BACGRAPH: Automatic Extraction of Object Relationships in the BACnet Protocol

Herson Esquivel-Vargas
University of Twente
Enschede, The Netherlands
h.esquivelvargas@utwente.nl

Marco Caselli
Siemens AG
Munich, Germany
marco.caselli@siemens.com

Andreas Peter
University of Twente
Enschede, The Netherlands
a.peter@utwente.nl

***Abstract*—This work presents BACGRAPH, a tool that extracts relationships among configuration parameters of Building Automation and Control Systems (BACSs) implemented using the BACnet protocol (ISO 16484-5). BACnet models these configuration parameters as *object* data structures comprised of multiple properties, some of which contain references to other objects. Given the regular exchange of objects among devices, we leverage these explicit references to build a graph of BACnet objects exclusively from network traffic. We tested BACGRAPH using traffic collected from a real building located at the University of Twente. After analyzing 66.8 hours of traffic, the resulting graph is comprised of 13,733 nodes and 3,169 edges. Such a graph improves the *system visibility* that BACS administrators have over their infrastructure, which is crucial for troubleshooting and security.**

I. INTRODUCTION

Building Automation and Control Systems (BACS) lie at the core of smart-buildings. BACSs are control systems that use sensors to collect information from the building and also have the capability to influence the physical environment through actuators (e.g., lights, pumps, valves, etc.) [1]. BACSs control building services such as ventilation, heating, illumination, and many more. Critical buildings such as airports and hospitals are fully dependent on automated building services to remain operational [2]. Thus, it is important to provide BACS administrators with tools that help in the management process.

System visibility is a crucial aspect to successfully operate BACSs. It allows BACS administrators to monitor what happens in their infrastructure. BACSs visibility is typically achieved using Supervisory Control and Data Acquisition (SCADA) software, similar to that used in Industrial Control Systems (ICSs). System visibility has long been acknowledged by the ICSs community as an essential capability for troubleshooting and security [3], [4].

BACSs can be implemented using a variety of industrial communication protocols. Some of them, however, are specifically tailored to meet the needs of the BACS domain. For example, some BACS protocols have built-in functionality that make it easier to handle lighting schedules, elevators, escalators, and other common building services. In this work, we focus on the BACnet protocol, one of the most popular protocols in the BACSs domain [5].

This work is partially funded by the Costa Rica Institute of Technology and the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 830927.

BACnet Protocol Overview. The Building Automation and Control network (BACnet) protocol is an ASHRAE, ANSI, and ISO 16484-5 standard [6]. BACnet spans through four layers of a collapsed architecture that corresponds to the physical, data link, network, and application layers of the OSI model. Buildings automated using this protocol feature devices capable of communicating with each other following the rules defined in the BACnet standard.

The application layer of the BACnet protocol stores information in predefined variables called *BACnet properties*, which are in turn, encapsulated in predefined data structures called *BACnet objects*. For example, sensor readings are stored in a property called *present-value*, which could be within an *Analog-Input* object if the sensor provides analog values, or within a *Binary-Input* object if the sensor provides binary readings. By aggregating different subsets of properties, BACnet has 60 built-in objects that represent very high level abstractions such as the *Lighting-Output* object, the *Escalator* object, and the *Elevator* object. Moreover, the BACnet standard allows manufacturers to include vendor-specific object types in their devices. However, BACnet devices are required to implement only a subset of these object types, depending on their profile (e.g., smart-sensor, controller, router, etc.). BACnet devices typically store many instances of the supported object types. For that reason, each instance has a unique numeric identifier. BACnet devices exchange information, in the form of BACnet objects, on a regular basis. This information exchange through the network allows the interoperability among different subsystems.

Most BACnet objects have properties that reference other BACnet objects. Those explicit pointers establish specific relationships among BACnet objects. However, the instantiation of BACnet objects and their relationships is typically hidden behind the programming environment used by BACS administrators. Thus, the visibility of BACS administrators at the application protocol level is limited.

