Development and optimization of an MTConnect based edge computing node for remote monitoring in cyber manufacturing systems

S M Nahian Al Sunny*, Xiaoqing "Frank" Liu, Md Rakib Shahriar
Department of Computer Science and Computer Engineering
University of Arkansas, Fayetteville, Arkansas, USA
Email: {smsunny*, frankliu, mrshahri}@uark.edu

***Abstract*—In recent years, MTConnect has emerged as a potential communication standard for cyber manufacturing (CM) domain by establishing remote monitoring of manufacturing processes through XML based data reporting structure and RESTful services. This paper presents the development and optimization of an MTConnect based edge computing node in service-oriented CM systems by utilizing data caching and processing at the edge. Unlike MTConnect agents, proposed MTConnect Edge Nodes (MENs) not only convert collected machining data to XML messages and host RESTful services, but also perform as an edge node by adopting a "*hold-until-changed*" approach for deciding which data to store and by keeping tracks of previously transmitted data to its clients to determine which data to transmit to whom and when. The primary objective is to minimize data storage requirements and cost while enabling rapid transmission and low bandwidth usage without increasing information loss. This paper describes the architecture of an MEN and its data caching and transmission strategies in details. Experiments were conducted in a CM testbed with three machine tools, raspberry pis hosting MENs and MTConnect agents, and two client applications to evaluate MEN's performance with respect to the conventional approach. Results showed 96.2 percent reduction in required storage size with 52.5 percent reduction in average communication latency and 99.5 percent reduction in average message size for proposed MEN in different manufacturing scenarios.**

***Index Terms*—MTConnect, edge computing, MTConnect Edge Node, data catching, performance optimization**

I. Introduction

Like many other aspects of today's world, the rapid growth of Internet has been reshaping manufacturing domain over past few decades. Emerging Internet-enabled technologies such as cloud computing, cyber-physical systems, Internet-of-Things (IoT), fog and edge computing etc. are opening up endless possibilities for advanced automation and manufacturing and also paving the path leading towards next industrial revolution, the so-called Industry 4.0 [1]. One of the primary objectives of Industry 4.0 is to enable seamless integration of web based client applications and physical machine tools for real-time data acquisition and analysis. In recent years, MTConnect has emerged as a potential communication standard for interacting with physical machines over the Internet and thus enabling development of cyber manufacturing (CM) systems [2]–[4]. MTConnect is an open-source standard that converts heterogeneous machining data into a common XML based format and transmits them to client applications via RESTful services [5]. However, manufacturing machine tools generates tremendous amount of data every second which are required to be transmitted at high-speed. This may lead to network traffic bottleneck and scalability issues, specially for large-scale CM systems managing hundreds of machines. Edge computing [6] can aid in this regard by storing and processing data locally before transmission. As the primary function of MTConnect is to transform data and provide a uniform interface for client applications, developing an MTConnect based edge computing node has the promise of resolving the aforementioned issues. But MTConnect is not specifically designed to be readily used for edge computing, as it does not offer specifications for localized data caching or processing in order to minimize cost and maximize efficiency, specially for resource constrained edge platforms. Therefore, adopting proper optimization strategies and evaluating performance through experimentation are required to develop such a scalable solution for CM systems.

In this paper we present the design and development of an MTConnect Edge Node (MEN) for CM systems. MEN is an extension of existing MTConnect agent program, as it not only converts collected machining data to XML messages and host MTConnect RESTful services, but also works as an edge node by efficiently caching data and performing computation locally to determine which data to be stored and transmitted to whom and when. To enhance its feasibility and performance while minimizing cost, we adopted unique optimization strategies for localized data caching and transmission using a "*hold-until-changed*" approach. Our primary objective is to minimize data storage requirements and cost while enabling high-speed transmission and low bandwidth usage without increasing information loss. Experiments were conducted in a CM testbed with three machine tools, raspberry pi devices, and two client applications - one deployed in a cloud server and another deployed as a web based smartphone app to evaluate proposed MEN's performance with respect to the conventional MTConnect agent. Results showed 96.24 percent reduction in required storage size with 52.5 percent reduction in average communication latency and 99.5 percent reduction in average message size for MEN in different manufacturing scenarios and thus providing proof of proposed concept.

