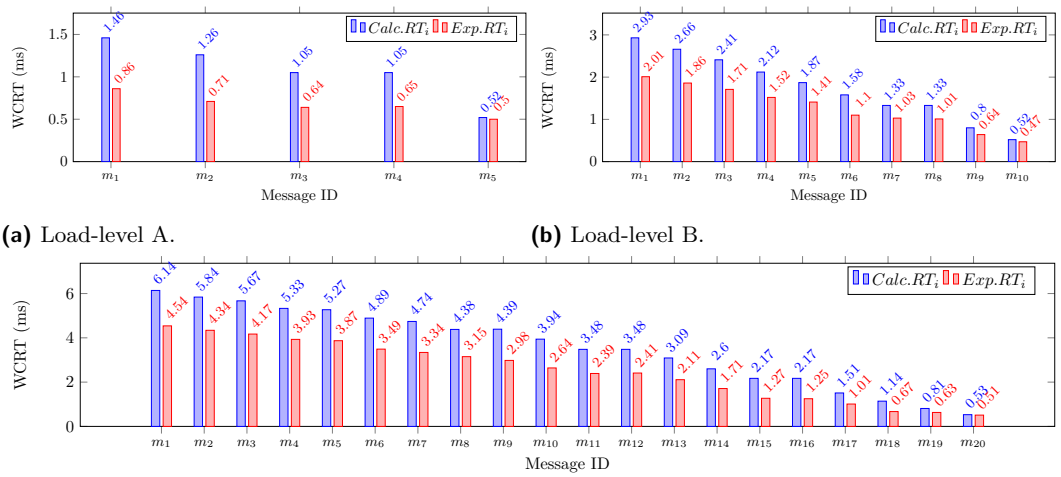


(a) Response time in the Single-Switch topology. **(b)** Response time in the Dual-Switch topology.

Figure 6 Response time measurements.

Figures 7 and 8 show the analytical worst-case response times ($CalcRT_i$) against the maximum observed response times in the experiments ($ExpRT_i$) for the real-time messages.

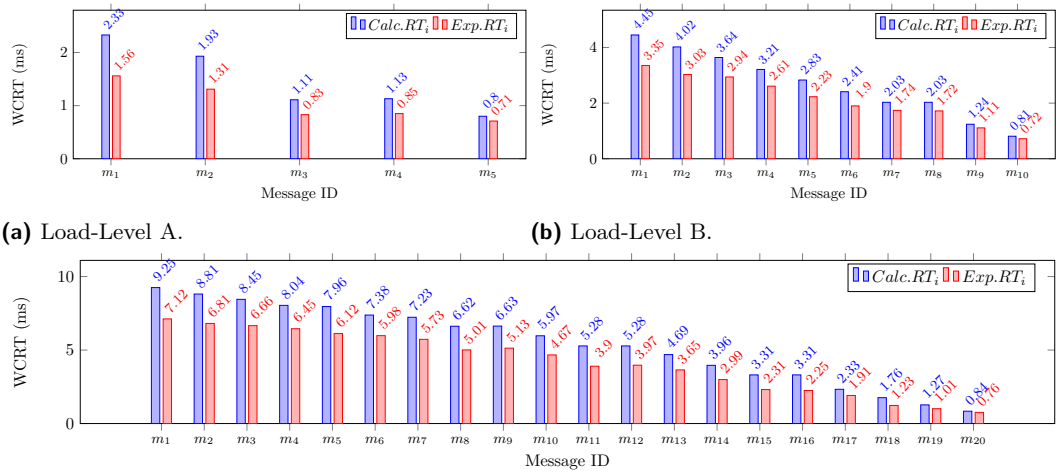
From these figures we can observe three main aspects. Firstly, for all real-time messages we see $CalcRT > ExpRT$, meaning the proposed analysis is safe. Secondly, the difference $CalcRT - ExpRT$, which approximates the pessimism of the analysis, is relatively small for higher priority messages, increasing as the priority decreases. This is expected since the analysis of the lower priority messages includes more pessimistic assumption, e.g. in the interference. Finally, both $CalcRT$ and $ExpRT$ increase similarly with the load level and



(a) Load-level A. **(b)** Load-level B.

(c) Load-level C.

Figure 7 Analytical (CalcRT) versus observed (ExpRT) WCRT for Single-Switch topology.