Summary of our Contribution. We leverage (1) the rich semantics of the application layer of the BACnet protocol; and (2) the regular exchange of BACnet objects among devices, to write *a software that reads BACnet network traffic and builds a graph data structure of BACnet objects and their*

978-1-6654-3566-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSN-S52858.2021.00029

relationships. We call our software BACGRAPH. The graph produced by BACGRAPH enhances the overall visibility of the system since it provides BACS administrators with a view unavailable in SCADA software. In practice, the graph-based view of the BACSs can help administrators to track the root cause of problems during troubleshooting.

BACGRAPH relies exclusively on standard definitions of the BACnet protocol. For that reason, it works on all BACnet networks regardless of the devices' brands and models. BACGRAPH's source code is freely available under the GNU/GPLv3 license (available at https://gitlab.com/bacgraph1/bacgraph).

Related Work. A variety of tools available on the market offer visualization capabilities for BACnet networks. However, the graphical capabilities in many of them is limited to a windows-based Graphical User Interface (GUI) that shows a *text-based* list of BACnet devices and their corresponding objects [7], [8], [9]. These tools enable the manipulation of BACnet objects and properties using their GUI. None of them, however, offers a *graphics-based* visualization of network components such as BACnet devices or BACnet objects.

Visual BACnet is a tool that offers more advanced graphical visualizations but mostly about network traffic statistics [10]. It shows plots about network message types (e.g., broadcast, unicast), histograms of the devices' traffic, number of packets observed over time, and others. However, this tool also lacks a *graphics-based* visualization of network components. A distinctive feature of *Visual BACnet* is that it works *off-line* by analyzing previously captured network samples.

GRASSMARLIN is a passive network mapper developed by the National Security Agency [11]. It analyzes network traffic of diverse industrial control protocols, including BACnet. It is capable to illustrate the network topology of physical devices but does not provide visualizations of data at the BACnet application layer.

To the best of our knowledge, BACGRAPH is the first tool that takes advantage of BACnet object references to enhance the visibility of BACnet networks.

II. BACGRAPH IMPLEMENTATION DETAILS

BACGRAPH is written in Python. BACGRAPH reads BACnet network traffic captures using the *pyshark* library to dissect the packets. It extracts information about BACnet objects – and their references – and stores them in a Neo4j database [12]. Neo4j is a NoSQL database which uses instead the *Cypher Query Language*. Using this language, it is possible to extract specific parts of the graph. Moreover, the database clients, available as desktop and web applications, have built-in graph visualization capabilities. The handling of user queries and the creation of graph visualizations are not part of BACGRAPH.

All BACnet object instances observed in the traffic become nodes of the graph. Additionally, BACGRAPH looks for specific properties in BACnet objects of certain types, which might contain references to other object instances. These references are used to create the edges of the graph. A deterministic Finite State Machine (FSM) is implemented within BACGRAPH to keep track of the *current* object instance (source of the edge) and the *referenced* object instance (destination of the edge), if any. An excerpt of the FSM is shown in Fig. 1.

A walk through the FSM starts with every network packet. If a packet has data to be analyzed, the FSM transitions to the *(R)ead* state. Once in the *R* state, the FSM reads the packet in sequential order until it finds a BACnet object instance. Such an object could be of any type, either standard or vendor-specific. However, instances of most object types rarely reference other object instances although they have the properties to do so. BACGRAPH focuses on 5 object types whose instances often contain references. Those object types are: *(S)*chedule, *(T)*rend-Log, *(L)*oop, *(E)*vent-enrollment, and *(N)*otification-class. Moreover, BACGRAPH also stores in the database all *(O)*ther BACnet objects observed. The FSM defines a state for each of those cases. From all object-related states, the FSM can transition to other states triggered by the occurrence of certain BACnet properties. Those properties contain references to other object instances. Once the information about those properties has been extracted, the FSM returns to the properties' corresponding object state. These transitions (depicted in Fig. 1 with dashed lines) not only change the FSM state, but also trigger the method that stores previously unknown information in the database . For the sake of readability, Fig. 1 omits *states* that look for additional information about BACnet objects to enrich the graph and *transitions* that go from the object-related states back to the *R* state. Next, we describe in detail the states related to BACnet objects and their properties.