Rest of the paper is organized as follows. Section II reviews related works and literatures. Architecture of the proposed MEN and optimization strategies are discussed in Section III.

978-1-7281-1086-8/20/$31.00 ©2020 IEEE
DOI 10.1109/ICFC49376.2020.00014

Section IV presents our experimental setup and analysis of results. Conclusions are given in Section V.

II. Related works

Machine Tool Connect or MTConnect was first introduced in 2008 with a view to developing a open-source and royalty-free standard for machining data collected from different types of machine tools, sensors, and other data sources during manufacturing processes [5], [7]. It is a read-only mechanism that uses a semantic ontology and XML based messaging structure to enable remote monitoring of machine tools through RESTful services. In recent years, MTConnect has been widely accepted and adopted for different types of manufacturing machines such as CNC (Computer Numeric Control) machines [8], robots [9], 3D printers [10], [11] etc. Moreover, MTConnect was recognized as a suitable candidate for establishing communication between machine tools and client applications hosted in web and cloud servers [12]. It was used in development of multiple web based machine monitoring systems for data collection, analysis, event notification, and machine health prediction [13], [14]. Lynn et al. evaluated three broad solutions for collecting data from machines using rapidly deployable MTConnect based monitoring systems [15]. Mazak Cooperation showcased a scalable, end-to-end Industrial IoT platform called 'Mazak SmartBox' which connects manufacturing equipment to a factory's network and management systems via MTConnect [16]. By extending MTConnect, an Internet-scale communication method titled MTComm was developed for performing both monitoring and operations of heterogeneous machine tools over the Internet [17]. Several researchers proposed frameworks for cyber-physical manufacturing systems using MTConnect [2]–[4], [18], [19]. However, there exist very few researches that utilized MTConnect to enable fog and edge computing in CM systems. Wu et al. proposed a fog computing based framework for process monitoring and prognosis in CM where machine data collected using different communication protocols including MTConnect was stored and processed in a local private cloud before transmitting to a central public cloud [20]. Parto et al. presented a MTConnect-compatible secured machine monitoring platform by integrating fog computing, cloud computing, and communication protocols [1]. They used microcontrollers as fog computing platforms where MTConnect XML messages were collected, converted to JSON format, and then transmitted to client applications using web APIs. We found no literature focusing on developing an MTConnect based edge computing node and adopting optimization techniques to enable scalable and efficient remote monitoring and enhance overall system performance.

III. Proposed methodology

A. Architecture of an MTConnect Edge Node

An MTConnect agent (MTCagent) works as a translator on an HTTP server. It is a software program that periodically acquires data either from MTConnect-compatible machine tools or from MTConnect adapters, which transform raw data

