Arhitektura sustava Interneta stvari temeljena na metaprotokolu za prijenos podataka

Milić, Luka

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Luka Milić

ARHITEKTURA SUSTAVA INTERNETA STVARI TEMELJENA NA METAPROTOKOLU ZA PRIJENOS PODATAKA

DOKTORSKI RAD

Zagreb, 2022.

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

LUKA MILIĆ

ARHITEKTURA SUSTAVA INTERNETA STVARI TEMELJENA NA METAPROTOKOLU ZA PRIJENOS PODATAKA

DOKTORSKI RAD

Mentori:
doc. dr. sc. Leonardo Jelenković
izv. prof. dr. sc. Ivan Magdalenić

Zagreb, 2022.

University of Zagreb
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Luka Milić

ARCHITECTURE OF INTERNET OF THINGS SYSTEMS BASED ON A DATA TRANSFER METAPROTOCOL

DOCTORAL THESIS

Supervisors:
Assistant Professor Leonardo Jelenković, PhD
Associate Professor Ivan Magdalenić, PhD

Zagreb, 2022

Doktorski rad izrađen je na Sveučilištu u Zagrebu, Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave.
Mentori: doc. dr. sc. Leonardo Jelenković i izv. prof. dr. sc. Ivan Magdalenić
Doktorski rad ima: 136 stranica
Doktorski rad br.: _____

O MENTORU (1)

Leonardo Jelenković rođen je 25. kolovoza 1973. godine u Pazinu. Diplomirao je 1996., magistrirao 2001. godine te doktorirao 2005. godine, na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Od 1.4.1997. godine, radi na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave. U znanstveno nastavno zvanje docent u znanstvenom području tehničkih znanosti - polje računarstvo izabran je 19.10.2006. Rezultate svojih znanstvenih istraživanja objavio je u osamnaest znanstvenih i stručnih radova u zbornicima domaćih i inozemnih znanstvenih skupova.

U nastavi sudjeluje u izvođenju predavanja, auditornih i laboratorijskih vježbi iz raznih predmeta na preddiplomskom, diplomskom i poslijediplomskom studiju. Pod njegovim mentorstvom je diplomiralo 54 studenata. Napisao je veći broj internih nastavnih publikacija iz područja operacijskih sustava, ugrađenih sustava i sustava za rad u stvarnom vremenu. Koautor je knjige Operacijski sustavi koja se koristi u istoimenom predmetu.

Osnovno područje znanstvenog istraživanja pristupnika uključuje svojstva operacijskih sustava u raznim okruženjima, od ugradbenih računala, sustava za rad u stvarnom vremenu, osobnih računala i poslužitelja, do okruženja Interneta stvari. U svojim nastavnim i istraživačkim aktivnostima nastoji izdvojiti bitne značajke operacijskih sustava potrebne studentima i inženjerima u navedenim okruženjima, kako u okviru studija, tako i nakon njega u okviru cjeloživotnog obrazovanja i usavršavanja. Osim navedenog osnovnog područja istraživanja, pristupnik se bavi i drugim problemima, uglavnom povezanih s mrežnom infrastrukturom, protokolima i sigurnošću.

ABOUT SUPERVISOR (1)

Leonardo Jelenković was born on August 25, 1973 in Pazin. He graduated in 1996, received his Master's degree in 2001 and his PhD in 2005 from the Faculty of Electrical Engineering and Computing, University of Zagreb. Since 1.4.1997 he has been working at the Faculty of Electrical Engineering and Computing, University of Zagreb, Department of Electronics, Microelectronics, Computer and Intelligent Systems. On October 19, 2006 he was elected as an assistant professor in the scientific field of technical sciences - area of computer science. He has published the results of his scientific research in eighteen scientific and professional papers in the proceedings of domestic and foreign scientific conferences.

He teaches several courses at undergraduate, graduate, and postgraduate levels. He has authored several internal teaching publications in the field of operating systems, embedded systems, and real-time systems. He is co-author of the book Operating Systems, used in the subject of the same name. Under his mentorship, 54 students have graduated.

His academic research areas include the characteristics of operating systems in various environments, from embedded computers, real-time systems, personal computers, and servers to the Internet of Things environment. In his teaching and research activities, he seeks to highlight the essential properties of operating systems needed by students and engineers in these environments, both as part of regular study and afterwards, in lifelong learning and continuing education. In addition, the candidate explores other topics, primarily related to network infrastructure, protocols, and security.

O MENTORU (2)

Izv. prof. dr. sc. Ivan Magdalenić rođen je 17. travnja 1977. godine u Čakovcu. Po završetku prirodoslovno-matematičke gimnazije 1995. godine upisuje diplomski studij na Fakultetu elektrotehnike i računarstva (FER) Sveučilišta u Zagrebu. Na FER-u je diplomirao 2000. godine na smjeru Telekomunikacije i informatika te magistrirao na istom fakultetu 2003. godine s temom "Elektronička razmjena poslovnih dokumenata". Od 2000. do 2004. godine radi kao znanstveni novak na FER-u, a od 2004. godien radi kao asistent na Fakultetu organizacije i informatike (FOI) Sveučilišta u Zagrebu. Doktorirao je na Fakultetu elektrotehnike i računarstva u znanstvenom polju Računarstvo s temom doktorske disertacije "Dinamičko generiranje ontološki podržanih usluga Weba za dohvat podataka". U znanstveno-nastavno zvanje docent izabran je 2012. godine, a u znanstveno-nastavno zvanje izvanredni profesor 2018. godine.

Područje znanstvenog istraživanja prof. Magdalenića uključuje automatsko i generativno programiranje, Internet stvari, elektroničko poslovanje, semantički web i druge napredne web tehnologije. Ivan Magdalenić je autor sveučilišnog udžbenika, objavio je poglavlje u znanstvenoj knjizi te je autor brojnih radova objavljenih u znanstvenim časopisima te na međunarodnim konferencijama.

Ivan Magdalenić aktivno sudjeluje u znanstvenim i stručnim projektima uvođenja elektroničkog poslovanja u Republici Hrvatskoj. Obnašao je dužnost pročelnika Katedre za informatičke tehnologije i računarstvo na Fakultetu organizacije i informatike Sveučilišta u Zagrebu u razdoblju od 2013–2017. Na preddiplomskom, diplomskom i specijalističkom poslijediplomskom studiju Fakulteta organizacije i informatike nositelj je brojnih kolegija kao što su Arhitektura računalnih sustava, Mreže računala, Operacijski sustavi, Sigurnost operacijskih sustava, Sigurnost umreženih računalnih sustava, itd.

Dobitnik je nagrade za mladog znanstvenika na Fakultetu organizacije i informatike 2012. godine.

ABOUT SUPERVISOR (2)

Ivan Magdalenić, PhD, was born in Čakovec on April 17, 1977. After graduating from the secondary school in Čakovec, he joined the undergraduate program at the Faculty of Electrical Engineering and Computing, University of Zagreb, in 1995. He received his B.S. and M.S. degrees in Electrical Engineering with a major in Telecommunications and Information Science from the University of Zagreb, in 2000 and 2003, respectively. The

research topic of his Master thesis was “Business Documents Interchange”. He worked as a research assistant at the Faculty of Electrical Engineering and Computing, University of Zagreb from 2000 to 2003. Since 2004 he has been working as a teaching assistant at the Faculty of Organization and Informatics, University of Zagreb. He holds a PhD from the Faculty of Electrical Engineering and Computing in the field of Computer Science with the topic of his doctoral dissertation "Dynamic generation of ontologically supported Web services for data retrieval". He was elected assistant professor in 2012, and associate professor in 2018.

Field of scientific research of prof. Magdalenić includes automatic and generative programming, Ineternet of Things, electronic business, semantic web and other advanced web technologies. Ivan Magdalenić is the author of a university textbook, he has published a chapter in a scientific book and he is the author of numerous papers published in scientific journals and international conferences.

Ivan Magdalenić actively participated in scientific and professional projects for the introduction of electronic business in the Republic of Croatia. He was the head of the Department of Information Technology and Computing at the Faculty of Organization and Informatics, University of Zagreb in the period from 2013-2017 and 2021-nowadays. In undergraduate, graduate and specialist postgraduate studies at the Faculty of Organization and Informatics, he teaches numerous courses such as Computer Systems Architecture, Computer Networks, Operating Systems, Operating Systems Security, Networked Systems Security, etc.

He won the Young Scientist Award at the Faculty of Organization and Informatics in 2012.

ZAHVALA

Prvo zahvaljujem mentorima Leonardu Jelenkoviću i Ivanu Magdaleniću na golemom trudu uloženom u mentoriranje ovoga rada. Zahvaljujem kolegama-profesorima Marinu Golubu i Nikoli Ivkoviću na korisnim svjetovanjima pri njegovoj izradbi. Zahvaljujem voditeljicama Studentske službe Mirjani Grubiši i Đurđici Tomić-Peruško na tehničkoj pomoći. Zahvaljujem profesorima doktorskoga studija Domagoju Jakoboviću, Vladi Sruku i Janu Šnajderu na susretljivosti tijekom rečenoga studija. Zahvaljujem dekanima matičnoga fakulteta Nini Begičević-Ređep i Vjeranu Strahonji na osiguravanju uvjeta za pohađanje doktorskoga studija. Zahvaljujem kolegama-studentima Nikoli Baniću, Janu Bergeru, Karlu Kneževiću i Ivanu Župančiću na poticaju u radu. Konačno zahvaljujem ženi Antoniji na neograničenoj potpori tijekom studiranja.

SAŽETAK

U radu se predlaže arhitektura za Internet stvari temeljena na metaprotokolu za komunikaciju među čvorovima i sustavu pravila za konfiguraciju čvorova. U njoj se ne upotrebljava tradicionalna podjela na slojeve, nego se svaki čvor promatra kao ravnopravan sa svima ostalima. Kategoriju čvora određuje uloga koja je za taj čvor skrojena. Pri tom čvor može biti jednostavan senzorski čvor, ili nešto složenije, kao što je usmjernik ili čvor koji pohranjuje i obrađuje podatke. Čvor može imati i više uloga, a one se ostvaruju ili posebnim programima ili definiraju sustavom pravila, kao u ostvarenom prototipu programske potpore.

Bez obzira na ulogu – čvorovi komuniciraju tim metaprotokolom, koji je temeljen na podatkovnim operacijama jezika SQL. Uporaba takvih operacija omogućuje vrlo veliku fleksibilnost u osmišljavanju sustavâ. S druge strane, brojni mehanizmi kraćenja poruka i delegiranje određenih, većih ili manjih, stvari nižim protokolima omogućuju stvaranje što kraćih i jednostavnijih poruka, pa tako i uključivanje i onih najjednostavnijih čvorova u takav sustav.

Usporedba predloženoga modela arhitekture s postojećim znanstvenim i komercijalnim rješenjima pokazuje da se u nekim slučajima očekuju prednosti uporabe predložene arhitekture pred tim rješenjima. Te su prednosti najizraženije kad se upotrebljavaju najjednostavniji čvorovi i kad se zahtijeva konfiguracija mreže po volji na način drugačiji od tradicionalnih arhitektura tipa „stvar-usmjernik-poslužitelj/oblak-klijent/korisnik“.

Ključne riječi

internet stvari, arhitektura, metaprotokol, baze podataka

ARCHITECTURE OF INTERNET OF THINGS SYSTEMS BASED ON A DATA TRANSFER METAPROTOCOL

Internet of Things (IoT), defined as “connecting anything, anytime, anywhere”, is a concept by which things around us should communicate, deduce and decide without human interference, in order to unburden man from everything that can be automatized. Today already not only there are thousands of articles written in the field, but also leading IT companies like IBM, Microsoft, Amazon, Google, Cisco... already offer everything for IoT, be it hardware solutions for “things”, software solutions for their gateways, or cloud solutions for data processing.

IoT is being developed since the 90s, when it was called “control networks”, and got its name in 1999., when Procter & Gamble needed to put things in their supply chain to the internet. Today it is applied in many fields, like agriculture, commerce, military, health, “smart” spaces, “sentient” devices, videogames, data sciences... However, there are many challenges in IoT, like flexibility, security, and simplicity.

The central part of the dissertation is a proposal of a new architecture that can stand up to the challenges. The architecture is based upon three foundations. Firstly, differences between node categories in IoT are disregarded and every node is seen as a “smart” node with a uniform interface. Secondly, a “metaprotocol” is defined for the transfer of data and operations between the nodes. The metaprotocol is designed to be able to be transferred using any lower-layer protocol. The operations in the metaprotocol are mainly based on SQL operations, namely SELECT-ing the data and receiving the response. Thirdly, the architecture includes a specially designed node configuration. Its biggest part is the rule system which enables including both very simple and very complex nodes in the architecture, along with the metaprotocol's flexible interface.

There are also some goals meant to be achieved with the proposal. Those are mainly flexibility of application and simplicity of usage. Not only should the architecture be able to encompass use cases from all areas of life, but it should also be applicable to resource-constrained devices. The simplest nodes can simply rely on many implicit behaviors, the more complex ones can use the rule system, or even override implicit rules. Security should also not be overlooked, and enable not only relying on lower-layer protocols, but also some explicit metaprotocol mechanisms, along with an authorization model in the

node configuration. The most important part of the architecture is building a node model and a communication model which are intended to be as general as possible.

In the Chapter 2 an overview of the research field is provided. At the beginning three common problems are identified. Firstly, the great majority of the research is focused on a very specific application. Researchers often implement a smart thing or a smart space with few devices and name that IoT, even though HCI would be a better name. Secondly, security is often left for later, especially the privacy of huge amounts of data. Even though huge amounts of data are already being collected by e.g. Google, there are many scenarios where access to that data by third parties is completely undesirable and harmful. Thirdly, IoT architectures are extremely heterogenous and incompatible today, thanks to their very specific data models and protocols.

Considering all that, a very general model of IoT is needed which is not burdened with a single application. Security and privacy are also needed, especially in an end-to-end fashion. However, it must not be disregarded that the users must find the model easy to use, which is a quality that many commercial solutions do not have as they often require installation of multiple software and interfaces. That is a part of the problem of IoT deployment, and the implicit actions in the proposed architecture are intended to address exactly that. A comparison with IP is drawn, and it supports the philosophy of “easy things being simple and hard things being possible”.

Next, common protocols used in IoT are analyzed. Bluetooth and, even more, its variant Bluetooth Low Energy are commonly used. Wi-Fi and, very rarely, its variant IEEE 802.11ah are also often mentioned. IEEE 802.15.6 and WiMAX are sometimes mentioned in the literature but almost never in practice. IEEE 802.15.4 and its most common upper-layer protocol, ZigBee, are also often used, but even complete-stack solutions like Thread and Z-Wave are only oriented on local networks, much less datalink-layer protocols mentioned so far.

For global networking standard internet protocols are used, like TCP, UDP, and IPv4, and other proposals for address translation or new IP for IoT fell into obscurity. As for the application layer, most common protocols are CoAP, XMPP, MQTT, AMQP, and other RESTful architectures are used, sometimes even HTTP. However, those are mainly publish-subscribe-oriented and very problematic without a client-server network architecture. To sum up, no protocol described here is as versatile as needed.

The metaprotocol is based on SQL, which is not used in communication in IoT, and SPARQL is the most similar technology that sometimes is used. However, SQL is more suited for the versatile data model, and it is a language more known to computer scientists.

Next, ten similar architectures from the scientific literature are analyzed and some common problems of architectures are identified. These are being created only for a specific use, requiring learning a special language, encompassing only local networks, having a special message abstraction layer, strictly separated node categories, no support for changing node role or its complexity, and making inserting a simple node with adding features later very difficult. On the other hand, the proposed architecture is analyzed to be intended for general-purpose IoT networks, to be using only SQL, to be handling messages directly, to be able to simply change node configuration, similarly to be able to add or remove features, to be providing security from the start, and finally to even be providing energy saving.

The model of the architecture is demonstrated in Chapter 3. First thing shown is the connectivity model, which encompasses three node types stemming from the physical properties of networks. The first node type is a metaprotocol node, which uses the metaprotocol to communicate with other such nodes. These nodes can be Class 0, 1, 2 or higher devices. The second type are the user nodes which access data from the nodes by some other interface, like web. The third type are networking nodes which carry metaprotocol messages without knowing anything, like IP routers. A connection can be made through any number of metaprotocol or network nodes.

This general model ensures that any architecture type can be implemented, including the standard 4-tier one. However, any node can take any part of any role or communicate with any other node in any direction.

The IoT actions intended to be supported in this architecture are discussed next, and these are sending data, querying data, subscribing, unsubscribing, establishing “confidentiality” through encryption, establishing “authenticity” through message signing, acknowledging messages, notifying errors, managing information about nodes, managing node data, managing triggers and handling node authorizations.

The description of the data model comes next. It is based on a relational data model, i.e., tables, that can be distributed so that any node can have only a part of its or other nodes' data, and the tables are identified by EUI-64 of the node. EUI can be calculated from lower

layers and a node can have more than one. There can also be other arbitrarily-named tables in the node. Data types are SQL ones, and the table rows are implicitly timestamped. Data can be stored in a more precise type, and its tables can be implicitly extended by new columns, whose names can be anything Unicode, together with strings, according to the standard.

The metaprotocol is described next. Firstly, the main differences between it and the “conventional” protocols are laid out. The metaprotocol can be transferred over any OSI layer, and that is because it encompasses protocol properties intended for multiple OSI layers. Also, the metaprotocol's usage depends on the lower layers where all things that are included on them can be dropped from it, and similarly the metaprotocol can turn on or off some of their mechanisms if they are not needed. In any case, the metaprotocol is intended to be as thin layer as possible in a given situation, sometimes not even a layer at all. Finally, the metaprotocol can be used both in its binary form and its text form, something not seen before.

The metaprotocol message fields are message header, message identifier, message length, destination, source, payload and CRC. These can all be dropped if necessary, and in the header the first 5 bits determine which of them are present, and the last 3 determine whether acknowledgment, confidentiality, and authenticity are needed.