(a) Load-Level A. **(b)** Load-Level B.

(c) Load-Level C.

Figure 8 Analytical (CalcRT) versus observed (ExpRT) WCRT for the Dual-Switch topology.

with the number of switches. Indirectly we can also see a consistent behavior of the real-time messages, despite the reasonable load of non-real-time traffic in the background. All details concerning the message sets and the analytical and experimental results are shown in tabular form in Annex A, in Tables 1–6.

7 Conclusions and Future Work

Despite the increasing popularity of MQTT in the scope of IoT and IIoT, its Quality-of-Service policies do not support timeliness requirements. A recent work addressed this limitation proposing a set of extensions to the MQTT protocol that allow applications to specify real-time requirements (RT-MQTT). Such specifications are then used by a resource manager, implemented in SDN/Openflow, to create real-time channels with suitable attributes, thus instantiating adequate network reservations to enforce the desired temporal behavior.

In this paper, we show that it is possible to apply existing response time analysis to RT-MQTT on the multi-hop SDN/OpenFlow switched network to derive worst-case response time upper bounds to the real-time traffic. In particular, we used the standard response time analysis for non-preemptive fixed-priority scheduling of sporadic messages. The analysis is validated empirically within the Mininet emulator framework, being safe and with relatively low pessimism, particularly for the higher priority traffic. Future work includes the analysis of the broker temporal behavior to support an end-to-end (publisher-to-subscriber) delay model. We will also apply other analytical techniques, e.g., the trajectory approach, in an attempt to reduce the analysis pessimism.

References

1 What's ryu. URL: `https://ryu-sdn.org/`.

2 M. Bala Krishna and Pascal Lorenz. Proactive replication scheme for resilient content delivery in software defined networks. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019. `doi:10.1109/GLOBECOM38437.2019.9013441`.

3 Andrea Bianco, Robert Birke, Luca Giraudo, and Manuel Palacin. Openflow switching: Data plane performance. In *2010 IEEE International Conference on Communications*, pages 1–5. IEEE, 2010.

4 Reinder J Bril, Johan J Lukkien, and Wim FJ Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, pages 269–279. IEEE, 2007.

5 Reinder J Bril, Johan J Lukkien, and Wim FJ Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42(1):63–119, 2009.

6 Robert I Davis and Alan Burns. Response time upper bounds for fixed priority real-time systems. In *2008 Real-Time Systems Symposium*, pages 407–418. IEEE, 2008.

7 Yong-Seong Kim et al. MQTT Broker with Priority Support for Emerg. Events in IoT. *Sensors and Materials*, 2018.

8 J Javier Gutiérrez, J Carlos Palencia, and Michael Gonzalez Harbour. Holistic schedulability analysis for multipacket messages in afdx networks. *Real-Time Systems*, 50(2):230–269, 2014.

9 Hamidreza Kasaei and Mohammadreza Kasaei. Mvgrasp: Real-time multi-view 3d object grasping in highly cluttered environments. *arXiv preprint*, 2021. `arXiv:2103.10997`.

10 Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

11 Roger A Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13), 2017.

12 Jukka Mäki-Turja, Kaj Hänninen, and Mikael Sjödin. On sustainability for offset based response-time analysis. In *7th Conference on the Engineering of Computer Based Systems*, pages 1–7, 2021.

13 Steven Martin and Pascale Minet. Schedulability analysis of flows scheduled with fifo: application to the expedited forwarding class. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pages 8–pp. IEEE, 2006.

14 Alessandro Massaro, Giuseppe Mastandrea, Luigi D'Oriano, Giuseppe Rocco Rana, Nicola Savino, and Angelo Galiano. Systems for an intelligent application of automated processes in industry: a case study from "pmi iot industry 4.0" project. In *2020 IEEE International Workshop on Metrology for Industry 4.0 IoT*, pages 21–26, 2020. `doi:10.1109/MetroInd4.0IoT48571.2020.9138231`.

15 Jun-Hong Park, Hyeong-Su Kim, and Won-Tae Kim. Dm-mqtt: An efficient mqtt based on sdn multicast for massive iot communications. *Sensors*, 18(9):3071, 2018.

16 Changheon Oh Seongjin Kim. A Study on Method for Message Processing by Priority in MQTT Broker. *JKIICE-Journal of the Korea Institute of Information and Communication Engineering*, Jul. 2017.