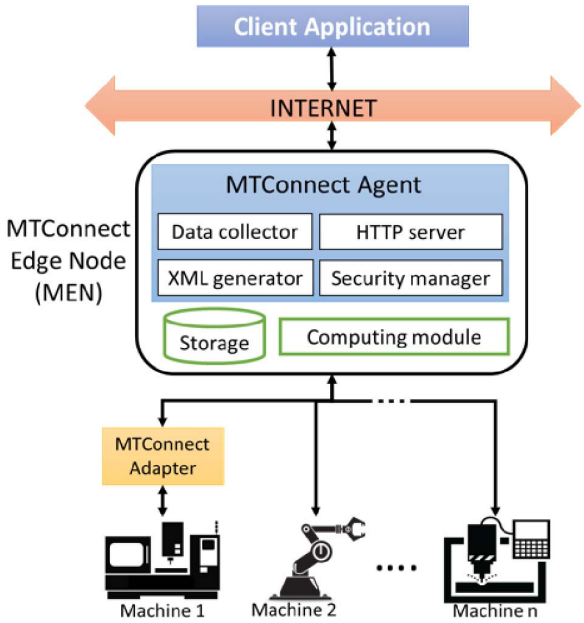

Fig. 1: Architecture of proposed MEN

from heterogeneous machine tools and sensors into simple text format understandable by MTCagents, and converts them to XML messages using MTConnect standard schemas and its semantic ontology [7], [11]. In MTConnect ontology, each machine tool is considered as a *device* and its parts or associated sensors are termed as *components*. A *dataitem* represents any piece of data generated by a *device* or its *components*. MTCagent also hosts RESTful services on an HTTP server to receive requests from client applications and responds with corresponding XML messages. There are four MTConnect services – `probe` which provides structural and configuration information of a machine tool and its associated *dataitems*, `current` that responds with most recent values of *dataitems*, `sample` which gives values of *dataitems* over a certain time period, and `error` to report invalid requests and other errors. All services are requested using HTTP GET method, except `error` which is generated by the agent upon an error occurrence. MTCagent usually has a buffer storage to temporarily store acquired data before transmission. It may also include a security manager for imposing access control mechanisms, malicious attack prevention etc. One MTCagent can support one or multiple *device(s)*.

To introduce edge computing using MTConnect in a CM system, we proposed to expand MTCagent's capabilities beyond data transformation and service hosting and convert it to an MTConnect Edge Node (MEN). Figure 1 illustrates the architecture of our proposed MEN. In addition to the modules of a typical MTCagent (data collector, XML generator, HTTP server, and security manager), an MEN also includes a local storage and a computing module. The data caching process of this storage is fundamentally different from a typical MTCagent's buffer storage, which is discussed in next section. The computing module performs additional computations to

optimize MEN's overall performance by analyzing incoming data from machine tools to determine which data to be stored locally and how, and also requests from client applications to determine which data to be transmitted and how. Its primary focus is to reduce data transmission latency and bandwidth usage without increasing information loss and also to enhance scalability of overall system by offloading tasks at the edge. Like an MTCagent, an MEN can support multiple machine tools and communicate with multiple clients over the Internet using HTTP. An MEN is designed as a standalone software program that is easily deployable in constrained embedded platforms. It can even be embedded into a machine's controller unit. The size of local storage and processing power of computing module depends on the hardware.

B. Optimization strategies

As MENs are primarily designed for constrained edge devices, it is necessary to adopt optimization strategies in order to reduce storage and processing power requirements and speed up data transmission mechanisms. In this section, we discussed two computing strategies for proposed MEN – data caching strategy and data transmission strategy.

1) Data caching at the edge: As described before, an MTCagent usually contains a buffer storage to temporarily store acquired data. It is a circular buffer storage meaning data are stored sequentially and when the storage is full, most recent data entry overwrites the oldest one [7]. Also MTCagent stores the last value for each *dataitem*, even if it is no longer in the buffer or the buffer is empty. As a machine tool usually generates significant amount of data at high frequency and MTCagent stores every data collected from it, the buffer storage can fill up very fast. Therefore, the buffer storage should be large enough to allow for adequate space required to minimize information loss. Typically MTCagents are deployed in desktop computers or servers, which contains enough storage to fulfill this requirement. However, for a constrained edge computing device, this can become a significant issue as large storage requirement leads to increased cost. So it is necessary to optimize data catching strategy in a MEN to minimize storage size and cost without losing information.