There are 10 message types, and the first is DATA, intended to carry a single data, a row or even a whole table of data. SELECT message serves for querying data in the shape of an SQL query working on exposed tables. SELECT_SUBSCRIBE similarly enables subscription for changes in SQL results, UNSUBSCRIBE enables unsubscribing, and UNSUBSCRIBE_ALL unsubscribing from all existing subscriptions. QUICK serves for quickly transferring a byte and can be simply converted from and to a DATA message. HELLO is used for node registering if there is no previous or recent communication, or simply heartbeats.

ACKNOWLEDGMENT acknowledges previous messages requesting it, and it can even be used for multiple acknowledging, aside for pinging devices. PAYLOAD_ERROR notifies malformed payloads, where other types of malformed messages are silently dropped. OPERATION_UNSUPPORTED notifies a too complex operation in the payload, especially complex SQL queries sent to simple devices.

The node model is described next. It consists of the following parts: communication, message processing, event handler, node data, web interface, manual configuration, and node management. The model is intended to be modular and therefore only the most complex nodes are expected to have all parts, the simpler nodes can drop some parts, and the simplest nodes can have only one main communication loop.

Next is the description of message shortening. DATA messages can drop a part or the whole table header, also drop spaces and the final semi-colon delimiter, and can also encode SQL keywords, and more. SELECT messages can drop a part or the whole FROM, also drop spaces and the semi-colon, and also encode keywords, and more. Similar goes for the SELECT_SUBSCRIBE, UNSUBSCRIBE, and UNSUBSCRIBE ALL. QUICK cannot be shortened, but HELLO can be just an empty message. Some binary encoding and delimiter dropping can also be done for ACKNOWLEDGMENT, PAYLOAD_ERROR, and OPERATION_UNSUPPORTED.

The encoding of the SQL keywords means that all of the SQL multi-character operators, aggregate functions, and keywords expected to be found in a SELECT query can be encoded as a single byte in the hexadecimal 0x80-0xFF range. Along with the other described mechanisms, this shortens the queries significantly.

The node middleware was implemented according to the node model. It consists of underlying-protocol modules, specifically, Bluetooth Low Energy, IEEE 802.15.4, TCP, and UDP ones, then it has a main program, external functions, a database, and a web server. The protocol modules communicate over network adapters with other nodes, and different categories of users access the system through the web server. As it is with the node model, the middleware is modularly created – a particular system does not need to include all of the parts, neither by this implementation nor any other different ones.

The rule system used as a part of the node configuration enables creation of arbitrarily complex rules. The rule can consist of many elements, but the majority does not have to be used: its owner, identifier, type, message action, additional SQL-query or bash command, additional message creation, created through a query or a command, for sending it to another node or injecting here, additional rule activation, additional rule deactivation, active flag, and the last-run time for periodic rules.

Each node can have configuration for: users, node data, rules, node permissions, protocols, adapters, node metadata, metaprotocol configuration, certificates, and private keys. There

exists the public user, regular users, regular administrators, and the root user. The user permissions available in the model are: viewing/modifying data/rules/metadata/ metaprotocol configuration/permissions/users/protocols/adapters/certificates/keys/yourself/ others' configuration, sending/injecting messages, executing queries/rules, and logging in. The metaprotocol configuration that can be defined in the model is: message forwarding, route learning, duplicate expiration, route expiration, overriding security for sending/receiving, default gateway, custom EUI, custom transport-layer secure/insecure port.

Next comes the security description, where the idea is that lower-layer mechanisms should be used as much as possible, and only if those are not available or enough, cryptographic primitives can be used in the metaprotocol. The technologies used are AES, RSA, SHA, CBC, and PKCS paddings, and the primitives offered are the digital envelope, digital signature, and both. Through message rules some more advanced security management is possible.

The default rules turn on lower-layer security for securely sending messages, and do not forward them if neither lower-layer security or metaprotocol security are turned on. They also drop secure messages unsecured by neither of the lower-layer security or the metaprotocol security, but both behaviors can be overridden. The rules can also provide the so-called proxy security, where cryptography can be done in some trusted more powerful node instead of a resource-constrained one.

The message routing in the metaprotocol differs a lot from the IP routing. Duplicate detection is used, and it also eliminates loops, but the message can have as many hops as needed. It can also have multiple routes between nodes. The simplest nodes can simply use broadcast, the more complex ones should remember some default metadata for routes to work, and the most advanced ones can have their own metrics and rules.

How to achieve specific IoT actions is shown next. In short, sending/receiving is done with DATA/SELECT messages, subscriptions with SELECT_SUBSCRIBE/UNSUBSCRIBE/ UNSUBSCRIBE_ALL messages, confidentiality/authenticity with C/A bits, configuration, and rules, acknowledgment/errors with ACKNOWLEDGMENT/PAYLOAD_ERROR/ OPERATION_UNSUPPORTED, metadata/data/trigger/permission configuration through the node configuration. At the end of the chapter there is a brief statement that, even though there is no explicit updating or deleting of the node data in the messages, just adding new data is still Turing-complete because only new data can be fetched.

In Chapter 4 there are some scenarios and use cases of the architecture for better understanding of communication and later-described potential advantages of the architecture. The first example details simple communication between 2 nodes, a gateway, and a user. The second details subscriptions in the same system. The third example details communication between 2 nodes in different local networks bridged in cloud. The fourth example details security options in the same system.

The fifth example talks about how raw programming in IoT systems can be evaded through the rule system implemented in the middleware. The specific rule format is also described here with examples. In the next section there is a guide for implementing the architecture. Basically, you start with some general architecture and choose node by node for implementing its functionalities. Those can be implemented with a simple programmed action for the simplest nodes, with the implemented middleware for complex ones, and with some other software for specific requirements.

The final section gives a use case of a smart farm where across some big piece of land there are smart feeding stations, water stations, fences, gates, etc. There are also animals with smart collars roaming the farm. They come to a central barn every so much for milking, and the barn is a central processing hub. These animals can interact with the smart stations as clients, but can also piggyback the stations' data, possibly over their own ad-hoc networks, back to the barn, where they act as servers. This is a very unconventional network configuration. Also, adding or removing some nodes has very little effect on the network in whole. The security is not considered by default as physically isolated subnetworks are used. However, if there are very close neighbors, some solutions are offered.

In Chapter 5 the architecture is evaluated by multiple criteria. Firstly, the comparison with the literature is summed up, and then it is compared with common commercial solutions like Azure. Even though those solutions have very many users for which they work very well, they are not ideal for all use cases. For example, if arbitrary node roles or extremely simple nodes are needed, they have setbacks. There are also free solutions out there, like Node-RED for smart gateways. Unfortunately, these solutions only cover parts of the IoT network and require some hardware requirements not always available. There are also some setbacks to the proposed architecture, like disregarding advanced analyses and displays, so naturally pros and cons are to be weighed.

Next there is a comparison about how easy is to include a simple node in common 4-tier architectures and in the proposal. It is shown that the implicit operations enable simplest nodes to work with almost no additional configuration. It can also be seen that thanks to no rigid hierarchy, one-size-fits-all software, and uniform access to nodes, usability is a very important pillar of the proposal.

Next energy savings are analyzed. When sending simple messages, it can be seen that the common solutions CoAP, XMPP, and MQTT are at disadvantage, especially when only 1 byte is sent rarely, and a little less when many bytes are sent often. That goes even disregarding the fact that those protocols require the complete TCP/IP stack with many handshake and acknowledgment messages.

There is also detailing about the implementation of the middleware. It is hosted on GitHub and written in C++ for the main program, PHP for web configuration, bash for test scenarios, and Kotlin for an Android app for additional testing. PostgreSQL and Apache Web Server were used and everything was setup for ArchLinux operating system. The main program is implemented using multithreading for UPMs and message queues for connecting the software model parts. The scenarios were tested using systemd-nspawn containers. Along with everything mentioned there are also some helper scripts in the repository. The implementation still needs further work for some less important features.

Closing the chapter there is a security analysis of the architecture. There are three cases detailed. In the first case the security is not implemented by either the lower-layer protocols or the metaprotocol. In that case anyone can do any malicious operation as soon as there is no physical isolation between the network and the malicious node. In the second case the security is implemented through lower layers, and in that case the link between two nodes is secured. However, if there are multiple hops to the destination, all those nodes and the links between them need to be trusted, or attacks are again possible.

In the third case, the security is implemented through the metaprotocol. Since the metaprotocol has end-to-end security, if the confidentiality is employed no one can read the message, and if there is authenticity no one can modify it. There are also some mechanisms against replay attacks, like the message ID and duplicate handling. As for denial-of-service attacks, the statelessness of the metaprotocol is important, but to prevent attacking the lower-layer protocol firewalls should be used.

To sum up everything, this dissertation proposes a new architecture for IoT, intending to enable easy inclusion of simple nodes and be flexible for any network configurations. This is accomplished by the metaprotocol and the node model. Simple nodes can rely on more complex ones and the implicit behavior. Complex nodes can use the rule system which can do advanced operations with the messages.

The metaprotocol's most important features are being able to be transferred over any lower-layer protocol and adding to the communication only what was not already implemented in that protocol. The messages are modeled mainly upon SQL and its SELECT and INSERT operations. There are some examples in the dissertation not only about how the metaprotocol works but also where the proposed architecture has advantages over the existing solutions.

In the practical part, a complex middleware for smart nodes is implemented and tested. However, the architecture does not exclude all other solutions, but enables compatibility with them mainly through its very flexible data model. Finally, for the future work some middleware features must be finished and then the architecture can be deployed to physical systems. Then real network performance will be tested for further evaluation of simplicity and flexibility.

In the Appendix A, there is the mapping of encoded SQL keywords mentioned when talking about the message shortening. Along with listing all 128 words for encoding, there are also some comments for the replacement. The words are listed alphabetically and there is no additional categorization applied in the list.

SADRŽAJ

1. UVOD

Internet stvari (engl. *Internet of Things*, IoT) jest sintagma koja se u računarstvu iz dana u dan pojavljuje sve više [1]. Definicija IoT-a koja se provlači od njegovih najranijih dana [2] jest da je to „povezivanje svega, uvijek i svagdje“ (engl. *connecting anyTHING anyWHERE anyTIME*) [3]. Efektivno se može, kako i u početku njegova ozbiljnijega istraživanja [4], tako i danas [5], reći da se radi o nazivu za koncept pokrenut po zamisli da sve fizičke stvari od kojih čovjek može imati koristi trebaju komunicirati, zaključivati i odlučivati bez njegova uplitanja, a u svrhu oslobađanja čovjeka od svega što te umrežene digitalizirane stvari mogu izvesti brže, jednostavnije ili bolje umjesto njega.

Danas se područje IoT-a intenzivno razvija, i znanstveno i komercijalno. Znanstveno govoreći, objavljene su tisuće članaka iz toga područja [6] u kojima se istražuju njegove primjene u svim područjima života. S druge strane IBM [7], Microsoft [8], Amazon [9] i slične tvrtke [10] odavna pružaju usluge za IoT. Usluge variraju od sklopovskih rješenja za spomenute „pametne“ (engl. *smart*) stvari, preko programskih rješenja za prijenos podataka među čvorovima IoT-a u mreži, pa sve do poslužiteljskih kapaciteta za obradbu prikupljenih informacija.

IoT nije nova tehnologija, nego pametna kombinacija postojećih tehnologija na nove načine. Naime, početci izgradnje IoT-a sežu još u 1990-te, kad su se takvi sustavi razvijali pod nazivom „kontrolne mreže“ (engl. *control networks*) [11], dokle mu je današnji naziv nadjenuo Kevin Ashton 1999. radeći za tvrtku Procter & Gamble [12]. Već iz samoga toga naziva može se uočiti i misao vodilja, a to je povezati „pametne stvari“ u mrežu. Isto tako, može se uočiti i da se ne radi o novoj tehnologiji, jer stavljanje stvarî na Internet to sigurno nije. Od izvorne zamisli digitaliziranja dobavnoga lanca danas se je IoT razvio u višemilijardsku industriju [13].

IoT je vrlo široko područje istraživanja i ujedinjuje i računalne mreže i računalne arhitekture i interakciju čovjeka i računala (engl. *Human-Computer Interaction*, HCI), među ostalima [1]. Kao takav on otvara brojne mogućnosti služenja čovjeku. To se vidi u svim područjima njegova života: od poljodjelstva i trgovine preko vojske i zdravstva do pametnih prostora i „svijesnih“ uređaja [13], ali se otvaraju i brojni izazovi koje treba pri tom svladati. Najvažniji od tih izazova, kao i pokušaj odgovora na te izazove, bit će

izloženi u ovom radu. Izazovi na kojima će posebno biti naglasak jesu izazovi fleksibilnosti, sigurnosti i jednostavnosti.

Središnji dio ovoga rada jest prijedlog nove arhitekture za IoT prilagođene tim izazovima. Ta se arhitektura temelji na trima važnim načelima izgradnje, koja će se ovdje ukratko izložiti. Prvo je načelo zanemarivanje razlike između čvorova u IoT-u koji izvorno pripadaju različitim kategorijama. Npr., to su spomenute kategorije „stvarî“ (engl. *thing*) koje stvaraju podatke, usmjernika koji ih prenose, poslužitelja koji ih obrađuju. To se obavlja tako da se svaki čvor promatra i s njim komunicira preko jednolika sučelja kao da je jednostavno „pametni čvor“.

Drugi je temelj, odnosno načelo, izradba „metaprotokola“ (engl. *metaprotocol*) za prijenos podataka, kao i operacija, među tim pametnim čvorovima koji izgrađuju IoT. Taj je metaprotokol izgrađen tako da se može prenositi, odnosno učahuriti, kojim bilo nižim protokolom. To može biti protokol podatkovnoga (engl. *datalink-layer*), mrežnoga (engl. *network-layer*), prijenosnoga (engl. *transport-layer*), primjenskoga (engl. *application-layer*) sloja današnjega Interneta, a može se prenositi i bez ikakva nižega protokola ako je makar definiran fizički sloj (engl. *physical layer*). Temelj operacija u tom metaprotokolu jesu operacije baza podataka – u daljem tekstu samo „baza“ – definirane u njihovu *strukturiranom upitnom jeziku* (engl. Structured Query Language, SQL) [14]. Konkretno, operacije su najviše nadahnute naredbom `SELECT` (hrv. *odaberi*), odnosno operacijom projekcije u relacijskom modelu podataka, i njezinim odgovorom, a to su tablice s redcima i stupcima. Druge se naredbe jezika SQL ne uzimaju u obzir, ali usprkos tomu model je Turing-potpun koliko i sam jezik SQL.

Konačno, ova se arhitektura temelji i na posebno oblikovanoj konfiguraciji čvorova u IoT-u. Pri tom se najviše govori o sustavu pravila (engl. *rule system*) kojim se mogu izgraditi, „isprogramirati“, pojedini čvorovi tako da oni mogu imati raznovrsne „uloge“ i raznovrsne složenosti, a da bi se mogli ostvariti sustavi IoT-a različitih mogućnosti i različitih veličina. Naime, kombinacijom definiranja povezanosti čvorova metaprotokolom i definiranja operacija čvorova sustavom pravila – sustavi temeljeni na predloženoj arhitekturi mogu se primijeniti na širok raspon scenarijâ, što je i jedan od ciljeva s kojima se je i krenulo.

Ciljevi predložene arhitekture su fleksibilnost primjene i jednostavnost izgradnje. Predložena arhitektura treba poduprijeti obrasce uporabe (engl. *use cases*) iz svih područja života, što je, pokazat će se, velik izazov IoT-a. Nadalje, arhitektura mora biti primjenjiva i

u uređajima s ograničenostima u kontekstu procesorske snage, količine memorije, energije baterije i slično. U prikazanoj arhitekturi navedeni se ciljevi ostvaruju kroz raznolike mogućnosti čvorova, ugrađenim implicitnim ponašanjima, te modularnom građom.

Najjednostavniji se čvorovi u sustavu mogu ostvariti oslanjanjem na implicitno ponašanje ugrađeno u predloženi model čvora. Za nešto složenije sustave korisnik može definirati dodatna pravila. U još složenijim primjerima može se sve riješiti preko skupa pravila, čak i bez uporabe implicitnih. Primjeri s različitim navedenim pristupima prikazani su kroz cijeli rad, posebice u poglavlju 4. Operacije poput: prikupljanja podataka od senzorâ, obradbe podataka, pohrane podataka, prosljeđivanja poruka drugim čvorovima, odgovora na zahtjeve od drugih čvorova itd. mogu se ostvariti i samo kroz postavke čvora, što znatno pojednostavljuje izgradnju i uporabu sustavâ.

U ovoj je arhitekturi puno truda posvećeno i vidu sigurnosti. Konkretno, moguće je i oslanjanje na postojeće mehanizme nižih protokola, ali i pružanje gotovih mehanizama za kriptiranje, dekriptiranje, potpisivanje poruka, provjeru potpisa, kad to nije omogućeno ili isplativo nižim protokolom. Isto vrijedi i za sprječavanje tzv. izmišljanja poruka, odnosno ponovnoga odašiljanja starih poruka. U dijelu koji se bavi konfiguracijom čvora i sustavom pravila izradio se je model autorizacije koji omogućuje podjelu korisnika na obične i javne korisnike i obične i korijenske administratore ako je ona potrebna u sustavu koji se izgrađuje. Pri tom opet svaki korisnik može i ne mora imati svoj skup dopuštenih operacija i pristupačnih podataka.

Ono što se ističe kao najvažniji dio predložene arhitekture jest izgradnja što je moguće općenitijega modela čvora, opisana pretežno kroz spomenutu konfiguraciju, odnosno pravila, u svrhu ujedinjavanja svih različitih kategorija čvorova. Takav pristup u literaturi još nije istražen, a omogućavao bi ostvarivanje kojega bilo tipa sustava IoT-a. Pogotovo kad se razmotri kako metaprotokol ujedinjuje sve slojeve IoT-a, vidi se njihovo komplementarno djelovanje koje omogućuje istinsku jedinstvenost predložene arhitekture među ostalim predloženim rješenjima, što je razjašnjeno kroz poglavlje 3.