17 Lui Sha, Tarek Abdelzaher, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, Aloysius K Mok, et al. Real time scheduling theory: A historical perspective. *Real-time systems*, 28(2):101–155, 2004.

18 Ehsan Shahri, Paulo Pedreiras, and Luis Almeida. Enhancing mqtt with real-time and reliable communication services. In *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, pages 1–6. IEEE, 2021.

19 Ehsan Shahri, Paulo Pedreiras, and Luis Almeida. Extending mqtt with real-time communication services based on sdn. In *2022 Sensor Applications in Industrial Automation (ISSN 1424-8220)*, pages 1–6. Sensors SAIA SI, 2022.

20 OASIS Standard. Mqtt version 5.0. *Retrieved June*, 22:2020, 2019.

21 Hiroshi Mineno Takuma Tachibana, Tetsuo Furuichi. Implementing and Evaluating Priority Control Mechanism for Heterogeneous Remote Monitoring IoT System. *MOBIQUITOUS '16 Adjunct Proceedings, Hiroshima, Japan*, December,01,2016.

22 Xueqian Tang, Qiao Li, Guangshan Lu, and Huagang Xiong. A revised trajectory approach for the worst-case delay analysis of an afdx network. *IEEE Access*, 7:142564–142573, 2019. doi:10.1109/ACCESS.2019.2943543.

23 Ken Tindell and John Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and microprogramming*, 40(2-3):117–134, 1994.

24 Jorge Otávio Trierweiler. Real-time optimization of industrial processes. In *Encyclopedia of Systems and Control*, pages 1827–1836. Springer, 2021.

25 Long Yan, Zexiong Luo, Xueqian Tang, and Yunwen Kong. Timing analysis of rate-constrained traffic in ttethernet using extended trajectory approach. In *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*, pages 1039–1042. IEEE, 2020.

26 Shen Yin, Juan J Rodriguez-Andina, and Yuchen Jiang. Real-time monitoring and control of industrial cyberphysical systems: With integrated plant-wide monitoring and control framework. *IEEE Industrial Electronics Magazine*, 13(4):38–47, 2019.

27 Ting Zhang and Bin Liu. Exposing end-to-end delay in software-defined networking. *International Journal of Reconfigurable Computing*, 2019, 2019.

28 Luxi Zhao, Paul Pop, and Silviu S. Craciunas. Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus. *IEEE Access*, 6:41803–41815, 2018. doi:10.1109/ACCESS.2018.2858767.

29 Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. Timing analysis of rate-constrained traffic in TTEthernet using network calculus. *Real-Time Systems*, 53(2):254–287, March 2017. doi:10.1007/s11241-016-9265-0.

30 Boyang Zhou, Isaac Howenstine, Siraphob Limprapaipong, and Liang Cheng. A survey on network calculus tools for network infrastructure in real-time systems. *IEEE Access*, 8:223588–223605, 2020.

A Detailed Experimental Results

Tables 1–3 for {Single-Switch} network topology and Tables 4–6 for {Dual-Switch} network topology show the message set details and associated worst-case response time analytical and observed values in three load-levels {A, B, C}, corresponding to Figures 7 and 8.

Table 1 Single-Switch network topology/load-level A.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Single-Switch Load-Level: A	m_1	1500	12.3	123	14	15	h1/p1	123	1.03	37	0	1.27	6	31	1.46	0.86
	m_2	1500	12.3	123	14	15	h2/p2	123	0.77	36	0	1.01	6	32	1.26	0.71
	m_3	1500	12.3	123	5	6	h3/p3	123	0.52	36	0	0.77	6	31	1.05	0.64
	m_4	1500	12.3	123	5	6	h4/p3	123	0.52	37	0	0.77	6	32	1.05	0.65
	m_5	1500	12.3	123	1	2	h5/p5	123	0	37	0	0.24	6	32	0.52	0.50