Therefore, the proposed MEN adopts a *"hold-until-changed"* approach where only unique *dataitem* values are stored in a local storage along with associated timestamp and sequence number while discarding repeating or redundant values. Algorithm 1 illustrates our data caching procedure. At the beginning, MEN uses the `probe` document and creates an object for each *dataitem* to hold associated attributes, such as id, name, type etc., and values. When MEN acquires data from machine tools for the first time, it stores all *dataitem* values in a local database with id, timestamp, and sequence number (which is 1 at this point). When next batch of data arrives, unlike MTCagent which just stores these new data without any processing, MEN's computing module checks whether the new value of a *dataitem* is different from its previously stored value or not. If the new value is same as the old one, it is discarded. Otherwise, a new entry is created in the database with this new

Algorithm 1 data caching procedure

Require: new data available from a *device*

$device.currentSeq \leftarrow device.currentSeq + 1$
$timestamp \leftarrow$ current time
$hasNewEntry \leftarrow$ `False`
for each $item$ object in $dataitemsList$ **do**
  if ($item.lastStoredData =$ `NULL` or
    ($item.lastStoredData \neq$ `NULL` and
    $item.currentData \neq item.lastStoredData$))
  then
    $hasNewEntry \leftarrow$ `True`
    Add a new entry to database with values of
      ($device.currentSeq, timestamp, item.id,$
      $item.currentData$)
    $item.lastStoredData \leftarrow item.currentData$
  end if
end for
if $hasNewEntry =$ `False` **then**
  $device.currentSeq \leftarrow device.currentSeq - 1$
else
  commit database
end if

dataitem value, latest timestamp, and incremented sequence number. If no *dataitem* has new value, then the whole batch is ignored and current sequence number remains the same. This way MEN's local database only contains unique data for all *dataitems*. Sequence numbers and timestamps are used to retrieve *dataitem* values at a certain time. The time interval between two consecutive sequences indicate that the *dataitem* had the same value for that particular time period. Using this method, information loss is minimized as all unique values are stored and timestamped, yet the amount of data stored for a given time and thus the minimum size of local storage are considerably smaller than those of a conventional MTCagent.

2) Data transmission strategy: Usually when an MTCagent receives a `current` request, it immediately responds with most recent *dataitem* values. It also allows client applications to perform queries with specific parameters, such as *dataitem* id, sequence number, timestamp etc. Also using `sample` service, a client application can request for data of a certain time interval, e.g. from sequence 20 to sequence 85. Therefore, it is upto client applications to determine how to send requests to an MTCagent. If a client wants to ensure that it receives all available data, it has to keep track of what data sequences it previously received and also determine which sequences are to be requested and how often. For a large-scale system, this may lead to scalability issue. Also this may lead to significant data loss for simple client applications. In one experiment, we developed a simple remote monitoring web application which was only capable of sending consecutive `current` requests to an MTCagent and displaying data extracted from response messages. The results showed considerable gaps between sequence numbers of consecutive responses, ranging

Algorithm 2 Prepare response for incoming request

Require: receive a new `current` request for a *device*
1: $timestamp \leftarrow$ current time
2: $clientAddr \leftarrow$ address/url of client
3: Fetch $clientDict$ containing list of previous clients
4: Create an empty dictionary named $itemsToAdd$
5: **if** $clientAddr$ exists in $clientDict$ **then**
6: $responseSeq \leftarrow clientDict[clientAddress] + 1$
7: **if** $responseSeq \leq device.currentSeq$ **then**
8: $results \leftarrow$ all entries in database WHERE
9: $sequence = currentSeq$
10: **for** each $item$ in $results$ **do**
11: add $(item.id, item.data)$ to $itemsToAdd$
12: **end for**
13: **end if**
14: **else**
15: $responseSeq \leftarrow device.currentSeq$
16: **for** each $item$ in $dataitemsList$ **do**
17: add $(item.id, item.currentData)$ to
18: $itemsToAdd$
19: **end for**
20: **end if**
21: **if** $itemsToAdd$ is empty **then**
22: Send `HTTP 204` response
23: **else**
24: **for** each $item$ in $itemsToAdd$ **do**
25: Call function $createXmlElement$ with
26: $(currentSeq, timestamp, item.id, item.data)$
27: **end for**
28: Send `HTTP 200` response with XML message
29: **end if**
30: add/update $(clientAddr, currentSeq)$ pair in $clientDict$

from tens to thousands, as data acquisition and conversion rate of the agent is faster than transmission rate over the Internet.