Disertacija je organizirana kako slijedi: u poglavlju 2 dan je pregled područja, odnosno prikaz postojećih problema i izazova u IoT-u i pokazivanje kako se predložena arhitektura s njima suočava, zajedno s izlaganjem najsličnijih rješenja iz znanstvene literature. U poglavlju 3 prikazuje se model arhitekture, počevši od modela podataka preko modela radnja IoT-a koje se u arhitekturi trebaju moći ostvariti i modela metaprotokola pa sve do

modela konfiguracije i sustava pravila, modela sigurnosti, modela usmjeravanja, itd. Posebno se ističu i načini za smanjivanje komunikacije, odnosno uštedu energije. U poglavlju 4 se sustav razrađuje na primjerima uporabe, prvenstveno s obzirom na metaprotokol i sustav pravila, a posebno se razrađuje i jedan obrazac uporabe na kojem se vide neke prednosti ovoga prijedloga. U tom se poglavlju naglašuju i određene mogućnosti metaprotokola i pravilâ, a i ističe njihova općenitost u svrhu ostvarivanja sustava IoT-a po volji.

U poglavlju 5 raspravlja se o konkretnim prednostima i nedostatcima prijedloga u odnosu na postojeća rješenja, i to i znanstvena i komercijalna. Posebno se ističu slučaji u kojima ovaj prijedlog ima najveće prednosti, a opisuje se i izgrađeni prototip općenite programske potpore za ostvarivanje arhitekture u najsloženijim čvorovima, i na kraju je još i sigurnosna analiza različitih slučaja u uporabi ove arhitekture. U poglavlju 6 daje se, konačno, zaključak ovoga rada i smjer nastavka istraživanja.

2. PREGLED PODRUČJA

Danas se u području IoT-a objavljuju tisuće članaka [15], više ili manje povezanih s njegovom početnom zamišlju sveopće povezanosti. Kad se pogleda sva ta količina istraživanja, vidi se prvo da se svako njegovo potpodručje intenzivno razvija, kao i da se IoT-u primjena širi, ali i da postoje brojni problemi ili izazovi s kojima se još treba suočiti [16]. Prvi, a sigurno i najveći problem, jest što se velika većina istraživanja IoT-a odnosi na vrlo konkretne primjene [17]. Takva usredotočena istraživanja ne daju općenita rješenja. Naime, IoT je danas zvučna riječ (engl. *buzzword*) za sve što ima veze s „pametnim stvarima“, ili, najčešće, „pametnom okolinom“.

Činjenica jest da je IoT „kišobran“ (engl. *umbrella term*) za mnoge stvari koje nemaju puno veze jedna s drugom, a ni s osnovnom zamišlju IoT-a „povezati sve, uvijek i svagdje“. Konkretno, brzi pregled najnovijih članaka, što publicističkih što znanstvenih, na Internetu, otkriva da se taj termin može staviti na sve što ima veze s digitalizacijom. Pa tako, dovoljno je otvoriti bazu članaka iz referencije da bi se vidjelo da je dovoljno ostvariti iole „pametnu“ stolicu, postelju, vrata, pa sve do spremnika kisika [18], …, učionicu, rudnik, cestu, pa sve do parkova prirode [19], … ili bilo koji drugi predmet ili mjesto da bi se tomu mogao pridijeliti naziv „IoT“, iako se možda radi samo o tom da taj predmet s vremena na vrijeme pošalje nešto pametnomu telefonu ili pametni telefon s vremena na vrijeme pošalje neku naredbu tomu predmetu.

Iako je istina da je i ovo spomenuto dio IoT-a – a na takvim najjednostavnijim sustavima bit će i puno naglaska kroz ovaj rad – IoT je puno više od toga. Ovakvo usko shvaćanje IoT-a ima i katkad poseban naziv u literaturi, koji se zove „izgradnja Intranetâ stvari umjesto Interneta stvari“ [20]. Nedostatak općenitosti, katkad zvan i svođenjem IoT-a na „povezivanje zubnih četkica na Internet“ [21], svodi IoT na interakciju HCI. On je i pojačan činjenicom da je područje primjene IoT-a stvarno golemo pa je dovoljno naći djelić ili nišu u kojem još nema specifične primjene te to svesti u područje IoT-a.

Drugi je veliki problem postojećega istraživanja što se često sigurnost ostavlja „za poslije“, odnosno, izgrade se cijele arhitekture, a u njima se uopće ni jedan dio ne posvećuje činjenici da je glavna zadaća IoT-a prikupljanje i obradba gomile, moguće i privatnih, podataka i da bi te podatke trebalo zaštititi od zlonamjernih korisnika [22]. Usprkos tomu, teško je taj problem pripisati istraživačima, budući da izgradnja arhitekture IoT-a zahtijeva

ujedinjavanje različitih područja. Tako da ako se sami korisnici, ako ih u sustavu koji netko predlaže uopće ima, ne zapitaju kako se upotrebljavaju njihovi podatci, više će se vremena posvetiti važnijim stvarima u tom trenu. Sličan se argument može postaviti i za, recimo, zanemarivanje uštede energije.

Korisnici su otupjeli na neograničeno prikupljanje i obradbu njihovih privatnih podataka, kako su i tako praćeni 24 sata na dan [23]. Pogotovo u sustavima izgrađenima za samo jednu posebnu primjenu to se i ne ističe kao velik problem. Ipak, ako se odmakne od takva uska shvaćanja IoT-a i on se gleda kao ta „mreža svega, uvijek i svagdje“, ne treba biti stručnjak da se uoči da će do problema kad-tad doći.

Spominjući to ujedinjavanje različitih mreža dolazi se i do trećega velikoga problema, a to je upravo golema razjedinjenost IoT-a, isprepletena s već opisanim problemima usredotočenosti samo na određene primjene i sigurnosti [24]. Naime, ako se gradi nov sustav koji služi precizno određenoj svrhi, često se stvara prilagođen model podataka koji nije poveziv ni s jednim drugim modelom. Katkad se izgradi i prilagođen protokol za njihov prijenos koji nema veze ni s jednim drugim protokolom. Da bi se uopće došlo do „mreže svega, uvijek i svagdje“, osim izgradnje modela koji bi mogao „probaviti“ većinu primjena, treba i izgraditi povezivanje jednoga načina ostvarivanja toga modela u jednoj mreži s drugim načinom u drugoj.

Razmotrivši sve gore napisano, vidi se da je jedan od velikih izazova IoT-a sklopiti dovoljno općenit model čvora, podataka, povezanosti i sl. koji nije primjenjiv samo na jedan razred uređaja u jednom području primjene nego istodobno i na mnoge, različite uređaje u različitim područjima [25]. Izgrađujući taj model svakako se treba i na sigurnost misliti od početka, a ne reći da „se ona može dodati ako treba“ kao što to često jest. Posebno kao poddio sigurnosti treba se voditi računa o privatnosti podataka. Kako se danas za privatnost upotrebljavaju mehanizmi „kriptiranja s kraja na kraj“ (engl. *end-to-end encryption*), nešto slično treba omogućiti i u svakom predloženom modelu.

Osim same izgradnje modela treba voditi i računa o tom da netko taj model mora ostvariti, odnosno implementirati, i upotrebljavati pa se ne može zanemariti prigovor o sljedećem: zašto bi itko to činio kad već postoje gotova komercijalna rješenja koja mogu nad podatcima izvoditi po volji složene obradbe? Stvar je u tom da je izgrađivanje računalnoga sustava nešto što većina ljudi, pa ni programera, ne zna; možda i zna izgraditi dva čvora gdje jedan nešto šalje a drugi prima, ali više od toga vjerojatno ne. Upravo ta složenost

postojećih rješenja može odbiti korisnike [26]. To se događa ako se za nešto jednostavno, kao npr. nekoliko pametnih čvorova, zahtijeva instaliranje više različitih programskih potpora, svaku s možda vlastitim operacijskim sustavom (engl. *operating system*, OS), umjesto samo ručno programiranje jedne jednostavne operacije ili u najgorem slučaju instaliranje samo jedne jedine programske potpore.

Primjerice, ako korisnik želi nešto „digitalizirati“, recimo želi mobitelom moći zaključati vrata na stanu, a za to dobije vrlo složene upute kako to učiniti, ne će vidjeti nikakvu prednost IoT-a u tom području primjene, nego će ostati na onom što je bilo prije. Ili će jednostavno čekati da golema potražnja stvori i zadovoljavajuću ponudu gdje je netko riješio taj problem umjesto njega. Naime, izvorno se je očekivala golema digitalizacija svijeta [27], valjda misleći da će svi željeti i sami doprinijeti tomu da pametne stvari prikupljaju podatke o korisniku i djeluju u skladu s tim. Pa ipak, prosječan čovjek ne upotrebljava IoT u svojem privatnom životu [28].

Ključ rješavanja toga svega jest jednostavnost u smislu da se sustav IoT-a može složiti u jedno poslijepodne, a opet da bude takav da se omogući njegov razvoj i njegovo usložnjavanje, kao i svjesnost da će se on vrlo lako moći povezati s drugim sustavima IoT-a upravo u „mrežu svega, uvijek i svagdje“. Svemu tomu pristupa se sa zamišlju da se u model ugrade mnoge podrazumijevane radnje koje se poslije mogu mijenjati.

Jedna od osnovnih pretpostavaka ovoga rada jest da je jednostavnost implementacije sustava put kojim treba krenuti da bi IoT doživio pravo samoostvarivanje. Usporedba se može povući s, recimo, razvitkom Interneta: da se je odmah krenulo sa složenim protokolima, a ne s *internetskim protokolom* (engl. Internet Protocol, IP), pitanje je bi li se Internet bio primio u tolikoj mjeri i tako brzo. A zanimljivo je da i u protokolu IP ima dosta implicitnih mehanizama (npr. kod usmjeravanja) kao i u ovom prijedlogu. Druga se usporedba može povući s jednim skriptnim jezikom koji ima filozofiju „lake stvari trebaju biti jednostavne a teške moguće“. Radi se konkretno [29] o jeziku Perl, ali konkretna zamisao ovdje oblikovana je neovisno o njegovoj sličnoj filozofiji. Sve to ide ruku pod ruku s uštedom energije i ostalim prednostima.

Ostvarivanje opisanih zamisli funkcionira kroz vrlo općenit model čvora i jednaki takav model metaprotokola. Prije toga treba razmotriti što sličnije pokušaje iz literature, odnosno najsličnije tehnologije onima koje su upotrijebljene u predloženim modelima.

U članku [30], gdje su opisane osnovne zamisli predloženoga sustava, može se naći i pregled područja bez usredotočenosti na slične sustave.

2.1. Pregled najčešćih tehnologija IoT-a

Razmotrit će se česta postojeća tehnološka rješenja, posebno ona koja su naišla na veći odziv među korisnicima IoT-a. Krenut će se prvo od protokola na kojima se rješenja najčešće temelje, jer se često cijelo istraživanje svodi na prilagođen model podataka ili prilagođen protokol za njihov prijenos.

Kao osnova IoT-a potrebni su, prvenstveno, bežični, štedljivi, podatkovni protokoli na najnižoj razini – razini lokalne mreže – za komunikaciju između senzora i čvorova koji od njih prikupljaju odčitane podatke. U tu se svrhu često u publicističkoj i znanstvenoj literaturi općenito spominje protokol Bluetooth [31]. Iako protokol Bluetooth izvorno nije bio zamišljen za uštedu energije, našao je široku primjenu u potrošačkim uređajima. Pogotovo ju je našao u mobitelima, pa makar za spajanje s bežičnim slušalicama. Tako je našao i golemu potporu velikih informatičkih društava, i prometnuo se je u česta kandidata i za IoT [32]. To ne bi bilo nikako moguće da od inačice 4.0 nije napisan dio standarda protokola Bluetooth koji se zove *Bluetooth niske energije* (engl. Bluetooth Low Energy, BLE) [31]. Naime, protokol BLE rješava velike preprjeke koje protokol Bluetooth ima za primjenu u IoT-u, a to su veća potrošnja energije, arhitektura tipa gospodar-sluga (engl. *master-slave*) i potrebna jačina procesora.

Osim protokola Bluetooth, odnosno protokola BLE uvijek u IoT-u, sljedeći mogući pristupnik jest protokol *standard 802.11 Zavoda električnih i elektroničkih inženjera* (engl. Institute of Electrical and Electronics Engineers, standard 802.11, IEEE 802.11) [33] (ovdje: protokol Wi-Fi), koji danas podupiru sva prijenosna računala i svi pametni mobiteli. Protokol Wi-Fi jednako kao i protokol Bluetooth nije izvorno namijenjen za takve sustave, pa je i on, iako puno poslije, a možda i prekasno [34], iznjedrio svoj podstandard IEEE 802.11ah [33], koji je njegova prilagodba za uštedu energije, ili efektivno primjenu u sustavima IoT-a.

Organizacija *zavod električnih i elektroničkih inženjera* (engl. Institute of Electrical and Electronics Engineers, IEEE) je predložila i poseban protokol za sustave IoT-a, IEEE 802.15.6 [35]. Taj se protokol službeno predstavlja kao protokol za mreže tipa *mreža područja tijela* (engl. Body Area Network, BAN), što su zapravo lokalne mreže IoT-a.

Organizacije IEEE je protokol i IEEE 802.16 [36] (ovdje: protokol WiMAX), koji je protokol za bežične mreže tipa *mreža područja grada* (engl. Metropolitan Area Network, MAN). Oba su se protokola, svaki u svoja doba, promicala kao prikladna za IoT. Kako IoT, prvo, nije krenuo u smjeru mrežâ MAN, a drugo, prihvaćeni su drugi gore opisani protokoli za male mreže, nijedan nije doživio veliko prihvaćanje i zapravo je protokol BLE zasjenio sve ostale [37]. Za posebno oblikovane sustave (mreže WAN) ne bi bilo dobro ne spomenuti da su neki slični protokoli naišli na bolji odziv. To su primjerice protokol LoRaWAN [38] i protokol Sigfox [39]. Pogleda li se ponuda velikih proizvođača oko modulâ za izgrađivanje sustava IoT-a, može se uočiti da nijedan osim protokola BLE, protokola Wi-Fi i protokola koji će biti obrađeni u upravo sljedećim dvama odlomcima ne dolaze u obzir i za kakvu širu primjenu [40]. Osim toga, kako su i mobiteli velik čimbenik, treba voditi računa i o tom što je poduprto na njima a to su samo protokol Wi-Fi i protokol BLE.

Drugi je podatkovni protokol koji se često upotrebljava protokol IEEE 802.15.4 [41]. Taj se protokol službeno predstavlja kao protokol za mreže tipa *mreža područja osobe* (engl. Personal Area Network, PAN) niske snage (engl. *low-power*), a do kraja teksta će se jednostavno zvati skraćeno „protokol 154“. Protokol 154 doživio je veću primjenu, prvenstveno kao osnova za druge protokole ili protokolne složaje (engl. *protocol stack*). Povijesno prvi je uspjeh doživio kao podatkovni sloj mrežnoga protokola ZigBee [42] za izgradnju lokalnih mreža IoT-a. Iako protokol ZigBee podupire brojne „profile“ (engl. *profile*) iznad sebe, nije zabilježio komercijalni uspjeh kolik je mogao. Ti profili su inače efektivno prijenosni protokoli zasnovani na njem za mrežni sloj i protokolu 154 za podatkovni sloj.

Druga velika suvremena primjena protokola 154 jest za osnovu protokolnoga složaja Thread [43]. Iako se složaj Thread službeno reklamira doslovno kao „*Thread network protocol*“ (mrežni protokol Thread), radi se efektivno o prijenosnom protokolu zasnovanom na postojećem protokolu mrežnoga sloja i postojećem protokolu podatkovnoga sloja. Za podatkovni se upotrebljava protokol 154, a za mrežni protokol *IP inačice 6* (engl. IP version 6, IPv6) i to u obliku *IPv6 za mreže područja osobe niske snage* (engl. IPv6 for Low-Power Personal Area Networks, 6LoWPAN) koji je efektivno njegova kompresija za što je manje moguće pakete protokola 154 u koje se učahuruje [44].

Osim složaja Thread, kao njegov „komercijalni komplement“ u smislu da njegove specifikacije nisu toliko transparentne, kao drugi protokolni složaj koji se snažno reklamira

ističe se složaj Z-Wave [45]. Iza složaja Z-Wave stoje mnogi veliki dionici i u njihovim proizvodima doživio je raširenu primjenu. Važno je istaknuti: iako ni složaj Thread ni složaj Z-Wave nisu ograničeni samo na podatkovni sloj, ipak nisu namijenjeni globalnim mrežama. To, istina, zvuči logično jer se lokalne mreže međusobno najčešće povezuju Internetom. Ali, opet, u njima ne postoji ni nikakvo razmišljanje u tom smjeru globalnoga povezivanja.

Uz to što su ti složaji namijenjeni samo za lokalne mreže i većinom komercijalne proizvode, zahtijevaju i posebnu arhitekturu lokalne mreže. Postoje precizno određene kategorije čvorova, pa nema univerzalnosti u smislu da se npr. složaj Thread može upotrebljavati bez ikakve lokalne mrežne infrastrukture između dvaju čvorova.

Sad će se razmotriti protokoli prijenosnoga sloja, a kako se IoT danas povezuje Internetom, radi se o *protokolu kontrole prijenosa* (engl. Transport Control Protocol, TCP) i *protokolu korisničkih datagrama* (engl. User Datagram Protocol, UDP). Kad se upotrebljava protokol IP, to može biti protokol *IP inačice 4* (engl. IP version 4, IPv4) ili protokol IPv6. Protokol IPv6 omogućuje jedinstvenu identifikaciju čvora zbog četverostruko veće adrese od protokola IPv4 [46]. Njegova brzina širenja primjene u Internetu stagnira pa se upotrebljavaju oba [47]. Za protokol IP su svojedobno izgrađivani i sustavi preslikavanja adresa kao što je „HAT“ [48], ali nisu doživjeli veću primjenu. Sličnu su sudbinu doživjeli i pokušaji izgrađivanja nova, samo mrežnoga protokola, za Internet s naglaskom na IoT – kao što je „MobilityFirst“ [49].