Schedule. This object type models a schedule of possibly recurrent activities that must be executed by the BACS. For example, turning the lights on at a certain time during work days. Object instances of this type have one property that might contain a list of pointers to other objects that must be manipulated to accomplish the intended task. Such a property is called *list-of-object-property-references* (loopr). If this property is found, the FSM transitions to the S_1 state to look for details about the referenced objects.

Trend-Log. "A Trend-Log object monitors a property of a referenced object and, when predefined conditions are met, saves ("logs") the value of the property and a timestamp in an internal log buffer for subsequent retrieval" [6]. The referenced object is stored in a property called *Log-DeviceObjectProperty*. If it is found, then the FSM triggers the *ldop* transition to T_1.

Loop. The Loop object allows the implementation of the closed control loop programming pattern [13]. A loop is comprised of tree basic components: a sensor, a setpoint, and an actuator. It operates by continuously comparing current sensor readings against the predefined setpoint and aims to minimize the difference by influencing the physical process using the actuator. The physical process could be e.g., temperature control, illumination control, etc. The Loop object type contains 3 properties that reference its basic components: the *controlled-variable-reference* (cvr) that points to the sensor; the *manipulated-variable-reference* (mvr) that points to the actuator; and the *setpoint-reference* (sr) that points to the setpoint.

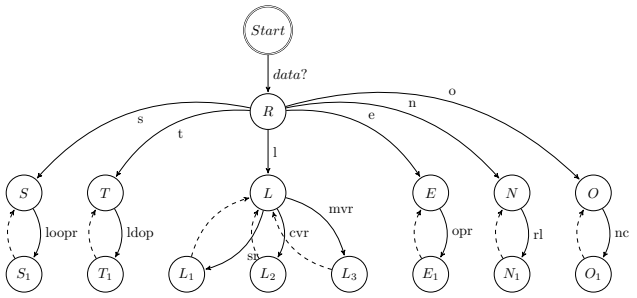


Figure 1. Simplified illustration of the Finite State Machine (FSM). It contains the most important states and transitions to identify BACnet objects and references in network traffic captures. We highlight with dashed lines the transitions that cause previously unknown objects and references to be stored in the database.

Event-Enrollment. In BACnet jargon, *events* typically refer to changes in the values stored in BACnet properties. An Event-Enrollment instance defines what constitutes an event on arbitrary objects (e.g., a temperature increase of $5°C$). The reference to the object and property being monitored by the Event-Enrollment object is stored in a property called *object-property-reference* (opr). If this property is found in an Event-Enrollment instance, the FSM transitions from the E state to the E_1 state.

Notification-Class. This object provides a standardized way to distribute event notifications within BACnet systems. Whenever an event occurs, a Notification-Class object is transmitted to a predefined list of devices stored in a property called *recipient-list* (rl). Conveniently, BACnet has a standard *Device* object type to be used, among other purposes, to specify recipients of Notification-Class objects. Typically, among those recipients is the SCADA software available in the network.

Others. Object instances that are not of the Schedule, Trend-Log, Loop, Event-Enrollment, or Notification-Class types are also stored in the database. In this case, the FSM looks for a property called *notification-class* (we write it in lowercase to distinguish it from the homonym BACnet object) which, as expected, references Notification-class objects.

III. Using BACgraph

We collect BACnet traffic from the University of Twente BACS. The network tap is located in the core switch of one of the buildings, which allows to observe the traffic exchanged among BACnet devices in this particular building. There are 4 general purpose BACS controllers in this building. The building hosts 375 employees in 252 rooms. The time span of the traffic collection is 66.8 hours. The traffic collection procedure is passive; we do not introduce any messages to the network during the collection period. The size of our dataset is 9.8 GB.

We executed BACgraph in a PC that features an AMD Ryzen 5 1400 Quad-Core Processor @3.2 GHz and 8 GB of RAM. It took BACgraph 40 minutes (wall-clock time) to analyze the traffic. The resulting graph is comprised of 13,733 nodes (BACnet objects) and 3,169 edges (BACnet object relationships). A break down of the object types observed in the traffic is shown in Table I. The 5 specific object types for which the FSM has a dedicated state add up to 2,302 object instances whereas the vast majority are other types of objects (11,431 instances). Out of those 11,431 instances, 9,918 belong to 12 standard object types and 1,513 to 22 vendor-specific object types. Since the definition of vendor-specific objects is out of the standard, the current BACgraph implementation does not look for any properties in them. However, vendor-specific objects are stored in the database as well, typically as isolated nodes.