Performing computation at the edge of a factory floor environment can assist in overcoming these issues by offloading simple repetitive tasks to MTCagents. Therefore, our proposed MEN keeps track of its clients and determines which and how much data should be sent to a client. Algorithm 2 demonstrates the procedure used for preparing responses for a client. Alongside its database, MEN maintains a key-value pair based dictionary which contains ip addresses or URLs of all client applications it has communicated with and last sequence numbers associated with them. When MEN receives an incoming `current` request, its computing module uses the client's address to check if this client has any entry in the client dictionary. If no pair is found, then it identifies the client as a new one. Then an XML response message is created with most recent values of all *dataitems*. As all *dataitem* objects holds their most recent values, this particular step does not require fetching data from local database. The XML message is timestamped and sequenced with corresponding values at that point. It also contains additional information such as header elements, attributes of *dataitems* etc., which can be acquired from the `probe` document. Then MEN sends an `HTTP 200` response with the generated XML message and adds an entry to its client dictionary with client's address as key and latest sequence number used to create the response as value. When MEN receives a new request from the same client, it finds its entry in the client dictionary and retrieves associated sequence number. Then it tries to find *dataitems* associated with the next sequence number. If no entry is found for this new sequence, it means the machine has not generated any new data since the client's last query. So MEN sends an `HTTP 204` response with no content. In case where MEN finds relevant database entries, it fetches those *dataitem* entries and create an XML element for each entry using id and stored value of the *dataitem*, most recent timestamp and sequence number, and other associated information. As mentioned in the previous section, the local database only stores *dataitems* with changed values, so the response XML only contains new data that were not previously transmitted to the client. After XML generation is completed, MEN sends an `HTTP 200` response with it and updates this client's entry in the client dictionary with new sequence number. The client dictionary has a time-out functionality which removes an entry when the stored sequence is no longer available in local database indicating that client has not communicated in a long time.

Using this strategy, MEN reduces bandwidth usage by preventing transmission of repeating or redundant *dataitem* values. It enables faster data transmission and processing as `HTTP 204` responses has no message body and thus can be processed much quickly. Also not all *dataitems* change values in every data acquisition cycle, specially in idle states, so the `HTTP 200` responses of proposed method contains fewer elements in average that those of a typical MTCagent. Furthermore, MEN keeps tracks of its clients and thus enhance scalability of CM systems through edge computing.

IV. Experimentation and Evaluation

A. Experimental Setup

To evaluate the performance of proposed MEN, experiments were conducted in a cyber manufacturing testbed with three machine tools, as shown in Figure 2. Ultimaker 2 and Bukito are 3D printers and X-Carve is a CNC drilling machine. Each machine tool was connected to a Raspberry Pi (RPi) 3, which contained both MEN and conventional MTCagent programs developed in Python. SQLite3 databases were used as local storages. As none of these machines were MTConnect compatible, MTconnect adapter programs were developed and deployed in corresponding RPis. Although one RPi had sufficient storage and processing capacity to support all three machines, individual RPis were used to simulate a manufacturing environment with multiple MTCagents. Ultimaker 2, Bukito, and X-Carve had 15, 11, and 7 *dataitems* respectively. A client application was developed and deployed in a virtual machine (cloud server) in the university network. Also another web based client application was developed for Android smartphones. Both applications were designed to continuously