Na protokolu TCP se zasnivaju najčešći spominjani primjenski protokoli IoT-a [50], a to su redom *primjenski protokol za ograničene sustave* (engl. Constrained Application Protocol, CoAP) [51], *proširivi protokol za razmjenu poruka i nazočnost* (engl. Extensible Messaging and Presence Protocol, XMPP) [52], *prijenos telemetrije redova poruka* (engl. Message Queue Telemetry Transport, MQTT) [53], *napredni protokol za redove poruka* (engl. Advanced Message Queuing Protocol, AMQP) [54], ostali modeli tipa RESTful (modeli čiji je komunikacijski model *prijenos reprezentativnoga stanja* (engl. Representational State Transfer, REST)) [55]. Protokol CoAP se u literaturi često spominje jer se radi o prilagodbi *protokola prijenosa hiperteksta* (engl. Hypertext Transfer Protocol, HTTP) za ograničene sustave pa je „bliži“ korisnicima svojim komunikacijskim modelom. Jedna je od zamisli protokola CoAP binarno kodiranje, što je predloženo i u ovom radu. Od ostalih se protokol MQTT u praksi dosta upotrebljava, a primjer se može naći u napisanom pregledu literature.

Usprkos množini protokola, svi su ti protokoli vrlo slični i dosta različiti od prijedloga metaprotokola u ovom radu. Naime, slični su po tom što svi imaju samo model objavi-pretplati (engl. *publish-subscribe*) u smislu da su svi arhitekture klijent-poslužitelj (engl. *client-server*) pa tako se razlikuju od metaprotokola po tom što ne ostavljaju mogućnost izgradnje drugačijih mreža, čak i kad se kombiniraju sa složajem TCP/IP. To jest, u tom slučaju složaj TCP/IP može pružiti mogućnost da su klijent i poslužitelj na različitim stranama svijeta, ali ne i da se ostvaruje ravnopravna mreža. Isto tako, višerazinska hijerarhija gdje makar poslužitelj može biti klijent nekomu drugomu čvoru nije lako ostvariva.

Ni jedan od spomenutih protokola ne može ponuditi istodobno rješenje koje bi bilo dobro za više razina. Predloženi se metaprotokol zato može osloniti na neki drugi koji upotrebljava za prijenos poruka. On zapravo uzima sve što može i što mu je potrebno od postojećih protokola i samo dodaje stvari koje nedostaju u konkretnom trenutku.

Metaprotokol se temelji na jeziku SQL, što odudara od drugih rješenja. Pokušaj usporedbe tzv. „metaprotokola“ temeljena na programskom jeziku ne nalazi sličnosti u literaturi. Tražeći najsličniju stvar, njom se čini tehnologija *protokol i upitni jezik za RDF zvan SPARQL* (engl. SPARQL Protocol and RDF Query Language, SPARQL) [56]. Jezik SPARQL je već svojim nazivom definiran kao upitni jezik kao i jezik SQL, ali ne radi s bazama nego sa skupovima podataka u obliku *okvira opisa sredstava* (engl. Resource Description Framework, RDF) za tzv. semantički *web* [57]. Neke su naredbe slične, recimo relacijska projekcija podataka. Ali, kako se ne radi o relacijskom modelu, i to se radi o modelu kombiniranu s tipovima okvira RDF, a ne standardnim tipovima podataka, uočavaju se i velike razlike [58]. Naime, jezik SPARQL radi na modelu okvira RDF, kombiniranu s podskupom „standardnih“ tipova.

Uočava se da je potreba učenja posebna jezika za rad s sustavom IoT-a preprjeka koja treba biti vrlo dobro argumentirana da bi se stavila pred korisnika. Radi li se o specijaliziranu jeziku kao u [59] koji se ne može upotrijebiti ni za što drugo osim te jedne primjene, argumentacija je skoro nemoguća. Zato se ovdje upotrebljava jezik SQL kao najbolje od oba svijeta. S jedne strane, to je Turing-potpun programski jezik [60] što specijalizirani jezici često nisu, a s druge, jezik prilagođen modelu podataka: može se reći i model prilagođen jeziku.

Kako se pokazuje da arhitektura temeljenih na sličnom načinu komunikacije nema, dat će se dalje usporedba s arhitekturama temeljenima na sustavu pravila, što je drugi temelj predložene arhitekture. Arhitektura ima dosta i „sustav pravila“ u literaturi jest termin koji se često spominje. U nastavku će se obraditi deset takvih arhitektura koje su slične predloženoj.

2.2. Arhitekture temeljene na sustavu pravila

U sljedećim arhitekturama naglasak će biti na opisu stvari specifičnih za njih. Vrijedi li nešto za sve ili skoro sve opisane arhitekture (npr. da se ne vodi računa o uštedi energije), to se ne će posebno bilježiti. U svakom slučaju, prednosti i nedostatci, bez obzira na to vrijede li za sve ili samo za neke, obradit će se potanje poslije njihova opisa.

U [61] se prikazuje arhitektura u kojoj usmjernik komunicira s nižim čvorovima-uređajima modelom REST preko različitih podatkovnih protokola. Čvorovi mogu primati određene podatke-naredbe, a mogu biti i pitani za određene podatke. Korisnik koji upotrebljava usmjernik mora naučiti poseban jezik za sustav pravila, a taj se jezik poslije prevodi u programski jezik Java i izvršava nad sustavom.

U [62] pravila u usmjerniku pišu se u skriptnom jeziku Lua, a ona rade sa semantičkim pogoniteljem (engl. *semantic engine*) koji se na usmjerniku izvršava. On komunicira s nižim uređajima preko određenih protokola i nad podatcima koje od njih dobiva izvodi zaključke temeljene na svojim ontologijama. Pravila onda rukuju podatcima na višoj razini, a uređuju se preko konfiguracije za *web*, slično kao i u predloženom sustavu iz disertacije.

U [63] korisnici stvaraju sustav IoT-a sklapanjem jednoga po jednoga pametnoga čvora, a čvor se slaže tako da se preko grafičkoga sučelja bira njegov oblik, njegovi dijelovi i njegovi obrasci ponašanja. Ti se obrasci sastoje od skupova pravila koja upravljaju tim dijelovima, odnosno nekim već gotovim funkcionalnostima tih dijelova, a prije izvršavanja se prevode u programski jezik Java kao u [61]. Ta su pravila u obliku „`kad se dobije neko odčitavanje, obavi određenu radnju u čvoru`“.

U [64] se upotrebljava viši čvor koji rukuje sustavom pravila definiranoga grafičkim sučeljem, a u tom se sučelju pravila izgrađuju slaganjem postojećih ponuđenih mogućnosti, zvanih ondje i servisima. Te ponuđene mogućnosti rukuju podatcima, i to onima prikupljenima od hijerarhijski nižih stvari, onima njima poslanima, kao i dodatnim

metapodatcima koji se automatski zaključuju o tim stvarima, slično opisanim ontologijama u [62]. Stvari se apstrahiraju dodatnim slojem, tako da se ni ovdje ne rukuje čistim porukama.

U [65] čvorovi komuniciraju isključivo ili protokolom HTTP ili preko posrednika (engl. *proxy*) protokola HTTP, a radnje koje se izvode u obliku su REST, pa zapravo zajedno tvore ono što se često zove „*web* stvarî“ (engl. Web of Things, WoT) u lokalnoj mreži. Mreža WoT je sustav IoT-a u kojem se komunicira tehnologijama za *web*, npr. protokolom HTTP. Pravila se pišu u skriptnom jeziku JavaScript, ali njihovi se okidači pišu posebno u *Javascriptovu objektnom zapisu* (engl. JavaScript Object Notation, JSON). Ta pravila mogu rukovati podatcima od drugih stvari ili već postojećim podatcima, a služe za generiranje događajâ u posebnom modulu za obavljanje operacija, koji se po potrebi može nadzirati i u stvarnom vremenu.

U [66] se u sustavu mogu za pravila definirati okidači različitih razina apstrakcije, koji, potpomognuti „kontekstualnim“ podatcima, rukuju ontološkom reprezentacijom hijerarhijski nižih uređaja razvrstanih po kategorijama i ponuđenih usluga tih uređaja razvrstanih po mogućnostima. U jednom projektu iz toga članka radi se na automatskom stvaranju prikladna grafičkoga sučelja za uređaje, a u drugom projektu na automatskom predlaganju gotovih pravila korisniku.

U [67] se kreće od gotova sustava pravila za neki sustav IoT-a, koja se prvo raščlanjuju i opisuju određenim formalnim modelima. Onda se, preko tu naučena preslikavanja uređaja i pravilâ, automatski, kad se nov uređaj doda u sustav, raščlanjuju njegove funkcionalnosti i automatski izrađuju pravila, koja se onda preporučuju korisniku toga sustava.

U [68] se opisuje sustav gdje se u višem čvoru grafički izrađuju i uređuju pravila za taj sustav, a pravila se ostvaruju u jeziku JavaScript i tehnologiji jQuery, što je dodatak za taj jezik. Pravila rukuju svojim pogoniteljem (engl. *engine*) koji može primati događaje, pohranjivati podatke i konačno izvoditi neke radnje. Taj pogonitelj komunicira sa svojim nižim čvorovima preko različitih protokola.

U [69] usmjernici pohranjuju podatke od nižih uređaja tako da se podatci posebno obilježuju vremenski (engl. *timestamping*), slično kao i u predloženoj arhitekturi. Ti se uređaji apstrahiraju tzv. „avatarima“, koji se opisuju ontologijama. Iznad usmjernikâ može postojati i viši sloj, moguće i samo korisnički, koji može pitati usmjernike različite stvari preko posebnoga usmjerničkoga upitnoga jezika, na što usmjernici odgovaraju podatcima.

U [70] se u „oblaku“ (engl. *cloud*) mogu definirati pravila tipa *događaj-uvjet-radnja* (engl. Event-Condition-Action, ECA), i to preko grafičkoga uređivača, čija se pravila prvo opisuju posebnim jezikom pravila za tu primjenu, pa onda prevode u jedan drugi jezik pravila, zvani Drools [71], za koji se u radu tvrdi da je popularniji. Taj oblak prima podatke od „stvarî“ preko njihovih lokalnih mreža. On isto tako može i pozivati (*pushati*) neke radnje u aplikaciji koja se na nj tad spoji.

U nastavku su dodatno istaknuti neki problemi prikazanih rješenja. Prvo, mnogi su sustavi razvijeni samo za jednu svrhu i njihova primjena nije zamišljena i za koji drugi scenarij. To ne znači da se oni nikako ne mogu nigdje drugdje primijeniti, ali bi zahtijevala raščlambu slučaja i moguću prilagodbu. Tako su i sustavi [63,70] namijenjeni konkretno pametnomu domu (engl. *home automation*), odnosno atletičkomu treniranju.

Drugo, mnogi sustavi zahtijevaju da korisnik nauči poseban jezik da bi mogao njima rukovati. To je najčešće jezik pravilâ, koji može biti neki izmišljeni jezik, neki skriptni jezik kao npr. Lua ili neki programski jezik kao npr. Java. Tako [61,62] djelomično, a [65,69] potpuno to zahtijevaju. Naime, bez toga se ne može služiti sustavom, tj. ne na smislen način. Konkretno to su redom jezici Lua, JavaScript, Java i Drools.

Treće, svi, osim donekle sustav opisan u [61], navedeni sustavi namijenjeni su samo za lokalne mreže. Dakle, pojednostavljeno, usmjernik kupi podatke od senzorâ i dostavlja ih korisniku. Ne postoji mogućnost ujedinjivanja više lokalnih mreža u globalni IoT, bilo Internetom ili kako drugačije. Iako se Internet može upotrebljavati za korisnikov pristup vršnomu čvoru u sustavu, tu nije riječ o ujedinjivanju.

Četvrto, svi navedeni sustavi imaju dodatan apstrakcijski sloj za komunikaciju. Naime, konkretan izgled poruka često nije opisan, nego se poruke razmatraju apstrahirano u smislu potrebnih podataka koji se trebaju prenijeti. Kako će ti podatci biti zapisani se ne opisuje u člancima i pretpostavlja se da će se upotrebljavati neki prikladan način prijenosa poruka. Zapravo, u člancima se cijela komunikacija razmatra tako: da je ona već nekako riješena, a kako je riješena nije važno za članak, tj. u optimizaciju komunikacije nije uložen znatan trud što se ne može reći za predloženi sustav.

Peto, svi navedeni sustavi, osim donekle [62], imaju kruto odvojene kategorije svojih čvorova. Recimo, imaju senzore, usmjernike i poslužitelje. Čvor mora upasti u neku od kategorija, jer inače nije predviđeno rukovanje nečim što ne zadovoljava te točno određene uvjete.

Šesto, ni jedan navedeni sustav ne podupire dinamičku promjenu uloge ili složenosti čvora samo promjenom svojstava. Najčešće se radi o potpuno različitim programskim potporama bez mogućnosti da jedna potpora ima više uloga. Nije često riječ o tom da su te potpore najjednostavniji programi koji i ne mogu imati neke naprednije mogućnosti, nego o tom da se za svaku kategoriju čvorova izrađuje specifična složena potpora.

Konačno, ubaciti jednostavan čvor i onda ga dotjeravati jednom po jednom mogućnošću nije moguće ili nije praktično ni u jednom od spomenutih sustava. Osim toga, ušteda energije i sigurnost nisu razmatrane ni u jednom od tih sustava dokle se u predloženom one razmatraju od začetka.

Kad se usporedi predloženi sustav s navedenim sustavima, uočavaju se velike razlike pa, u slučajima kad korisnik neku od navedenih značajka smatra nepoželjnom, predloženi sustav može biti bolje rješenje. Slijedi usporedba predloženoga sustava po tih sedam stavaka.

Prvo, predloženi je sustav zamišljen za mreže IoT-a opće namjene, a ne za jednu jednostavnu primjenu. Model je podataka dovoljno razrađen tako da se ne usredotočuje samo na jedno područje primjene nego se gleda sveopća primjena.

Drugo, umjesto učenja posebna jezika, upotrebljava se samo jezik SQL, odnosno neki njegovi dijelovi u smislu pisanja izraza jezika SQL. Osim izvođenja upitâ jezika SQL neke složenije mogućnosti mogu u sustavu biti dostupne kao vanjski program: taj vanjski program može rukovati i samo bazom, može npr. komunicirati s vanjskim uređajima. Ti vanjski programi mogu biti i napisane naredbe za naredbenu ljusku, a najčešće bi bili nešto kao „`pročitaj senzor`“.

Dodatno, neke česte složene operacije dostupne su kao posebne naredbe napisane među naredbama jezika SQL, primjerice naredba `ENCRYPT` (hrv. *kriptiraj*) i naredba `SIGN` (hrv. *potpiši*). Osim toga, može se izraditi i grafičko sučelje za stvaranje pravila koje će dodatno pojednostaviti uporabu u smislu minimizacije ikakva programiranja.

Treće, nema apstrakcijskoga sloja u porukama nego se porukama izravno rukuje implicitnim ili eksplicitnim pravilima. Iako to može otežati eksplicitno rukovanje – recimo, nije prejednostavno ručno izvući podatak iz tijela poruke – nije smisao sustava da se pozivaju operatori jezika SQL za rad sa znakovnim nizom u tijelu poruke. Naime, ako se treba dogoditi neka radnja na neki podatak, puno je zahvalnije postaviti okidač koji će ju izvoditi. To može biti jednostavna pretplata na odgovarajuće podatke u drugom čvoru ili

čak istom u posebno konfiguriranom sustavu; može biti ručno postavljen okidač jezika SQL; mogu se pregledavati tablice ručno i tako odlučivati, itd.

Dakle, poruke se prenose preko čvorova i svakakvih mreža nepromijenjene osim ako nisu postavljena pravila koja ih mijenjaju. Primjerice, ako se poruke kriptiraju ili potpisuju, očito se njihov oblik mijenja, ali sadržaj, koji u konačnici dobije odredišni čvor, nakon dekriptiranja ostaje jednak početnomu poslanomu sadržaju.

Četvrto, mijenjanje postavaka čvora može biti vrlo jednostavno. Mijenjanje se izvodi bazom ili korisničkim sučeljem, a može se uz dodatno programiranje i namjestiti automatska promjena kad dođe odgovarajuća poruka. Na taj se način zapravo konfiguracija čvora može slati i preko Interneta; ovdje taj pristup nije razrađen. Sustav je pravila isto tablica u bazi i može se mijenjati, ako su omogućena pristupna prava nad njom tomu korisniku. U svakom slučaju, svim se opisanim mogu čvoru i dodavati i oduzimati funkcionalnosti. Tako se može cijeli sustav pomalo mijenjati, evoluirati, kad se promijene zahtjevi.

3. MODEL ARHITEKTURE

Model predložene arhitekture opisuje se njezinim čvorovima, povezivošću između tih čvorova i operacijama tih čvorova. Prvo će se opisati tri „tipa“ čvorova koji se mogu pojaviti, ovisno o načinu na koji se uključuju u sustav. Tipovi nisu povezani s tradicionalnim kategorijama jer ne moraju svi postojati u sustavu.

Prvi i najvažniji tip čvora je „metaprotokolski čvor“, temelj ove arhitekture. Takvi čvorovi upotrebljavaju „metaprotokol“ za razmjenu podataka i operacija, a oko njih će se uključiti drugi tipovi čvorova. U skladu s ulogom u sustavu metaprotokolski čvorovi mogu biti vrlo jednostavni, odnosno čvorovi ograničeni sredstvima koji samo s vremena na vrijeme nešto pošalju, ili mogu i primiti neke jednostavnije poruke. Isto tako mogu biti i složeniji. Pa tako to mogu biti čvorovi koji osim uporabe svojih senzora ili svojih aktuatora možda mogu i prosljeđivati poruke. Neki će moći i mijenjati poruke prije prosljeđivanja uz pomoć različitih implicitnih pravila, ili slanjem preko različitih nižih protokola. Još složeniji čvorovi mogu biti vrlo napredni, i preko eksplicitnih pravila moći će rukovati porukama, pohranjivati ih, dohvaćati, obavljati neke operacije nad skupom podataka iz poruka, upotrebljavati sigurnosne mehanizme povjerljivosti (engl. *confidentiality*) i autentičnosti (engl. *authenticity*), itd.