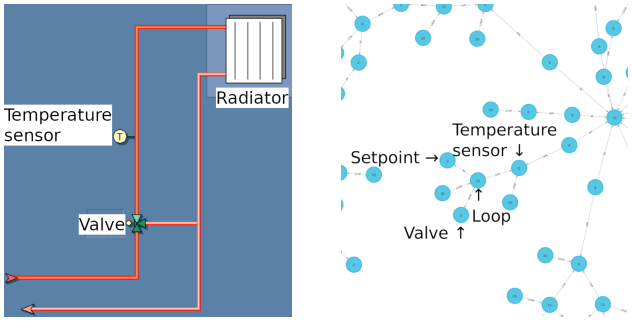
Table I
Observed BACnet object types and their number of instances.

BACnet Object Type	Count	BACnet Object Type	Count
Schedule	105	Event-Enrollment	350
Trend-Log	1,374	Notification-Class	379
Loop	94	Others	11,431

To evaluate BACgraph's success in discovering object instances from network traffic, we look at engineering documents listing the object instances stored in BACnet controllers. According to those documents, there are 11,293 object instances stored in the building's controllers. There are two main reasons that explain why BACgraph found a larger amount of objects in the network. The first reason is that the engineering documents only include object instances of standard types and not of proprietary object types. The second reason is that there are other BACnet devices in the network that are not controllers (e.g., routers and other device profiles), all of which exchange BACnet objects in the network on a regular basis. Although we do not have an absolute ground truth about how many object instances are stored in the building BACS, our results suggest that the graph comprises most of the expected nodes. To ensure graph completeness in terms of object instances, an active elicitation of BACnet objects from all devices is needed.

An important feature of the automatically generated graph is to remain up-to-date. Whenever new objects and references are created, they will likely be observed in the network and added to the database. In the case of deleted objects and references, such changes must also be reflected in the graph. To do so, BACgraph keeps two timestamps for each node and reference. The first timestamp records the creation time i.e., the first time that it was observed; and the second timestamp records the last time that it was observed. The second timestamp can be used to determine when a node or reference is stale and should, therefore, be removed from the graph.

SCADA software provides a *physical* oriented view of the BACS. Fig. 2a shows a heating module as seen on the SCADA software. The radiator shown takes hot water from the pipe on top and, after heating up the environment, exhausts colder water through the pipe at the bottom. Moreover, there is a valve that mixes hot water coming from the building's boiler (not shown in the picture) and the radiator's exhaust pipe. The proportion of hot and cold water in the mix is determined by



(a) View from the SCADA software.

(b) View from the graph database client.

Figure 2. The same heating module as seen on the SCADA software (a) and the BACnet objects graph (b).

a pre-established setpoint. The temperature sensor shown in the figure measures whether the current water mix is at the setpoint value. Otherwise, the valve has to be adjusted.

BACnet object graphs provide a *configuration* oriented view of the BACS. Fig. 2b depicts an excerpt of the graph extracted exclusively from network traffic. The image shows the nodes that represent the valve and the temperature sensor in Fig. 2a. Moreover, it shows the setpoint and the Loop object that controls the valve. Since the setpoint and loop are part of the BACS configuration instead of physical components, they are typically not shown in the SCADA view by default. On the other hand, the radiator is not shown in the graph because its modeling is not needed as part of the BACS configuration. It is worth noting that Fig. 2b also depicts the context in which the heating module is involved. All the other nodes shown in the graph are part of different modules in the same subsystem. This context improves the *system visibility* since the configuration dependencies among different components are explicitly shown.

BACnet object graphs can help BACS administrators to identify problems in their infrastructures. In real BACSs, it is common to find components that are shared by multiple modules. SCADA software typically displays the hardware components used in each module, but not their role in other modules. Whenever atypical behavior is observed in different modules, a graph-based view of the BACS can help to quickly identify the possible root cause of the problem (e.g., a common ancestor in the graph).