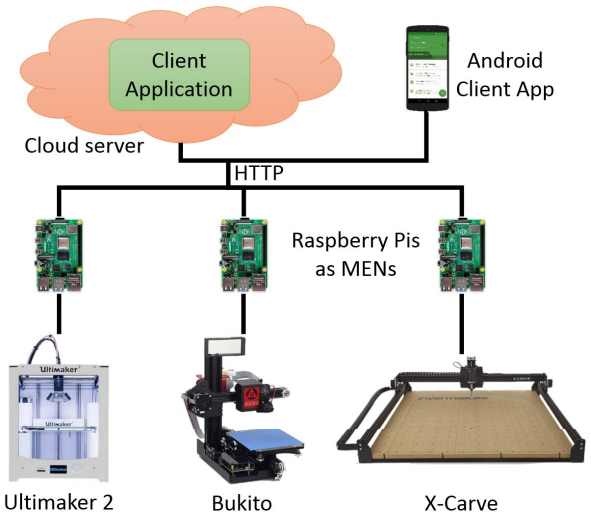

Fig. 2: Experimental setup

send HTTP GET requests to RPis, store and analyze response messages, display extracted data values, and calculate response time and message size. Data were collected using both conventional and proposed methods from all machine tools in different manufacturing scenarios. At first, only one machine was kept active at a time and clients sent continuous requests to its RPi in both idle and busy (performing a drilling or printing operation) states. Duration of operations done by Ultimaker 2, Bukito, and X-Carve were about 5, 13, and 2 minutes respectively. Then data were collected in a 'Combined' state, where all three machine were kept active and in busy state and clients queried machines sequentially (one-at-a-time), e.g. Ultimaker 2 → X-Carve → Bukito → Ultimaker 2 and so on.

B. Results and analysis

Performance of MEN was evaluated in three categories – response time (RT) depicting the time between client sending a `current` request and getting response back, minimum size of local storage, and bandwidth usage or size of response messages. Table I and Figure 3 presents the comparison of average RT for each machines in both states calculated from 10000 consecutive requests. In each case, MEN provided about 50-55 percent reduction in RT compared to conventional method, 52.5 percent in average. In [15], average time to complete an MTConnect HTTP GET request using conventional method

TABLE I: Average response time in milliseconds(ms)

Machine name	Machine state	Avg. RT for conventional method	Avg. RT for proposed method	RT reduction (%)
X-Carve	Idle State	7.365 ± 1.2	3.813 ± 0.9	**48.228**
	Busy State	7.756 ± 1.3	4.002 ± 1.1	**48.401**
Ultimaker 2	Idle State	13.053 ± 2.7	6.438 ± 2.4	**50.678**
	Busy State	16.057 ± 2.9	7.237 ± 2.6	**54.929**
Bukito	Idle State	12.931 ± 1.3	5.706 ± 1.7	**55.874**
	Busy State	14.608 ± 2.9	6.255 ± 2.5	**57.181**

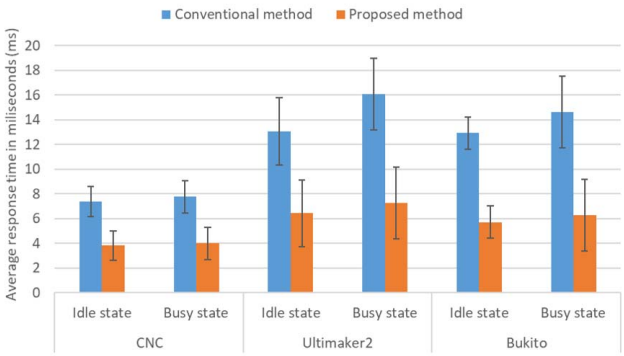

Fig. 3: Comparison of average response time (in milliseconds)

was 4.2 ms in a setup where MTCagent and its client were close and part of the same local area network (LAN). In our setup, all RPIs were given global static ip addresses and client applications communicated with MTCagents and MENs over the Internet. Still the average RT of MENs was very close (5.6 ms). As HTTP 204 responses contain no message body, clients can process such responses quicker than HTTP 200 responses. Also this reduces the amount of data transferred or bandwidth used, which is supported by Table II showing total local storage size required and total number and size of messages transferred during an operation in individual and combined scenarios. For combined state, data were collected from all machines, but results were calculated only for Ultimaker 2, to understand how collecting data sequentially from additional machines affects performance of an agent. As MEN only stores unique *dataitem* values, required size of local storage to store all available data during an operation was more than 90 percent less in all cases (96.24 percent in average), compared to a conventional MTCagent which stores all incoming data. This also shows that the minimum storage size required to ensure minimal information loss is significantly smaller for proposed MEN, which is very crucial for constrained edge devices. Table II also shows significant decrease in amount of data (bytes) being transmitted by MENs during an operation – 85.32 percent in average. Total number of messages were also reduced in proposed method, except in combined state which showed a 19.32 percent increase. However, this increase actually indicates performance improvement, as it shows that proposed method was able to extract more information in combined state than conventional method. Typical MTCagent has to process full-size responses from all three machines before querying the first machine again. This time delay is much smaller for MEN, as its responses are shorter and in some cases empty. Therefore, in a given time, MEN allows clients to query one machine in a multi-machine factory floor more frequently than an MTCagent and thus receive more messages leading to better information gain.