Ovdje navedene složenosti čvorova mogu se donekle povezati s opisom čvorova IoT-a iz standarda *zahtjev za komentare, broj 7228* (engl. Request for Comments, number 7228, RFC 7228) [71] koji uvodi određeno nazivlje za „mreže ograničene sredstvima“ (engl. *resource-constrained networks*). On opisuje i određene razrede čvorova po složenosti, gdje je razred 0 (engl. Class 0) sredstvima najograničeniji razred a razred 2 (engl. *Class 2*) najmanje ograničen iako još uvijek ne pretjerano napredan. Kad se to uzme u obzir, može se reći da, kad se god u radu spomenu „sredstvima najograničeniji“ čvorovi, oni se mogu poistovjetiti s razredom 0. S druge strane, kad se govori o metaprotokolskim čvorovima, ograničenosti ne vrijede. Oni mogu biti razreda 0, 1 (engl. *Class 1*), 2 ili čak jači od razreda 2. Poenta arhitekture jest da čvor može biti po volji jednostavan ili složen ali svejedno može na neki način sudjelovati u sustavu uporabom metaprotokola.

Drugi tip čvora može se zvati „korisničkim čvorom“. Takav čvor pristupa podatcima i operacijama metaprotokolskih čvorova, ali preko drugih sučelja neovisnih o

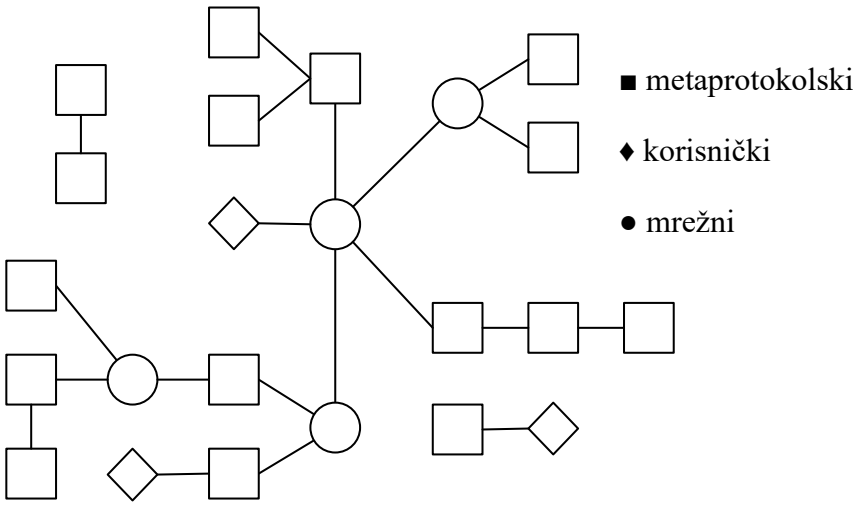
metaprotokolu. Npr., to može biti sučelje za *web* preko kojega korisnici ili administratori mogu raditi s čvorom, kao što je to i u prototipu implementirano.

Zadnji, treći, tip čvorova ovdje će se zvati „mrežnim čvorom“. On je jednostavno dio mrežne infrastrukture sustava koji se ostvaruje, kao što su usmjernici (engl. *router*), prespojnici (engl. *switch*), obnavljači (engl. *repeater*) i sl. Oni omogućuju komunikaciju nižim protokolima kojima se metaprotokol prenosi i ne znaju ništa o metaprotokolu, niti bi morali znati.

Dva čvora koja se u sustavu povezuju tu povezanost, odnosno razmjenu poruka, mogu ostvariti na četiri načina:

1. izravnom fizičkom vezom, žično ili bežično, upotrebljavajući niže protokole u koje se metaprotokol učahuruje
2. preko nekoga metaprotokolskoga čvora, pri čem se poruka može samo proslijediti, ali se može i izmijeniti
3. preko nekoga mrežnoga čvora koji zna samo za niže protokole kojima se prenosi poruka
4. preko više međučvorova, gdje neki mogu biti metaprotokolski, a neki ne.

Sl. 3.1 prikazuje općenit primjer skupa čvorova koji se na ovaj način povezuju. Metaprotokolski su čvorovi prikazani kao kvadrati, korisnički čvorovi su rombovi i mrežni su čvorovi krugovi.



Slika 3.1. Primjer arhitekture

U ovakvu se sustavu svaki čvor može po potrebi povezati s nekim drugim čvorom. Osim toga, mogu postojati i izolirani čvorovi, kao na Sl. 3.1, koji neizravno sudjeluju u radu

sustava kroz npr. promjene u okolini. Njihovi senzori mogu te promjene otkriti kad ih drugi čvorovi učine, ili obrnuto, uz pomoć aktuatora mogu ih oni učiniti.

Ponašanje sustava definirano je samo metaprotokolskim čvorovima. U daljem tekstu samo se oni modeliraju i kad god se spomene „čvor“ misli se upravo na metaprotokolski čvor.

Ostatak modela arhitekture opisan je najprije predloženim metaprotokolom i predloženim sustavom pravila. Metaprotokol se može opisati svojim porukama i rukovanjem tim porukama, kao i njihovim kraćenjem. Konfiguracija čvora opisuje se sustavom pravila i ostalim postavkama koje se mogu mijenjati u čvoru, kao i njihovim podatcima.

U predloženom modelu ima više tipova poruka i one na relacijskom modelu podataka izvršavaju tekstualni ili binarni protokol. U tom je protokolu od jezika SQL uzeta samo operacija projekcije, to jest naredba `SELECT`, i sintaksa tipova podataka. Možda je najbolje reći da je temelj svega naredba `SELECT` i odgovor te naredbe, odnosno operacija zahtijevanja i slanja različitih podataka. To jest, model koji se može zvati „porini-povuci“ (engl. *push-pull*).

Poruke metaprotokola ostvaruju određene radnje u sustavu IoT-a, a glavna im je svrha komunikacija između različitih kategorija čvorova. To su najčešće, prvo, čvorovi koji proizvode podatke, a obično se zovu jednostavno „stvari“. Dalje su to čvorovi koji prosljeđuju podatke, a obično se zovu „*gatewayi*“, a u nedostatku boljega prijevoda ovdje će se do kraja teksta zvati usmjernicima. Onda dođu čvorovi koji obrađuju i pohranjuju podatke, a obično se zovu „poslužitelji“ – engl. *server*, i konačno čvorovi koji traže te podatke za neku svrhu, a obično se zovu „klijenti“ – engl. *client*, a mogu se zvati i korisnicima ili korisničkim primjenskim programima (engl. *user application*). Te se kategorije odnose na spominjanu tradicionalnu „četveroslojnu“ arhitekturu.

Kao važan vid komunikacije ističe se i mehanizam „objavi-pretplati“, koji je dio metaprotokola, što treba i očekivati u sustavu IoT-a [73]. Moguće je nekomu čvoru poslati zahtjev za pretplatu kojim se definira koji ga događaji zanimaju. Kad se takvi događaji dogode u tom drugom čvoru, on treba odgovoriti prvomu prikladnom porukom (podatcima). Svi protokoli IoT-a imaju mehanizam objavi-pretplati, ali se u predloženom modelu upotrebljava jezik SQL za definiciju događaja za pretplatu, što daje vrlo veliku fleksibilnost koju drugi sustavi nemaju.

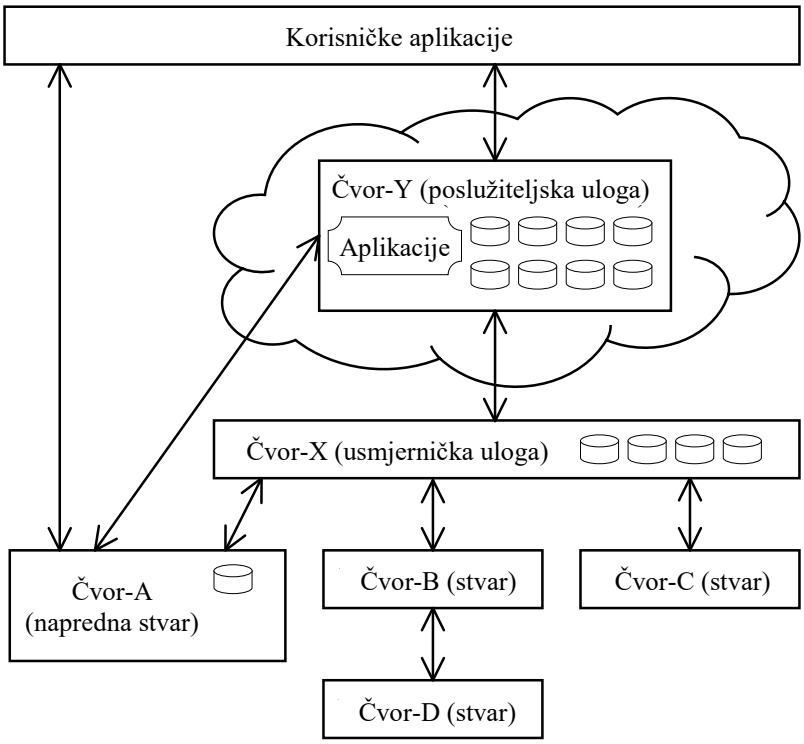
Ne postoji nikakav fiksni, kruti, tip arhitekture koja se ostvaruje ovdje opisanim modelom. Naime, u njem ne postoji podjela na vrste čvorova opisane gore, jer iako svaki čvor može

imati ulogu „stvari“, „usmjernika“, „poslužitelja“ ili „klijenta“, može isto tako imati i dio jedne uloge ili kombinaciju uloga. Zamišljeno je da se sve te uloge mogu ujediniti jednim modelom čvora iz kojega proizlaze različiti skupovi funkcionalnostî. Ti skupovi funkcionalnosti mogu se onda zvati „ulogama“ (engl. *role*) ili konkretno npr. „ulogom stvari“, „ulogom klijenta“ itd.

Ukratko, danas je pretežan tip arhitekture IoT-a upravo taj „stvar-usmjernik-poslužitelj-klijent“. Iako se takav model može ostvariti i u ovom sustavu, zapravo se ovim sustavom može ostvariti bilo koji tip arhitekture. Npr., može u sustavu biti samo jedan čvor koji šalje podatke i jedan koji ih prima, a može biti i više čvorova od kojih cijelim skupinama upravlja po jedan nadčvor, a onda nadčvori mogu slati podatke u oblak ili skup oblaka od kojih svaki ima više korisnika, itd. Svaki je čvor sam po sebi ravnopravan i jedinstven, a suradnja s drugim čvorovima definira se njegovom konfiguracijom: ona može biti po volji jednostavna ili složena.

Konfiguracija može biti ostvarena i ručnim programiranjem čvora tako da on komunicira metaprotokolom ili njegovim dijelom. Model čvora svojom generičnošću dopušta da različiti čvorovi imaju različitu programsku potporu, dokle god ona upotrebljava isti metaprotokol.

Primjer jedne naprednije arhitekture koja se ostvaruje ovim modelom prikazan je na Sl. 3.2. Ovdje se mogu vidjeti čvorovi različitih kategorija kako komuniciraju jedan s drugim i odrađuju različite zadaće u skladu sa svojim ulogama. Iako su na njoj prikazane sve tradicionalne uloge čvorova, dopušta se i, npr., da usmjernici preuzimaju dio posla od oblaka, ili da stvari izravno komuniciraju s oblakom, ako su za to sposobne. Tradicionalne su te uloge „stvari“, „usmjernici“… Zapravo, nema preprjeka da itko preuzme koji bilo dio koje bilo uloge ili da komunicira s kim bilo u kojem bilo smjeru – što je nešto jedinstveno za ovaj model. Na ovom primjeru sustava može i prikazana “napredna stvar” upravljati svim ostalim stvarima, a ne mora upravljanje dolaziti odozgora.



Slika 3.2. Primjer sustava IoT-a

Sve opisano omogućeno je općenitim sveopćim modelom podataka i definiranjem unaprijed tko treba s kim kako komunicirati. Treba tu naglasiti da nije potrebno unaprijed precizno znati koji će čvor sudjelovati u komunikaciji da bi se drugi čvorovi ispravno konfigurirali, nego samo koje će se „uloge“ gdje pojaviti. Mnoga su implicitna pravila ugrađena u sustav, i posebna konfiguracija treba samo u slučaju da ona nisu dovoljna.

Slijedi opis poduprtih operacija IoT-a pa poruka koje iz njega proizlaze, onda ide model samoga čvora i sustav pravila ostvaren u čvoru. Implementacijske potankosti spomenut će se samo ondje gdje to bude prijeko potrebno.

3.1. Radnje Interneta stvari

Kao sastavni dio arhitekture osmišljen je skup operacija koje su se željele poduprijeti u predloženom sustavu. Skup je operacija oblikovan usporedbom arhitektura iz literature iz kojih su izlučene operacije potrebne za svaki sustav IoT-a sa širom primjenom. Iz skupa operacija proizlaze i poruke metaprotokola za prijenos podataka i poslije opisana konfiguracija čvora. Ukupno je izlučeno dvanaest operacija, i još dvije „podoperacije“. Slijedi njihov opis.

1) Poslati podatke drugomu čvoru.

1') Poslati 1-bajtne podatke drugomu čvoru sa što je manje moguće suvišnih operacija (engl. *overhead*).

2) Pitati odnosno tražiti podatke od drugoga čvora. Traženi podatci mogu se moći lako dohvatiti ili mogu moći uključivati dodatne radnje.

   2') Pitati podatke preoblikovane tzv. osnovnim operacijama (engl. *basic operation*) od drugoga čvora.

3) Pretplatiti se (engl. *subscribe*) na podatke od drugoga čvora.
4) Otkazati pretplatu (engl. *unsubscribe*).
5) Ostvariti povjerljivost, gdje će biti dovoljno kriptirati poruku.
6) Ostvariti autentičnost, gdje će biti dovoljno potpisati poruku.
7) Potvrditi primanje (engl. *acknowledgement*) poruke.
8) Javiti pogrješku u poruki.
9) Upravljati informacijama o jednom čvoru u drugom čvoru, uključujući registraciju i deregistraciju.
10) Upravljati podatcima iz poruka jednoga čvora u drugom čvoru.
11) Postaviti okidače za radnje prigodom primanja poruka, prigodom slanja poruka i vremenske.
12) Upravljati autorizacijom jednoga čvora u drugom čvoru.

Veza jednoga čvora s drugim čvorom može biti izravna, gdje poruka izravno dolazi od izvorišnoga čvora do odredišnoga upotrebljavajući niže protokole od metaprotokola, ili neizravna. U neizravnoj u komunikaciji sudjeluju drugi čvorovi koji upotrebljavaju metaprotokol te prosljeđuju poruku u izravnom ili promijenjenom obliku dalje prema odredišnomu čvoru. Ti drugi čvorovi ovdje se zovu „međučvorovima“. U ovom se opisu ovih radnja „podatci“ odnose na korisne podatke IoT-a, a ne na recimo podatke protokola TCP. „Poruka“ se isto tako odnosi samo na poruku IoT-a, a ne recimo na poruku *potvrda* (engl. Acknowledgment, ACK) protokola TCP. „Čvor“ se odnosi na „pametni“ čvor IoT-a, a ne na recimo usmjernik protokola IP. „Međučvor“ isto tako na međučvor IoT-a, što je čvor IoT-a preko kojega se šalje poruka (engl. *interconnecting node*), a ne npr. prespojnik protokola Ethernet. „Okidači za radnje“ odnose se na smislene radnje IoT-a, kao što su slanje neke poruke, pokretanje nekoga programa, promjena neke konfiguracije… a ne na recimo podrazumijevanu jednostavnu obradbu poruka. Jednako vrijedi i za „uključivane dodatne radnje“. Od 12 opisanih operacija može se vrlo grubo reći da se prvih 8 odnosi na

metaprotokol a druge 4 na konfiguraciju, ali pošto se opiše i jedno i drugo, svaka će se stavka proći potanje.

Pod „povjerljivost“ i „autentičnost“ misli se na pojme iz protokolâ podatkovnoga sloja; dakle, povjerljivost uključuje i besprijekornost (engl. *integrity*), ali ne i neporecivost (engl. *nonrepudiation*). Pod „pogrješke u porukama“, nadalje, može biti više vrsta pogrješaka, od kojih neke možda i ne treba javljati. Recimo, zlonamjerno izobličenu (engl. *malformed*) poruku treba jednostavno odbaciti. Potanje u potpoglavlju 3.11.

Ove dvije „podradnje“ obilježene apostrofom više su kao dodatan zahtjev na njima prethodne, o kojem treba razmišljati zajedno s tima prethodnima. Pod „pretplata“, „otkazivanje pretplate“, misli se na ono što se u literaturi zove „model pretplati-objavi“.

Ove se radnje IoT-a ne moraju sve ostvariti u jednom čvoru, nego mogu proizlaziti iz suradnje više čvorova. Jednostavniji čvorovi oslanjat će se tako na rad složenijih čvorova. Naravno, najsloženiji čvorovi kao npr. oblak mogu imati ostvarene sve mogućnosti, osim možda nekih specifičnih za podatkovni sloj. Ne moraju sve radnje biti potrebne u svim sustavima.

Isto tako, ne moraju sve radnje biti jednako jednostavne za ostvarivanje u svakom čvoru. Neke će možda čvor sam po sebi podupirati a neke će korisnik sam konfigurirati. Opet, to ovisi i o tom što korisniku u kojem čvoru treba.