IV. Conclusion

System visibility is a desirable property in Building Automation and Control Systems (BACSs). Although SCADA software provides a view of the system emphasizing the *physical* components, *configuration* components typically suffer from: (1) lack of visibility since they are difficult to find among multiple menus; and (2) lack of context since they are shown in isolation.

In this work, we have presented BACGRAPH, a tool that creates graphs of related BACnet objects. The graphical display of such graphs enhances the visibility of the *configuration* components of BACSs implemented using the BACnet protocol. Moreover, the links between nodes provide the context needed to better understand the role that each of them plays in the infrastructure. Our graphs are generated in a fully automated way using exclusively network traffic. We believe that the BACS visibility provided by BACGRAPH at the *configuration* level is complementary, and not a replacement, to the visibility provided by SCADA software that focuses on *physical* components.

Our experience using BACGRAPH shows that *close-to-complete* graphs are created in short time. Using approximately 3 days of passively collected traffic, we created a graph comprised of 13,733 nodes and 3,169 edges. However, a faster graph creation is possible by actively eliciting BACnet objects and properties from all the devices in the network.

Although the traffic processing speed is primarily determined by the protocol dissector (not part of our contribution), we showed that BACGRAPH can analyze traffic collected during 66.8 hours in approximately 40 minutes, using modest hardware resources. This is relevant because it indicates the feasibility to analyze live traffic.

For future work, we plan to use BACGRAPH as the core component of a monitoring tool that looks for modifications in BACS graphs using live traffic. Keeping track of changes in BACS graphs is important because they could be a sign of unintended misconfiguration or active attacks on BACSs.

References

[1] H. Merz, T. Hansemann, and C. Hübner, *Building Automation Communication Systems with EIB/KNX, LON and BACnet.© 2009*. Springer-Verlag Berlin Heidelberg.

[2] H. Esquivel-Vargas, M. Caselli, E. Tews, D. Bucur, and A. Peter, "BACRank: ranking building automation and control system components by business continuity impact," in *Computer Safety, Reliability, and Security - 38th International Conference, SAFECOMP 2019, Turku, Finland, September 11-13, 2019, Proceedings*, ser. Lecture Notes in Computer Science, A. B. Romanovsky, E. Troubitsyna, and F. Bitsch, Eds., vol. 11698. Springer, 2019, pp. 183–199. [Online]. Available: https://doi.org/10.1007/978-3-030-26601-1_13

[3] M. J. Assante and R. M. Lee, "The industrial control system cyber kill chain," *SANS Institute InfoSec Reading Room*, vol. 1, 2015.

[4] P. Ackerman, *Industrial Cybersecurity: Efficiently secure critical infrastructure systems*. Packt Publishing Ltd, 2017.

[5] M. Caselli, E. Zambon, J. Amann, R. Sommer, and F. Kargl, "Specification mining for intrusion detection in networked control systems," in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, T. Holz and S. Savage, Eds. USENIX Association, 2016, pp. 791–806. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/caselli

[6] ANSI/ASHRAE STANDARD 135-2016, "A data communication protocol for building automation and control networks," 2016.

[7] MBS, "BACeye," https://www.baceye.com/, 2021, [Online; accessed 10-Mar-2021].

[8] CODESYS, "CODESYS BACnet," https://www.codesys.com/products/codesys-fieldbus/bacnet.html, 2021, [Online; accessed 10-Mar-2021].

[9] Inneasoft, "BACnet Explorer. Explore and manage your BACnet/IP devices." https://www.inneasoft.com/en/bacnet-explorer/, 2021, [Online; accessed 10-Mar-2021].

[10] Optigo Networks, "Visual BACnet," https://www.optigo.net/visual-bacnet, 2021, [Online; accessed 10-Mar-2021].

[11] NSA, "GRASSMARLIN," https://github.com/nsacyber/GRASSMARLIN, 2021, [Online; accessed 10-Mar-2021].

[12] Neo4j, "Neo4j Graph Platform – The Leader in Graph Databases," https://neo4j.com/, 2021, [Online; accessed 09-Jan-2021].

[13] J.-P. Corriou, *Process control*. Springer, 2004.