Table 2 Single-Switch network topology/load-level B.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Single-Switch Load-Level: B	m_1	1500	13.53	123	14	15	h1/p1	123	2.33	48	0	2.67	6	34	2.93	2.01
	m_2	1500	13.53	123	14	15	h2/p2	123	2.13	48	0	2.38	6	34	2.66	1.86
	m_3	1500	13.53	123	11	12	h3/p3	123	1.88	47	0	2.13	6	35	2.41	1.71
	m_4	1500	13.53	123	11	12	h4/p4	123	1.59	46	0	1.84	6	34	2.12	1.52
	m_5	1500	13.53	123	8	9	h5/p5	123	1.34	47	0	1.59	6	33	1.87	1.41
	m_6	1500	13.53	123	8	9	h6/p6	123	1.06	47	0	1.30	6	34	1.58	1.10
	m_7	1500	13.53	123	5	6	h7/p7	123	0.81	46	0	1.05	6	34	1.33	1.03
	m_8	1500	13.53	123	5	6	h8/p7	123	0.81	46	0	1.05	6	35	1.33	1.01
	m_9	1500	13.53	123	1	2	h9/p9	123	0.28	47	0	0.53	6	35	0.80	0.64
	m_{10}	1500	13.53	123	1	2	h10/p10	123	0	46	0	0.24	6	35	0.52	0.47

Table 3 Single-Switch network topology/load-level C.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Single-Switch Load-Level: C	m_1	1500	111	123	14	15	h1/p1	123	5.60	61	0	5.85	6	42	6.14	4.54
	m_2	1500	111	123	14	15	h2/p2	123	5.31	61	0	5.55	6	42	5.84	4.34
	m_3	1500	111	123	11	12	h3/p3	123	5.14	60	0	5.38	6	42	5.67	4.17
	m_4	1500	111	123	11	12	h4/p4	123	4.80	61	0	5.04	6	41	5.33	3.93
	m_5	1500	111	123	8	9	h5/p5	123	4.74	60	0	4.98	6	42	5.27	3.87
	m_6	1500	111	123	8	9	h6/p6	123	4.35	60	0	4.60	6	43	4.89	3.49
	m_7	1500	111	123	6	7	h7/p7	123	4.25	59	0	4.49	6	43	4.74	3.34
	m_8	1500	111	123	6	7	h8/p8	123	3.85	60	0	4.09	6	41	4.38	3.15
	m_9	1500	111	123	4	5	h9/p9	123	2.98	59	0	4.10	6	42	4.39	2.98
	m_{10}	1500	111	123	4	5	h10/p10	123	3.40	59	0	3.65	6	42	3.94	2.64
	m_{11}	1500	111	123	4	5	h11/p11	123	2.95	61	0	3.20	6	42	3.48	2.39
	m_{12}	1500	111	123	4	5	h12/p11	123	2.95	60	0	3.20	6	43	3.48	2.41
	m_{13}	1500	111	123	2	3	h13/p13	123	2.55	60	1	2.80	6	42	3.09	2.11
	m_{14}	1500	111	123	2	3	h14/p14	123	2.07	60	1	2.31	6	41	2.60	1.71
	m_{15}	1500	111	123	2	3	h15/p15	123	1.64	59	1	1.88	6	42	2.17	1.27
	m_{16}	1500	111	123	2	3	h16/p15	123	1.64	59	1	1.88	6	43	2.17	1.25
	m_{17}	1500	111	123	1	2	h17/p17	123	0.98	61	1	1.23	6	43	1.51	1.01
	m_{18}	1500	111	123	1	2	h18/p18	123	0.61	61	1	0.85	6	42	1.14	0.67
	m_{19}	1500	111	123	1	2	h19/p19	123	0.28	60	1	0.53	6	42	0.81	0.63
	m_{20}	1500	111	123	1	2	h20/p20	123	0	60	1	0.24	6	42	0.53	0.51

Table 4 Dual-Switch network topology/load-level A.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Dual-Switch Load-Level: A	m_1	1500	12.3	123	14	15	h1/p1	123	1.55	57	0	1.92	6	68	2.33	1.56
	m_2	1500	12.3	123	14	15	h2/p2	123	1.15	57	0	1.52	6	66	1.93	1.31
	m_3	1500	12.3	123	5	6	h3/p3	123	0.79	56	0	1.16	6	68	1.11	0.83
	m_4	1500	12.3	123	5	6	h4/p3	123	0.79	57	0	1.16	6	67	1.13	0.85
	m_5	1500	12.3	123	1	2	h5/p5	123	0	56	0	0.37	6	67	0.80	0.71