3.2. Model podataka

Model podataka koji se upotrebljava u arhitekturi temelji se na relacijskim bazama i djelomično je nadahnut modelom podataka u popularnim komercijalnim rješenjima. Naime, u nekim se rješenjima sve pohranjuje u bazi na oblaku, slično kao ovdje, a onda se toj bazi pristupa kao i svakoj drugoj bazi [74].

Svaki čvor ima svoje podatke. Ti podatci mogu biti pohranjeni samo u tom čvoru – ili njihova preslika ili kao dio svoje preslike, možda i izmijenjeni, mogu biti pohranjeni u nekom drugom čvoru. Podatci mogu biti i raspodijeljeni na više čvorova, a postoje i druge kombinacije. U svakom slučaju, svaki čvor može čuvati podatke svakoga čvora, ako je za to sposoban, to jest, ako ima bazu ili neku njezinu zamjenu (engl. *counterpart*).

U čvoru koji ima podatke drugih čvorova ti se podatci moraju nalaziti u tablici jer se operacije temelje na relacijskom modelu podataka. Naziv te tablice mora nedvosmisleno

identificirati taj drugi čvor. Za identifikaciju se čvorova u predloženom sustavu upotrebljava *prošireni jedinstveni identifikator, 64-bitan* (engl. Extended Unique Identifier, 64-bit, EUI-64) [75].

Identifikator EUI-64 se upotrebljava zato što se može vrlo jednostavno „izvući“ iz nižih protokola koji prenose predloženi metaprotokol, i to na jedinstven način. Pri tom se ne misli na jedinstvenost u smislu kao što npr. u protokolu IP kombinacija adrese protokola IPv4, identifikatora višega protokola Protocol (hrv. *protokol*) i vratâ (engl. *port*) čine jedinstvenu kombinaciju, nego jedinstvenost u smislu da različiti protokoli ne mogu generirati jednak identifikator za različite strojeve. Konkretno, npr. protokol BLE često upotrebljava [31] *prošireni jedinstveni identifikator, 48-bitan* (engl. Extended Unique Identifier, 48 bit, EUI-48) [75] koji se može lako proširiti u identifikator EUI-64, protokol 154 često upotrebljava [41] baš identifikator EUI-64, svako računalo koje je spojeno na Internet ima adresu tipa *kontrola pristupa mediju* (engl. Media Access Control, MAC) koja je 48-bitna, ali koja se isto može proširiti u identifikator EUI-64 na isti način [75], a jedinstvena je i nedvosmisleno identificira mrežnu karticu na kojoj se upotrebljava [76]. Osim toga, i protokol IPv6 upotrebljava identifikator EUI-64 [77].

Jedan čvor može imati više mrežnih kartica, recimo, jednu za protokol Wi-Fi, jednu za protokol Ethernet, jednu za protokol BLE, jednu za protokol 154… Tako može imati i, logično, više mogućih pristupnika za identifikator EUI-64. U svrhu što češćega ispuštanja adresa iz predloženoga metaprotokola odlučeno je da su svi oni jednakovrijedni za naslovljavanje čvora. Za naslovljavanje podataka toga čvora treba ili paziti ili postaviti pravila, npr. za preimenovanje tablica u upitu, koja će voditi računa o tom.

Dakle, postoji veza „*N* na 1“ između identifikatora i čvora, u smislu da identifikator sigurno identificira čvor jer se identifikatori EUI-64 različitih vrsta prilagodnika ne preklapaju. Ali svaki čvor može imati više identifikatora. To znači da neki čvor može imati i više tablica s više identifikatora EUI-64 koje čuvaju podatke istoga čvora. Osim takvih tablica mogu postojati i druge tablice, ali se one drugačije zovu.

Naziv tablice koja je povezana s nekim identifikatorom EUI-64 glasi „`t`<16 heksadekadskih znamenaka malim slovima>“. Naziv tablice u sustavima za baze (engl. *database management system*, DBMS) ne može počinjati brojem, a „`t`“ je kratica za „`table`“ (hrv. *tablica*) ili „`tablica`“, tako da naziv bude što kraći a opet valjan po standardu jezika SQL. Tako 16 heksadekadskih znamenaka obilježava 8 bajtova u koje se

pohranjuje identifikator EUI-64. Primjerice, ako je identifikator EUI-64 čvora vrijednost `0xabcdefabcdefabcd` onda je naziv odgovarajuće tablice „`tabcdefabcdefabcd`“. Ovaj se naziv može i skratiti, kao što će se vidjeti u potpoglavlju 3.5.

Članovi tablice mogu biti skoro koji bilo osnovni tip podataka iz jezika SQL. Tipovi `NCLOB` ili `NATIONAL CHARACTER LARGE OBJECT`, `NCHAR(n)` ili `NATIONAL CHARACTER(n)` i `NVARCHAR(n)` ili `NATIONAL CHARACTER VARYING(n)` namjerno nisu poduprti jer su potpuno suvišni od kad je standard Unicode poduprt u tipu `CHARACTER` i tipu `CHARACTER VARYING`, slično kao što u programskom jeziku C tip `char8_t` potpuno zamjenjuje tip `wchar_t`. Što se tiče složenih tipova podataka kao što su tip `ARRAY` (hrv. *niz*) ili tip `SET` (hrv. *skup*), procijenjeno je da oni za IoT nisu potrebni, iako ne postoji pravilo koje ih zabranjuje. Primjeri podatkovnih literala bit će pokazani poslije.

Svaka tablica bilo implicitno bilo eksplicitno, ovisno o poslanim podatcima, vremenski potpisuje podatke (engl. *timestamping*), što znači da ima i stupac „`t`“, a to je ovdje kratica od „`timestamp`“ (hrv. *vremenski biljeg*), opet birana tako da bude što kraća. Njezin je podrazumijevani tip `TIMESTAMP(4) WITHOUT TIME ZONE`, ali može se i promijeniti ili automatski proširiti (ovdje je preciznost 10^{-4} sekunde i ne pamti se vremenska zona, a najprecizniji bi bio tip `TIMESTAMP(6) WITH TIME ZONE`).

Očekuje se da će se vrlo često dohvaćati samo zadnji redak ili nekoliko zadnjih redaka [78]. Isto se tako očekuje i da će vrijeme biti jedinstveno za svaki primljeni redak. Konačno, očekuje se i da će se redci po nečem morati identificirati, pa se u ostvarivanju modela stupcu „`t`“ mora pridijeliti nešto slično primarnomu ključu jezika SQL – „`, PRIMARY KEY(t)`“ – tako da, između ostaloga, upit tipa „`SELECT * FROM tabcdefabcdefabcd ORDER BY t DESC FETCH FIRST 1 ROW ONLY;`“ bude vrlo brzo izvršiv.

Prigodom pohrane podataka smije se dogoditi da se podatak pohranjuje u preciznijem tipu, tako da se zabranjuje zloraba u smislu da se računa na aritmetički preljev ili slično. Npr. ako se računa da je podatak tipa `INT(EGER)` i pošalje se upit `SELECT` za taj broj uvećan za neki veliki broj želeći dobiti negativni broj, to nije dobra uporba modela podataka. Naime, iz podataka koji se šalju u jeziku SQL ne može se zaključiti tip sam po sebi, osim izrijekom dodjelom tipa, što je suvišno u IoT-u zbog vrlo duge standardne sintakse. Tako

da će vrlo često tip biti „širi“ nego se očekuje. Recimo, literal „`64`“ može biti i tip `SMALLINT` i tip `INTEGER` i tip `BIGINT` i tip `NUMERIC(`<cijeli broj veći ili jednak 2>`,`<cijeli broj veći ili jednak 0>`)`, i jedino bi se, opet, za specificiranje mogla upotrijebiti po standardu jezika SQL prilično nezgodna dodjela tipa, npr. „`CAST(64 AS INTEGER)`“. Tek tad bi se moglo zaključiti da je korisnik naumio da to bude tip `INTEGER`, a ne nešto drugo.

Zbog svega toga određuju se podrazumijevani „širi“ tipovi za sve tipove:

- brojčani literali – tip `NUMERIC(`<broj znamenaka najduljega poslanoga broja>`,` <broj decimalnih znamenaka toga broja>`)`;
- tekstovni literali – tip `CLOB` odnosno npr. u bazi PostgreSQL to je tip `TEXT`;
- binarni literali – tip `BLOB` odnosno npr. u bazi PostgreSQL to je tip `BYTEA`;
- vremenski literali – preciznost i nazočnost vremenske zone lako se može odrediti iz literala.

Dakle, npr., neka neki čvor prima podatke „`6, 'f', X'ab', TIMESTAMP '2022-02-02 02:02:02'`“ koje mu je poslao čvor identifikatora `0xabababababcdcdcdcd`. Tad će on imati tablicu „`tabababababcdcdcdcd`“ sa stupcima, ovdje nevažnih naziva, tipova `NUMERIC(1, 0)`, `CLOB`, `BLOB`, `TIMESTAMP(0) WITHOUT TIME ZONE`, a ako dođe nov podatak u većoj preciznosti u pojedinom stupcu, onda se taj stupac proširi na tu veću preciznost.

Tablica se, osim širenjem tipova, može povećati i dodavanjem stupaca. To ima smisla najviše na samom početku rada s nekim čvorom, npr., ako čvor može slati dvije vrijednosti, vrijednost koja se ne šalje bit će implicitno vrijednost `NULL`.

Nazivi stupaca mogu biti bilo što što se može napisati standardom Unicodeom, kao i u tekstovnim literalima. Nasuprot podatcima zapisa JSON koji se obično upotrebljavaju, zasnivanje na jeziku SQL čini se puno prilagođenijim IoT-u zbog toga što se očekuje često kombiniranje i agregiranje tih podataka, za što su relacijske tablice i upitni jezici izvrstan izbor.

Osim tablica drugih čvorova čvor može imati, kako je već spomenuto, i svoje tablice koje može, ali i ne mora, nazvati po svojem identifikatoru EUI-64. Nazivanje po identifikatoru je poželjno jer se tad podatci lakše dohvaćaju i prava konfiguriraju. Ako ta tablica ima neke konfiguracijske podatke za neki čvor, smislenije je da se zove npr.

„`configuration`“ (hrv. *konfiguracija*) i da ima, primjerice, stupac „`node`“ (hrv. *čvor*) i stupac „`parameter`“ (hrv. *parametar*). Tako se može slati upit kao „`SELECT parameter FROM configuration WHERE node = X'abababababcdcdcdcd';`“.

U jeziku SQL se binarni podatci kodiraju heksadekadskim znamenkama, ali postoje mehanizmi kodiranja upita koji mogu smanjiti takve zapise, što je pored ostalih smanjivanja opisano u potpoglavlju 3.5.

3.3. Poruke

Predloženi metaprotokol ne specificira niži protokol kojim se on prenosi, tako da se može prenositi primjerice protokolom TCP, protokolom UDP, protokolom BLE, protokolom 154, protokolom Signal [79], protokolom MQTT, složajem Thread, serijskom vezom, itd. Njegove se poruke dakle mogu učahuriti u protokol bilo koje razine, ali neki niži protokol mora biti. U metaprotokol su ugrađeni neki mehanizmi koji su svojstveni višim protokolima. To su, npr., detekcija duplikata s podatkovnoga sloja, usmjeravanje s mrežnoga itd., pa ako se prenose nižim protokolom, bit će uključeni – a ako se prenose višim, bit će isključeni.

Zamisao je pružiti što tanji „sloj“ iznad postojećih protokola koji dodaje samo ono što nedostaje trenutačnomu nižemu protokolu ili protokolnomu složaju. Ako je protokol vrlo napredan, nema nikakve nadgradnje osim kodiranja tijela. Zapravo, skoro nikad se ne upotrebljava cijeli sloj metaprotokola, nego se skoro uvijek izbacuju dijelovi koji već postoje. U raščahurivanju (engl. *decapsulation*) metaprotokola treba paziti da se neki dijelovi izvuku iz nižega protokola, pa se može i reći da uporaba metaprotokola ovisi o nižim slojevima, što je još jedna značajka koja se ne pojavljuje u „tradicionalnim“ protokolima.

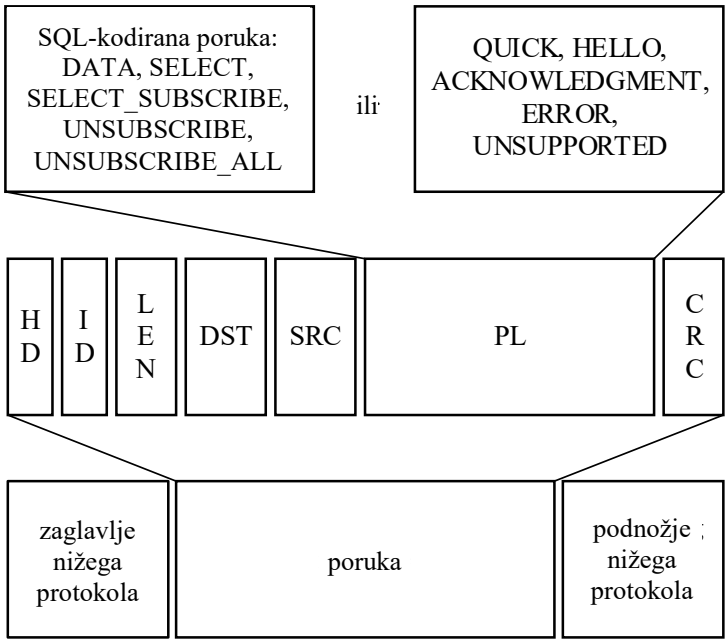
Dodatno treba naglasiti da bi se metaprotokol mogao zvati i međuslojnim protokolom (engl. *cross-layer protocol*) jer obuhvaća više „tradicionalnih“ slojeva. Ipak, tako se ne bi dovoljno naglasile i prethodne značajke. U svakom slučaju, rezimirat će se najvažnije razlike u odnosu na obične protokole:

- može se prenositi protokolom koje god razine;
- obuhvaća više tradicionalnih slojeva;
- njegova uporaba ovisi o nižim slojevima (npr. uključeni dijelovi);

- niži slojevi ovise o metaprotokolu (npr. uključeni mehanizmi);
- pruža najtanji mogući sloj iznad nižih protokola;
- istodobno se može upotrebljavati u binarnom i tekstualnom obliku.

Dijelovi poruke zajedno s prikazom učahurivanja (engl. *encapsulation*) mogu se vidjeti na Sl. 3.3. Što se tiče tih članova, opis im je sljedeći:

- `HD` – **zaglavlje** poruke (engl. *header*), 1 bajt;
- `ID` – identifikacijski broj poruke (engl. *identifier*), 1 bajt;
- `LEN` – duljina tijela poruke (engl. *length*), 2 bajta;
- `DST` – identifikator odredišnoga čvora (engl. *destination*), 8 bajtova;
- `SRC` – identifikator izvorišnoga čvora (engl. *source*), 8 bajtova;
- `PL` – **tijelo** poruke (engl. *payload*), neodređene veličine;
- `CRC` – zaštitni zbroj (engl. *cyclic redudancy check*), 4 bajta.



Slika 3.3. Obličje i učahurivanje poruka metaprotokola

Skoro nikad ne će biti uključeni svi članovi. Npr., ako se identifikator odredišta ili izvorišta može izvući iz nižega protokola, oni se ispuštaju; recimo, ako se identifikator EUI-64 oblikuje iz adrese protokola BLE. Često se ispuštaju svi ili skoro svi članovi, pogotovo ako se želi da poruka bude što kraća. Kraćenju poruka na druge načine posvećena su posebna potpoglavlja. Slijedi opis svih dijelova poruke.

Zaglavlje se poruke sastoji od 8 bitova, a to su kako slijedi:

- `I` – određuje je li `ID` dio poruke;
- `L` – određuje je li `LEN` dio poruke;
- `D` – određuje je li `DST` dio poruke;
- `S` – određuje je li `SRC` dio poruke;
- `R` – određuje je li `CRC` dio poruke;
- `K` – određuje zahtijeva li se potvrda primitka (engl. *acknowledgment*);
- `C` – određuje zahtijeva li se povjerljivost, *Confidentiality*, od `SRC` do `DST;`
- `A` – određuje zahtijeva li se autentičnost, *Authenticity*.

Bitovi zaglavlja određuju koji se članovi poruke uključuju u poruku (prvih pet bitova). Isto tako određuju i koji se mehanizmi dodatno upotrebljavaju u metaprotokolu (zadnja tri bita).

Kombinacija poljâ `ID`, `SRC` i `DST` određuje jedinstvenost poruke u određenom vremenskom razdoblju tijekom kojega se provjerava je li došlo do dvostrukoga ili, općenito, višestrukoga primitka poruke. Polje `ID` je stavljeno da bude 1 bajt tako da se na najmanje 2 bajta poravnaju ostala polja za brži rad, a i zbog toga jer se 256 poruka ne očekuje primiti u vrlo kratkom definiranom vremenu za provjeru duplikata. Bez ovoga se polja ne može u lokalnim mrežama gdje se upotrebljava razašiljanje (engl. *broadcast*), a u „oblacima“ će najvjerojatnije vremenski odsječak za duplikate biti postavljen na 0 – čim se isključuje provjera – ili skoro 0, jer se ondje oni ne očekuju.

Duljina je poruke odabrana da bude 2 bajta zato što bi samo jedan bajt implicirao najveću veličinu od 255 bajtova ($2^8 - 1$), što možda jest dovoljno za lokalne mreže, ali za komunikaciju s korisnicima sigurno nije; za usporedbu se može uzeti najveći paket protokola BLE 5.0 koji toliko prenosi. 65535 bajtova ($2^{16} - 1$) možda jest malo preveliko, ali ako recimo oblaci trebaju uskladiti tablice koje imaju s podatcima nekoga čvora, i nije.