As depicted in Table III, the average size of response messages also showed a significant reduction in proposed method – 99.5 percent in average. This is expected though, as MEN only sends changed values and thus generates smaller messages less frequently than a typical MTCagent, specially in

TABLE II: Comparison of total size of local database and transferred messages and total no. of messages

		X-carve	Bukito	Ultimaker 2	Combined
Total size of database	Conven.	3147732	13085696	2787263	1171046
	Proposed	8616	368640	102672	96635
	Reduction	**99.726**	**96.316**	**97.183**	**91.748**
Total bytes sent	Conven.	47739494	198461787	16489780	6928051
	Proposed	111439	47679347	2854141	2686327
	Reduction	**99.767**	**97.598**	**82.691**	**61.225**
Total No. of messages	Conven.	17449	50728	4201	1765
	Proposed	115	4699	2381	2106
	Reduction	**99.341**	**90.737**	**43.323**	**-19.32**

TABLE III: Average size of response message in bytes

Machine name	Machine state	Avg. size for conventional method	Avg. size for proposed method	Size reduction (%)
X-Carve	Idle State	2721	1.7	**99.938**
	Busy State	2725.677	4.047	**99.852**
Ultimaker 2	Idle State	3896	6.847	**99.824**
	Busy State	3923.496	64.616	**98.353**
Bukito	Idle State	3918	5.646	**99.856**
	Busy State	3906.316	33.212	**99.149**

idle states. For instance, 3D printers have temperature sensors for extruders and heatbeds, which change values with room temperature. All other *dataitem* values remain unchanged in idle states. Even in busy state, not all *dataitems* generate new values in every clock cycle. For example, the "AVAILABILITY" *dataitem* of a machine only changes its value at the start and end of an operation, so is only required to be transmitted twice during a manufacturing process. Conventional method still reports these redundant unchanged values to client, which leads to bandwidth wastage.

V. Conclusion

As modern manufacturing machines are producing more and more data, it is high time to adopt edge computing strategies to offload iterative small-scale computations to network edges and reduce load on cloud and web applications. The proposed MEN has the potential to become a suitable and efficient edge computing solution for cyber manufacturing systems. Our proposed optimization strategies for data caching and transmission significantly improve MTConnect's performance, efficiency, and feasibility for edge computing. As MENs are software programs capable of rapid data transmission with low storage and processing power requirements, it can be easily deployable in low-cost plug-n-play embedded hardware, even be embedded within a machine's controller unit, and thus enhance overall system scalability. Our experimental results demonstrated MEN's superiority over traditional MTConnect agent in communication latency, bandwidth usage, and local storage size requirement. Future works may include further enhancement of MENs by using high-performance hardware platforms such as FPGAs and introducing advanced edge based data processing mechanisms such as big data analysis, artificial intelligence, deep and machine learning etc.

Acknowledgment

This was partially supported by NSF Grant CMMI 1551448 entitled "EAGER/Cybermanufacturing: Architecture and Protocols for Scalable Cyber-Physical Manufacturing Systems".

References

[1] M. Parto, M. Dinar, and T. Kurfess, "An mtconnect-compatible platform for secured machine monitoring through integration of fog computing, cloud computing, and communication protocols," in *Proceedings of the International Symposium on Flexible Automation*, 2018, pp. 329–336.