Table 5 Dual-Switch network topology/load-level B.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Dual-Switch Load-Level: B	m_1	1500	13.53	123	14	15	h1/p1	123	3.54	72	0	4.01	6	78	4.45	3.35
	m_2	1500	13.53	123	14	15	h2/p2	123	3.20	72	0	3.57	6	78	4.02	3.03
	m_3	1500	13.53	123	11	12	h3/p3	123	2.83	72	0	3.20	6	79	3.64	2.94
	m_4	1500	13.53	123	11	12	h4/p4	123	2.39	71	0	2.76	6	77	3.21	2.61
	m_5	1500	13.53	123	8	9	h5/p5	123	2.02	71	0	2.39	6	77	2.83	2.23
	m_6	1500	13.53	123	8	9	h6/p6	123	1.59	73	0	1.96	6	77	2.41	1.90
	m_7	1500	13.53	123	5	6	h7/p7	123	1.21	71	0	1.58	6	78	2.03	1.74
	m_8	1500	13.53	123	5	6	h8/p7	123	1.21	72	0	1.58	6	78	2.03	1.72
	m_9	1500	13.53	123	1	2	h9/p9	123	0.43	72	0	0.79	6	79	1.24	1.11
	m_{10}	1500	13.53	123	1	2	h10/p10	123	0	71	0	0.36	6	77	0.81	0.72

Table 6 Dual-Switch network topology/load-level C.

Experiment	Message Set Parameters						Response Time Analytical Values								Results	
	m_i	PS_i	IL_i	C_i	D_i	T_i	h_i/P_i	B_i	I_i	J_i	Q_i	v_i	TS_i	SD_i	$Calc.RT_i$	$Exp.RT_i$
	$m_\#$	$(Bytes)$	$(Mbps)$	(μs)	(ms)	(ms)	$h\#/p\#$	(μs)	(ms)	(μs)	$q\#$	(ms)	$\#$	(μs)	(ms)	(ms)
Dual-Switch Load-Level: C	m_1	1500	111	123	14	15	h1/p1	123	8.41	86	0	8.78	6	105	9.25	7.12
	m_2	1500	111	123	14	15	h2/p2	123	7.97	86	0	8.33	6	104	8.81	6.81
	m_3	1500	111	123	11	12	h3/p3	123	7.73	86	0	8.10	6	105	8.45	6.66
	m_4	1500	111	123	11	12	h4/p4	123	7.20	85	0	7.75	6	105	8.04	6.45
	m_5	1500	111	123	8	9	h5/p5	123	7.12	84	0	7.36	6	106	7.96	6.12
	m_6	1500	111	123	8	9	h6/p6	123	6.54	85	0	6.91	6	105	7.38	5.98
	m_7	1500	111	123	6	7	h7/p7	123	6.39	85	0	6.76	6	105	7.23	5.73
	m_8	1500	111	123	6	7	h8/p8	123	5.78	86	0	6.15	6	104	6.62	5.01
	m_9	1500	111	123	4	5	h9/p9	123	5.79	84	0	6.15	6	104	6.63	5.13
	m_{10}	1500	111	123	4	5	h10/p10	123	5.12	84	0	5.49	6	105	5.97	4.67
	m_{11}	1500	111	123	4	5	h11/p11	123	4.43	85	0	4.80	6	105	5.28	3.90
	m_{12}	1500	111	123	4	5	h12/p11	123	4.43	85	0	4.80	6	105	5.28	3.97
	m_{13}	1500	111	123	2	3	h13/p13	123	3.84	85	1	4.21	6	104	4.69	3.65
	m_{14}	1500	111	123	2	3	h14/p14	123	3.12	86	1	3.49	6	106	3.96	2.99
	m_{15}	1500	111	123	2	3	h15/p15	123	2.47	84	1	2.84	6	105	3.31	2.31
	m_{16}	1500	111	123	2	3	h16/p15	123	2.47	84	1	2.84	6	105	3.31	2.25
	m_{17}	1500	111	123	1	2	h17/p17	123	1.48	85	1	1.85	6	104	2.33	1.91
	m_{18}	1500	111	123	1	2	h18/p18	123	0.92	86	1	1.28	6	106	1.76	1.23
	m_{19}	1500	111	123	1	2	h19/p19	123	0.4	86	1	0.79	6	106	1.27	1.01
	m_{20}	1500	111	123	1	2	h20/p20	123	0	84	1	0.36	6	105	0.84	0.76