Polje `DST` i polje `SRC` su identifikatori prema standardu EUI-64, tako da moraju biti veliki upravo 64 b odnosno 8 bajtova. Polje `CRC` se konkretno računa algoritmom *provjera kružne zalihosti, 32-bitna, Castagnoli* (engl. CRC-32C) [80], koji je za razliku od algoritma *provjera kružne zalihosti, 32-bitna* (engl. Cyclic Redudancy Check, 32-bit, CRC-32) [81] bolji za kraće poruke [82], a i vrlo je dobro sklopovski poduprt [83]. Treba reći i da se to polje očekuje samo kad ne postoji podatkovni niži protokol, jer svi iole upotrebljivaniji podatkovni protokoli tu mogućnost imaju. To polje se dakle upotrebljava

kad se metaprotokol prenosi protokolom fizičke razine. Fizičkomu se sloju prepuštaju i sinkronizacijski mehanizmi bitova kao što je preambula (engl. *preamble*) ili postambula (engl. *postamble*), jer oni izravno o njem ovise.

Još o polju `ID` treba reći da se taj identifikator upotrebljava za potvrđivanje da je poruka razumljena, javljanju pogrješke u tijelu poruke ili nepoduprte operacije u tijelu poruke. Nije li poruka sama dobro oblikovana, odbacuje se. Za ove poruke treba u tijelu upotrebljavati jednako polje `ID` poruke na koju se odgovara.

Poruke koje se šalju mogu ispustiti dijelove slično kako se to čini u protokolu 6LoWPAN i složaju Thread. Ono što je posebno drugačije jest da se, kao što je rečeno prije, može ispustiti cijelo zaglavlje poruke – takoreći cijeli sloj. U tom se slučaju sve izvlači iz nižega protokola ili se računa. Recimo, polje `LEN` skoro nikad nije potrebno, kao ni polje `CRC`, jer opet više-manje svi protokoli imaju duljinu poruke i ona se može izračunati jednostavnim oduzimanjem. Polje `ID` može biti problematično jer je pitanje kako osigurati jedinstvenost poruke. Iako se je mogao recimo računati i kod *provjera kružne zalihosti, 8-bitna* (engl. Cyclic Redudancy Check, 8-bit, CRC-8), smatra se da je za IoT dovoljno izračunati kod CRC-32C i uzeti prvi bajt da bi se poruka mogla jedinstveno identificirati, iako takvo kidanje koda CRC nije jednako učinkovito kao pravi manji polinom [84]. Tako se pojednostavljuje implementacija, pogotovo jer za kod CRC-8 postoji stvarno previše mogućih polinoma [85].

O opisu članova poruke koji nisu njezino tijelo sad je više-manje sve rečeno i u nastavku će se skoro isključivo razmatrati samo tijelo poruke (polje `PL`), po kojem se razlikuju tipovi poruka.

Metaprotokol je prilagođen uređajima ograničenima sredstavima (engl. *resource-constrained devices*). Posebno se je usredotočilo da on bude „lagan“ (engl. *lightweight*), i to na više načina. Recimo, poruke su zamišljene da budu najkraće moguće, što opcionalnim binarnim kodiranjem, što opcionalnim ispuštanjem dijelova tijela kao što su nazivi stupaca, što već opisanim ispuštanjem članova poruke.

Nadalje, išlo se je u smjeru i da se pojednostavi obradba poruka. Tu treba spomenuti, osim odbacivanja izobličenih poruka, i mogućnost da se skoro uvijek već iz prvoga znaka u tijelu zna o kojem se tipu poruke radi. Isto tako tu je i mogućnost da se jednostavno u nekom ili u mnogim čvorovima neke operacije ne podupiru. Npr. senzor nema razloga išta

znati osim poslati podatke, a ako se recimo nudi izbor da se od njega podatci samo pitaju, nema razloga zašto bi on pohranjivao podatke drugih čvorova.

Najvažniji tipovi poruka u predloženom metaprotokolu su:

- `DATA` (hrv. *podatci*);
- `SELECT` (hrv. *odaberi*), po naredbi `SELECT;`
- `SELECT_SUBSCRIBE` (hrv. *pretplati se na odabir*);
- `UNSUBSCRIBE` (hrv. *otkaži pretplatu*).

Ostali su tipovi:

- `UNSUBSCRIBE_ALL` (hrv. *otkaži pretplatu sa svega*);
- `QUICK` (hrv. *brzo*);
- `PAYLOAD_ERROR` (hrv. *pogrješka u tijelu*);
- `ACKNOWLEDGMENT` (hrv. *potvrda*);
- `OPERATION_UNSUPPORTED` (hrv. *operacija nepoduprta*);
- `HELLO` (hrv. *pozdrav*).

Slijedi opis poruka. Jednostavniji se opis može naći u članku [86], a ovdje se posebno opisuje puni tekstualni, a posebno sažeti binarni oblik. Uloženo je puno razradbe i u jedan i u drugi oblik, i to kako bi prvi bio od koristi prigofom ispravljanja pogrješaka u sustavu, a drugi kad se ta faza riješi.

3.3.1. Poruka DATA

Poruka `DATA` služi za prijenos podataka od izvorišta, čvora `SRC`, do odredišta, čvora `DST`. Ti podatci mogu biti neko odčitavanje senzora, stanje nekoga uređaja, odgovor za zahtjev za podatke, odgovor na pretplatu, itd. U svojem najosnovnijem obliku poruka `DATA` ima jednu imenovanu vrijednost:

<naziv>=<vrijednost>;

Naziv stupca može biti sve što jezik SQL podupire za naziv stupca tablice. To može biti nešto jednostavno, kao npr. u:

```
podatak=56;
```

ili nešto složenije, kao primjer u nastavku. Literali se u ovom sustavu moraju pisati po standardu jezika SQL, pa se tako zbog jedinstvenosti dekodiranja malih slova kao

identifikatora ne prihvaća recimo malo slovo „u“ u „U&'<znakovni niz>'“ (npr. „u&'\ABAB\CDCD'“) ili malo slovo „e“ u „<broj>E<eksponent brojevne baze>“ (npr. „1e10“), ali se svakako prihvaćaju svi valjani načini unosa, pa tako i taj dodatni način upisivanja znakovnih nizova standarda Unicode i taj način upisivanja denormaliziranih brojeva. Složenije se poruke DATA ne obrađuju ručno, tako da alternativni načini ne stvaraju probleme. Odnosno, ne bi se trebale obrađivati ručno nego jednostavno slati u bazu. Nadalje, ako se baš dogodi da čvor nije sposoban obraditi neke mogućnosti literalâ jezika SQL, šalje se poruka OPERATION_UNSUPPORTED.

Osim osnovnoga oblika sa samo jednom vrijednošću poruka DATA može imati i složenije oblike. Prva je proširenost omogućiti slanje više podataka istodobno, odnosno:

<naziv>,<naziv2>,<naziv3>,<…>=<vrijednost>,<vrijednost2>,<vrijednost3>,<…>;

npr.

```
redbroj,ulaz,nadnevak=1,'2',DATE '3333-03-03';
```

ili sa složenijim nazivima stupaca:

```
stup1,"STUPČIĆ",U&"stup\010Di\0107"=1,'2',DATE '3333-03-03';
```

gdje se nazivi stupaca tumače kao stup1, "STUPČIĆ", "stupčić".

Ovdje treba reći da se, po standardu, tip DATE kao i TIME i ostali vremenski literali uvijek piše s ključnom riječju, iako baze podupiru i bez nje, a ta se ključna riječ u predloženom kodiranju može i efektivno ukloniti kodiranjem. Druga je proširenost poruke DATA slanje više skupova podataka u smislu slanja više redaka ili čak cijele tablice. Prošli se način, naime, može razmatrati kao slanje jednoga imenovanoga retka tablice. Prošireni način glasi:

<naz>,<naz2>,<…>=<vrijed>,<vrijed2>,<…>;<vrijed3>,<vrijed4>,<…>;<…>;

npr.

```
stupac1,stupac2=U&'1',TIME '02:02:02';U&'3',TIME '04:04:04';
```

„Redci“ se tablice odvajaju točka-zarezom. Točka-zarez odabran je kombinacijom jednostavnoga dekodiranja i prikladnosti za prikaz, a sa samim upitom INSERT nema veze, kao ni onaj „=“. Treba reći i da se po standardu u tipu TIME i tipu TIMESTAMP ne navodi tip podatka do kraja, odnosno nema zapisa TIME(STAMP) WITH(OUT) TIME

ZONE, osim eksplicitnom dodjelom tipa. U standardu se iz znakovnoga niza zaključuje o čem se radi. Baze mogu imati drugačiji standard, pa se recimo u bazi PostgreSQL mora napisati puni tip bez obzira na to što piše u nizu ako se tip želi ispravno identificirati, pa ako se upotrebljava takva baza, prigodom upita INSERT se literal mora dopuniti.

Podrazumijevana radnja na poruku DATA jest da se stvori tablica s nazivom „t<polje SRC>“ ako već ne postoji, postavi kao javna, odrede tipovi podataka za stupce, poruka pretvori u naredbu INSERT, i konačno se ta naredba pošalje u bazu. To sve vrijedi ako podatke treba pohranjivati. Npr., ako je ta poruka došla kao odgovor na pretplatu, vjerojatno će se umjesto pohrane poželjeti pokrenuti neka konkretna radnja na čvoru-primatelju. Mijenjanje naziva primljene tablice (engl. *aliasing*) se može izvesti kroz sustav pravila.

Poruka tipa DATA se šalje u mnogim različitim situacijama i može se obrađivati na više načina. Njezina je obradba jednostavna u implicitnim operacijama, a za sustav pravila vjerojatno je najbolje prvo ju dodati, tj. upisati podatke u bazu, pa poslije provjeriti kako se je promijenio skup podataka nekim pravilom.

3.3.2. Poruka SELECT

Poruka SELECT služi za zahtijevanje podataka od drugoga čvora, čvora koji je naveden u polju DST. Taj čvor treba vratiti podatke čvoru koji je naveden u polju SRC. Poruka je strukturirana kao naredba SELECT jezika SQL. Taj upit SELECT može biti nešto jednostavno, npr. „SELECT stup1, stup2 FROM tablica;“, ili složeniji. Složeniji upit može uključivati preimenovanja tablica, stupaca, obojega, uz ključnu riječ AS, dodatke kao što je surečenica WHERE (engl. *WHERE clause*), sintagma GROUP BY (grupiranje), sintagma ORDER BY (redanje), surečenica FETCH (ograničavanje broja redaka), surečenica OFFSET (dohvaćanje ne nužno prvih dostupnih redaka), i druge surečenice i uvjete.

Kad se poruka SELECT šalje jednostavnijemu senzoru, očekuje se da će on uzvratiti zadnjom odčitanom vrijednošću jer nema mogućnost pohrane više njih i to je jedini podatak koji ima. Tad se i ne upotrebljavaju dodatci kao što je surečenica WHERE i sl. Oni imaju više smisla za složenije čvorove.

Vrlo se često očekuje dohvat samo zadnjega poslanoga podatka i surečenica FETCH može služiti za to, uz „ORDER BY“ po vremenskom biljegu. Surečenica FETCH je razmjerno

nov dodatak standardu jezika SQL koji služi za ograničavanje broja redaka. Iako je zapravo stara više od desetljeća, neki njezini dijelovi tek sad ulaze u standardu orijentirane baze kao što je baza PostgreSQL. Sintagma `ORDER BY`, naravno, reda retke po nekom kriteriju. Pohrana i obradba podataka zadaća su „viših“ čvorova. U svakom slučaju, čvor sam bira što podupire, a što ne; najsloženiji će podupirati cijeli standard što se tiče naredbe `SELECT`.

Rezultat naredbe `SELECT` mora biti neki podatak koji se šalje porukom `DATA`, makar i prazna tablica. On se, osim ako pravilima nije definirano nešto drugo, vraća čvoru koji je i poslao zahtjev. Odgovor može biti prazan, u smislu da nema vrijednostî. U tom slučaju poruka glasi npr. „<stupac>=“, ali ne samo „=“, jer bi se kosilo s porukom `QUICK`, koja će biti opisana poslije.

Pakiranje podataka, dobivenih pozivima naredaba u bazi, u poruku `DATA` može biti problematično, jer svaka baza ima svoju ispisnu sintaksu. Za brojeve to nije problem, ali jest za druge literale. Recimo, binarni podatci tipa `BINARY/BINARY VARYING/BLOB` se npr. u bazi PostgreSQL unose kao „`'\xabcd'`“ (malim ili velikim slovima), ispisuju kao „`\xabcd`“, a po standardu postoji samo „`X'ABCD'`“ (u standardu velikim slovima), i zato se ovisno o bazi u implementaciji treba pripaziti kako se čitaju literali i prepisuju u poruku, koja mora biti uvijek jednaka i u skladu s ovim opisom neovisno o implementaciji. Procjenjuje se da je to mala cijena za ujedinjavanje različitih svrha kojima poruka `DATA` služi.

3.3.3. Poruka SELECT_SUBSCRIBE

Poruka `SELECT_SUBSCRIBE` služi za ostvarivanje modela „objavi-pretplati“, zajedno s porukom `UNSUBSCRIBE`. Zamisao je da se čvor-pošiljatelj u drugom čvoru pretplati na promjene u rezultatu upita `SELECT`, a taj drugi čvor tomu svojemu „pretplatniku“ šalje podatke, porukom `DATA`, kad do te promjene dođe.

Poruka `SELECT_SUBSCRIBE` je proširena poruka `SELECT` uz dodatak „`SUBSCRIBE` <broj>“ na kraju, npr. „`SELECT stup FROM tabl SUBSCRIBE 1;`“. Ona identificira, zajedno sa poljem `SRC`, pretplatu u čvoru-primatelju. Očekuje se da čvor koji pamti pretplate ima bazu, jer se njom ovakvo nešto razmjerno lako ostvaruje.

Uz pomoć baze se okidači pretplate mogu ostvariti na sljedeći način: pohrani se upit u bazi kao virtualna tablica `VIEW` jezika SQL i kroza nju se gleda jesu li se rezultati promijenili.

Naime, prigodom primitka poruke `SELECT_SUBSCRIBE` izvede se upit `SELECT` i pohrani u novu tablicu, a kad se nešto promijeni u tablici `VIEW` u odnosu na tu tablicu, tablica će se ažurirati i pokreće se vanjska funkcija koja će rezultate poslati pretplatniku.

3.3.4. Poruka UNSUBSCRIBE

Poruka `UNSUBSCRIBE` služi za otkazivanje pretplate. Ona se sastoji samo od smišljene riječi „`UNSUBSCRIBE`“, razmaka i broja, npr. „`UNSUBSCRIBE 1;`“, čim se uklanja podatak o pretplati na podatke identificiranoj poljem `SRC` i brojem `1` u čvoru-primatelju. To može povlačiti i uklanjanje okidača ili internih funkcija baze za rad s pretplatama, kao u prototipu, ovisno o implementaciji.

3.3.5. Poruka UNSUBSCRIBE_ALL

Poruka `UNSUBSCRIBE_ALL` je zapravo proširka poruke `UNSUBSCRIBE` po tom što se umjesto broja navodi ključna riječ `ALL` posuđena iz jezika SQL. To uklanja sve pretplate čvora `SRC` u čvoru `DST`, bez potrebe posebnoga navođenja brojeva, a posebno može poslužiti kad se čvor ide ugasiti, tako da mu se poslije primitka te poruke više ništa ne šalje.

3.3.6. Poruka QUICK

Najjednostavniji je tip poruke poruka `QUICK`, koja je zapravo posebno kodirana poruka `DATA`. Takva poruka ima samo 1 bajt i nosi posebno značenje, a to je konkretno značenje poruke `DATA` oblika „`d=`<bajt dekodiran kao dekadski cijeli pozitivni broj>`;`“. Npr., poruka `QUICK` u obliku „`\x20`“ jednakovrijedna je poruki `DATA` u obliku „`d=32;`“ jer $20_{16} = 32_{10}$.

Slovo „`d`“ je odabrano kao naziv stupca za rad s najjednostavnijim čvorovima. Naime, u njima će se tražiti da nazivi stupaca budu što kraći, a najkraći je jedno slovo. Slovo „`d`“ odabrano je zbog toga što je to skraćeno od „*default*“ (hrv. *podrazumijevano*), slično kako je i „`t`“ kratica za „*timestamp*“, ali i „*table*“ ako poslije nje slijedi identifikator EUI. Za razliku od „`t`“ stupac se „`d`“ automatski ne stvara u tablici.

Očekuje se da će spominjani najjednostavniji čvorovi željeti najjednostavnije moguće javiti neku brojevnu vrijednost, a „`d`“ će biti i zgodan za ispuštanje u svrhu skraćivanja poruke.

Prigodom primanja poruke `QUICK` on se pretvara u poruku `DATA`, a prigodom slanja poruke `DATA` ona se pretvara u poruku `QUICK` ako treba. Konkretno, ako poruka `DATA` ima tijelo „`d=`<vrijednost koja može stati u 1 bajt>`;`“, pretvori se u „`\x`<heksadekadski zapis bajta>“. Treba još reći i da nije zamišljeno da broj bude predznačen (engl. *signed*), odnosno da bude u rasponu od `-128` do `127`, jer se i tako najvjerojatnije pretvara u neku drugu vrijednost. Naime, već ako se ima najjednostavniji temperaturni senzor koji radi u „normalnom“ rasponu od -10 do 30 Celzija, vjerojatno će se željeti biti precizniji od jednoga stupnja i kodirati vrijednosti makar u rasponu od `0` do `80`. To bi bilo `0 = -10`, `1 = -9.5`, …, `79 = 39.5`, `80 = 40`. Na primjeru današnjih temperaturnih senzora proizvođača *teksaški instrumenti* (engl. Texas Instruments, TI) [87] ta se preciznost lijepo vidi.

Ono što je zgodno u tim najjednostavnijim čvorovima jest da će se najvjerojatnije željeti ispustiti i sva ostala polja poruke i dobiti samo 1 bajt ukupne veličine poruke, što je, naravno, vrlo učinkovita uporaba energije za komunikaciju, a i jedan od pokazatelja kako ovo uistinu nije protokol sličan drugima.

Poželi li tko zatražiti podatke čvora koji je poslao poruku `QUICK`, zgodno mu je i da je naziv stupca vrlo kratak, a da se može i ispustiti. To omogućuje samomu čvoru da upotrebljava nadređeni pametniji čvor za pohranu podataka i npr. dohvati prosječno odčitavanje na jednostavan način „`SELECT AVG(stupac);`“.