[2] S. M. N. A. Sunny, X. F. Liu, and M. R. Shahriar, "Communication method for manufacturing services in a cyber–physical manufacturing cloud," *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 7, pp. 636–652, 2018.

[3] A. J. Álvares, L. E. S. d. Oliveira, and J. C. E. Ferreira, "Development of a cyber-physical framework for monitoring and teleoperation of a cnc lathe based on mtconnect and opc protocols," *Int. J. of Computer Integrated Manuf.*, vol. 31, no. 11, pp. 1049–1066, 2018.

[4] C. Liu, X. Xu, Q. Peng, and Z. Zhou, "Mtconnect-based cyber-physical machine tool: a case study," *Procedia CIRP*, vol. 72, pp. 492–497, 2018.

[5] A. Vijayaraghavan, W. Sobel, A. Fox, D. Dornfeld, and P. Warndorf, "Improving machine tool interoperability using standardized interface protocols: Mt connect," 2008.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] M. Institue, *MTConnect Standard Specification - Version 1.4.0*, 2019 (accessed November 3, 2019). [Online]. Available: http://mtconnect.org/

[8] S. Chen, C. Yin, and X. Li, "Implementation of mtconnect in machine monitoring system for cncs," in *2017 5th International Conference on Enterprise Systems (ES)*. IEEE, 2017, pp. 70–75.

[9] M. Robinson, *Cost Effective Coordinated and Cooperative Robotics Enabled by Open Technologies*, NIST, 2019 (accessed November 3, 2019). [Online]. Available: https://www.nist.gov/

[10] E. Rodriguez and A. J. Alvares, "Implementation of the step-nc and mtconnect standards for additive manufacturing," in *Anais do X Congresso Brasileiro de Engenharia de Fabricaçao, ABCM*, 2019, pp. 1–5.

[11] X. F. Liu, S. M. N. A. Sunny, M. R. Shahriar, M. C. Leu, M. Cheng, and L. Hu, "Implementation of mtconnect for open source 3d printers in cyber physical manufacturing cloud," in *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. ASME, 2016, p. V01AT02A019.

[12] X. Xu, "From cloud computing to cloud manufacturing," *Robotics and computer-integrated manufacturing*, vol. 28, no. 1, pp. 75–86, 2012.

[13] S. Atluru and A. Deshpande, "Data to information: can mtconnect deliver the promise," *Transactions of NAMRI/SME*, vol. 37, no. 2009, pp. 197–204, 2009.

[14] B. Edrington, B. Zhao, A. Hansel, M. Mori, and M. Fujishima, "Machine monitoring system based on mtconnect technology," *Procedia Cirp*, vol. 22, pp. 92–97, 2014.

[15] R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, "Rapidly deployable mtconnect-based machine tool monitoring systems," in *Proceedings of the 12th ASME Manufacturing Science and Engineering Conference (MSEC)*, 2017.

[16] M. Corporation, *Mazak SmartBox*, 2017 (accessed November 3, 2019). [Online]. Available: https://www.mazakusa.com/

[17] S. M. N. A. Sunny, X. F. Liu, and M. R. Shahriar, "Mtcomm: A semantic ontology based internet scale communication method of manufacturing services in a cyber-physical manufacturing cloud," in *2017 IEEE International Congress on Internet of Things*. IEEE, 2017, pp. 121–128.

[18] L. Wang, R. Gao, and I. Ragai, "An integrated cyber-physical system for cloud manufacturing," in *ASME 2014 International Manufacturing Science and Engineering Conference collocated with the JSME 2014 International Conference on Materials and Processing and the 42nd North American Manufacturing Research Conference*. ASME, 2014.

[19] C. Liu, H. Vengayil, Y. Lu, and X. Xu, "A cyber-physical machine tools platform using opc ua and mtconnect," *Journal of Manufacturing Systems*, vol. 51, pp. 61–74, 2019.

[20] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, 2017.