Osim u porukama `QUICK` ispuštanje svih drugih polja vrlo je zgodno i u drugim porukama, kao što je poruka `HELLO`.

3.3.7. Poruka HELLO

Poruka `HELLO` glasi jednostavno „`HELLO;`“. Riječ je suvišna, ali u neskraćenom obliku želi se dobiti zapravo potpuno tekstualni oblik metaprotokola, u svim porukama, pa tako i u ovoj. Ona služi za javljanje dostupnosti čvora i prešutno javlja dostupnost na toj adresi nižega protokola kojim se šalje. Npr., neka se uzme da poruka `HELLO` prispije protokolom BLE porukom koja ima svoju adresu. Ta adresa je npr. *oglašivačeva adresa* (engl. Advertiser Address, AdvA) postavljena na 6-bajtnu vrijednost `0xabcdefabcdef`. U tom se slučaju pohranjuje na neko vrijeme ruta do toga čvora od čvora-primatelja protokolom BLE preko adrese `0xabcdefabcdef` protokola BLE.

Poruka HELLO ima smisla samo ako nema nikakvih drugih korisnih poruka, a osim javljanja dostupnosti može služiti i za provjeru dostupnosti (engl. *ping*) nekoga čvora ako je bit K postavljen u zaglavlju. Ne provjerava li se dostupnost, ovakva poruka služi najčešće za javljanje vrlo jednostavnoga čvora da se je sad „probudio“ i možda želi primiti poruke koje su čekale dokle je spavao. To je posebno zgodno za čvorove koji žele primati poruke, ali ne žele biti stalno uključeni nego rade u ciklusima. U drugim čvorovima koji žele komunicirati s ovakvima tad se *modulom nižega protokola* (engl. underlying protocol module, UPM) ili pravilima može definirati da se njima šalju poruke samo kad pošalju nešto jer se „zna“ da će poslije slanja oni se prebaciti na primanje na anteni i poslušati imaju li kakvu poruku. Poruka može poslužiti i za održavanje veze (engl. *heartbeat*) s drugim čvorom.

3.3.8. Poruka ACKNOWLEDGMENT

Poruka ACKNOWLEDGMENT služi za javljanje da je poruka primljena i razumljena. Ako je, recimo, polje CRC pogrješno, ili polje LEN, ili se je došlo do kraja poruke prije nego su pročitana sva polja prije tijela, poruka se odbacuje. Ako je poruka poslana sa zahtjevima sigurnosti po bitu C i bitu A u zaglavlju koji nisu poštovani do čvora koji je dobio poruku, gdje se ne misli na poruku ACKNOWLEDGMENT nego onu koju treba potvrditi, poruka se odbacuje. Pravilom se može namjestiti da se umjesto odbacivanja pošalje neka posebna poruka tijekom uklanjanja pogrješaka prigodom izgradnje sustava, ali je to onda odgovornost programera. Automatskih negativnih potvrda (engl. *negative acknowledgment*, NACK) da poruka nije dobra nema u predloženom metaprotokolu.

Tijelo poruke ACKNOWLEDGMENT glasi „ACKNOWLEDGMENT 0x<polje ID poruke dekodirano kao cijeli pozitivni heksadekadski broj>;“. Npr., ako se potvrđuje poruka kojoj je polje ID bilo vrijednost 0x12, onda ona glasi „ACKNOWLEDGMENT 0x12;“. Ovdje se upotrebljava heksadekadski ispis jer se polje ID isto ispisuje heksadeksadski u ispisu izvorne poruke prigodom tumačenja korisniku. Polje ID poruke ACKNOWLEDGMENT smije isto biti vrijednost 0x12, ali i ne mora, jer ona nisu povezana.

Poruka ACKNOWLEDGMENT šalje se kad je bit K bio podignut u zaglavlju primljene poruke, ili kad je slanje definirano nekim dodatnim pravilom. Potvrđivanje može biti i višestruko ako se i u tipu ACKNOWLEDGMENT isto postavi zastavica K. Na taj način drugi čvor zna da je prvi čvor doznao da je primljena i razumljena poruka, pa se tako može

izbjegnuti ponavljanje cijele komunikacije, odnosno poruke i odgovora, ako je samo odgovor izgubljen.

Napredniji načini potvrđivanja s ponovnim odašiljanjem (engl. *retransmission*) nisu predviđeni u modelu, ali se mogu ručno namjestiti u pravilima. To ručno namještanje ne utječe na metaprotokol, jer provjera duplikata postoji. Nadalje, za čvorove spojene na Internet ionako će se najvjerojatnije upotrebljavati protokol TCP(+TLS) ili protokol HTTP(S), koji sigurno na prijenosnom sloju imaju ponovna odašiljanja u slučaju problema. U svakom slučaju, potvrđivanje poruka, zajedno s drugim sigurnosnim mehanizmima, sprječava sigurnosne napade prekidanjem komunikacije.

3.3.9. Poruka PAYLOAD_ERROR

Poruka `PAYLOAD_ERROR` šalje se kad postoji pogrješka u tijelu poruke. Na taj se način izbjegava obradba potpuno izobličenih poruka i to jednostavnim njihovim ispuštanjem, a opet se može javiti korisniku ako je nešto slučajno pogriješio u tijelu poruke. Uostalom, to se „nejavljanje“ upotrebljava i u protokolu IP, a i u drugim protokolima.

Kao jedna od pogrješaka može se pojaviti situacija da ne postoji tablica koju korisnik traži. Recimo, to je zato jer čvor čiji su podatci trebali biti u njoj nije još poslao podatke. Pri tom sustav ne može nikako znati da korisnik nije slučajno pogriješio, pa se i u tom slučaju vraća poruka `PAYLOAD_ERROR`. Još jedan primjer pogrješke bio bi da se pošalje pogrješno kodiran upit `SELECT`. Recimo, umjesto ključne riječi `FROM` kodira se neka druga ključna riječ i tad će se vratiti poruka `PAYLOAD_ERROR` u obliku pogrješke u kojem ga vrati baza. U svakom slučaju, uvijek kad tijelo poruke nije dobro oblikovano vraća se poruka `PAYLOAD_ERROR`. Odnosno, pogrješka se dojavljuje svaki put kad čvor ne razumije tijelo poruke koja mu je došla. Razumije li ga čvor, ali ne može izvršiti operaciju, šalje poruku `OPERATION_UNSUPPORTED` iz sljedećega potpoglavlja.

Poruka `PAYLOAD_ERROR` može biti kratka „`PAYLOAD_ERROR 0x`<polje ID>`;`“ ili može sadržavati dodatak kao „`PAYLOAD_ERROR 0x`<polje ID> `"`<opis oblika UTF-8>`";`“, npr. „`PAYLOAD_ERROR 0x12 "injekcija jezika SQL otkrivena";`“.

Poruka pogrješke može biti u obliku koji vraća baza, ili ručno oblikovanom obliku. Smisao je samo da se korisniku javi, za vrijeme otklanjanja pogrješaka (engl. *debugging*), što je

pogriješio, i to rječito (engl. *verbose*), bez potrebe da on istodobno prati i što se događa u čvoru-primatelju. Izvorna se poruka svakako odbacuje u tom čvoru koji ju je primio.

3.3.10. Poruka OPERATION_UNSUPPORTED

Poruka `OPERATION_UNSUPPORTED` šalje se, slično kao i poruka `PAYLOAD_ERROR`, kad dođe do problema u tijelu poruke, ali za razliku od poruke `PAYLOAD_ERROR` ne indicira pogrješno oblikovano tijelo, nego da čvor koji je primio poruku ne podupire sadržanu operaciju. Npr., ako se šalje poruka `SELECT` čvoru koji ne podupire nikakvu pohranu pa nema što ni odgovoriti, ili ne podupire traženje podataka od drugih čvorova jer nema bazu, ili recimo njegova baza ne podupire neki dio vrlo složena upita…

Poruku `OPERATION_UNSUPPORTED` najčešće će slati vrlo jednostavni čvorovi tijekom ispravljanja pogrješaka kad se od njih zatraži presložena operacija, ali može se dogoditi i, rijetko, da i najsloženiji čvorovi tako nešto pošalju.

Ovaj je tip sličan tipu `PAYLOAD_ERROR` uz zamjenu ključne riječi s „`OPERATION_UNSUPPORTED`“, npr. „`OPERATION_UNSUPPORTED 0x12 "surečenica PERCENT nepoduprta";`“.

Poruke se mogu skoro uvijek odrediti već iz prvoga njihova znaka, jer, npr., poruka `SELECT` počinje ili ključnom riječju `SELECT`, što je slovo „`S`“, ili ključnom riječju `TABLE`, što je slovo „`T`“, ili oblom zagradom ako postoje skupovni operatori, poruka `HELLO` počinje slovom „`H`“, poruka `ACKNOWLEDGMENT` slovom „`A`“ itd. U slučaju malih slova ili dvostrukih navodnika to je očito naziv stupca i poruka je tip `DATA`. U slučaju brojeva ili ključnih riječi vremenskih literala to je očito literal jezika SQL, odnosno poruka `DATA` bez zaglavlja itd.

Što se tiče tih alternativnih početaka poruke `SELECT`, ne radi se, dakle, o posebnim vrstama poruka. Naime, upit `SELECT` po standardu može biti u obliku npr. „`TABLE tablica;`“ umjesto „`SELECT * FROM tablica;`“, s tom razlikom da upit `TABLE` ne podupire neke surečenice u svrhu bržega izvršavanja. Isto tako upit `SELECT` može početi predupitom (engl. *prequery*) kao što počinje „`WITH (SELECT a FROM b) AS c SELECT stupac FROM c WHERE stupac = 1;`“. Upit `SELECT` može sadržavati i skupovne operatore. Tad bi glasio npr. „`SELECT a FROM b UNION`

`SELECT c FROM d;`“, i u takvim upitima katkad između upita odvojenih operatorima kao što su `UNION`, `INTERSECT` i `EXCEPT` trebaju ići i oble zagrade, možda i na početku.

3.4. Model čvora

Čvor u predloženom modelu sustava može biti po volji jednostavno računalo, od čvora koji periodično razašilje 1 bajt svojim podatkovnim protokolom, pa do zamjene za oblak s više, možda i tisuća, korisnika s vlastitim pravilima i vlastitim pridruženim daljinskim stvarima IoT-a. Korisnici mogu imati više kategorija.

Najjednostavniji čvorovi mogu imati ponašanje opisano jednostavno retkom „`prikupi podatak od ugrađenoga fizičkoga senzora i pošalji ga nekomu;`“. S druge strane, spomenuta „zamjena za oblak“ možda obavlja puno složenije stvari. Recimo: prikuplja podatke od više nižih čvorova, obrađuje ih svakakvim pravilima, kombinira ih tako da se dobiju novi podatci na osnovi njih, upravlja složenim operacijama… Složena operacija može biti zahtijevati svježe podatke; ili raspodijeliti se na više poslužitelja; itd.

Ti najsloženiji čvorovi zahtijevali bi dosta programiranja, bilo kao posebno ostvarivanje metaprotokola, bilo kao izgradnju vrlo složenih pravila za koje korisnik treba znati i jezik SQL, ili kao kombinaciju programâ i pravilâ. I najjednostavniji čvorovi isto imaju taj izbor, ali je u složenijima to izraženije, jer trebaju, logički gledajući, cijeli skup različitih programa i usluga; ovi najjednostavniji će se najvjerojatnije ostvarivati običnim programom, koji će ostvariti samo dio predloženoga metaprotokola.

Čvorovi srednje složenosti mogu se najčešće modelirati nekolicinom operacija za rad s podatcima. To mogu biti „individualne“ operacije, koje se događaju periodično ili na neki interni događaj – ili „reakcijske“, koje se događaju kad dođe neka poruka. Zapravo ne postoji neka oštra granica između njih, jer se poruke mogu i injektirati kad dođe neki događaj u svrhu složenije obradbe. U svakom slučaju, operacije mogu biti i vrlo različitih složenosti, ne samo različitih okidača.

Na čvorovima srednje složenosti dat će se nekoliko jednostavnih primjera operacija. Poslije se mogu vidjeti i primjeri konkretne pohrane takvih pravila.

Primjerice, neka postoji čvor koji upravlja nekolicinom aktuatora, ili prikuplja podatke od nekolicine senzora. Takav se čvor može modelirati operacijom „`svakih X`

`vremenskih jedinica izvedi OPERACIJA;`“ ili „`na događaj Y izvedi OPERACIJA;`“. To „`OPERACIJA`“ može biti odčitavanje senzora, slanje naredbe aktuatoru, pohranjivanje primljene vrijednosti nekamo, slanje podataka nekomu daljinskomu čvoru…

Takve se operacije mogu razlomiti na jednostavnije radnje, opisane nekolicinom parametara. Svaka radnja može biti jedno pravilo u sustavu pravila, a jedno pravilo može i okidati drugo, injektiranjem poruke ili aktiviranjem pravila, po redu. Taj sustav pravila korisnik će svakako definirati kako njemu odgovara.

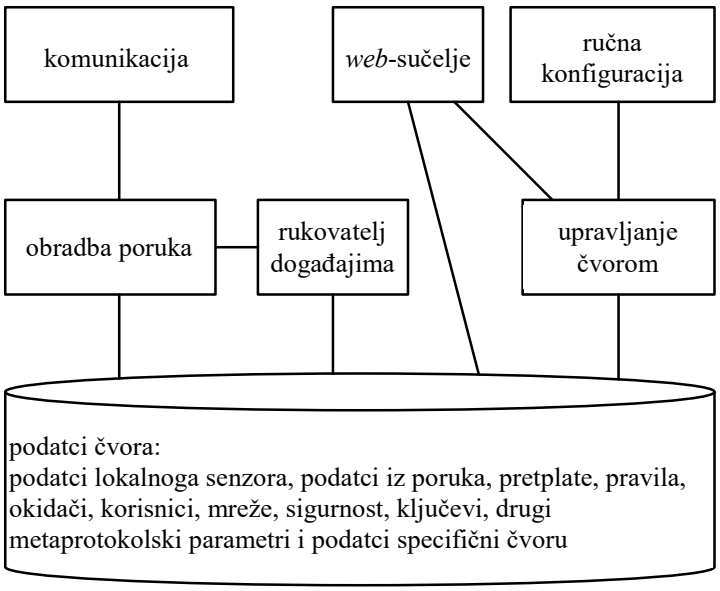
Postoji li čvor koji mora odgovarati na neke poruke, to ponašanje se isto može definirati sustavom pravila, okidanih na njihov dolazak. Tad se poruka može npr. pohraniti, proslijediti, upotrijebiti za neku internu radnju kao što je primjerice čitanje senzora ili slanje naredbe. Isto tako može postojati i neka posebna radnja prigodom slanja poruke, recimo pohrana njezine preslike, slanje na čvor za pričuvne preslike (engl. *backup*), osvježavanje nečega i sl.

Sve se operacije s porukama mogu ostvariti „okidačima“, ili, možda bolje reći, „filtrima“, koji se aktiviraju prigodom svakoga slanja i primanja. „Uhvati“ li filtar poruku, nad njom se izvodi pravilo, a pravilo može biti i složena naredba. Operacije koje se izvode na lokalni događaj (ručno ugrađene) definiraju se bez poruke, a filtre nemaju osim da bi uhvatile taj događaj. Konačno, periodične se operacije jednostavno izvode svaki put kad prođe zadani broj sekunda, jer se veća preciznost od sekunde ne čini smislenom.

Sustav pravila može poslužiti i za najjednostavnije i za najsloženije čvorove, ali će možda u najjednostavnijima pravila biti programirana u programskom jeziku radi izbjegavanja instaliranja složene programske potpore, a u najsloženijima će možda samo služiti za prosljeđivanje posebnomu sustavu za naprednije obradbe podataka.

Da bi se što više smanjila potreba ručnoga konfiguriranja, u metaprotokol su ugrađene neke podrazumijevane radnje. Te radnje ovise o tom što je ostvareno u čvoru. Recimo, podrazumijevano se svi podatci koji dođu pohranjuju u čvoru, ali ako se ne ostvaruje uopće mogućnost primanja podataka ne će biti ni podrazumijevane obradbe tih podataka. Ako je čvor složen, onda će imati vlastita pravila što pohraniti, što proslijediti, što odbaciti i sl. Način definiranja pravila isto je pojednostavljen. Tako npr. filtar može biti „`LEN > 10`“, bez ikakve potrebe ikakva programiranja.

Model čvora opisan ovdje može se zornije vidjeti na Sl. 3.4. Apstraktno govoreći, on rukuje podatcima i konfiguracijom čvora preko više mogućih sučelja. Konkretizacija modela obavit će se kroz sljedeća poglavlja.



Slika 3.4. Model čvora

3.5. Smanjivanje veličine poruka

Može se uočiti da su neke poruke u opisanom metaprotokolu prilično duge i s toga neprikladne za IoT, gdje se teži smanjivanju komunikacije u svrhu štednje energije na uređajima pogonjenima baterijama. Zato se je pristupilo smanjivanju veličine poruka na više načina, a u srži kodiranjem poruka na način da dekodiranje bude što je lakše moguće, a kodirana poruka što kraća.

Razumijevanje poruka u kodiranom obliku nije nešto što je predviđeno da bi korisniku bilo od koristi. Poruke se umjesto toga mogu razmjerno lako dekodirati u prostornoj i vremenskoj složenosti O(N), gdje je N duljina poruke. Više-manje se ide od početka do kraja i u trenu nalaženja koda – on se zamijeni dekodiranim tekstom i razmacima. Tek su dekodirane poruke namijenjene za korisnikovo pregledavanje, ako mu to uopće bude i potrebno.

Kodiranje poruka najveći utjecaj ima u porukama `SELECT`, gdje se upit `SELECT` kodira na način da se svaka ključna riječ zamijeni jednim znakom a slobodno ispuste svi razmaci iz upita. Prije toga će se objasniti kraćenje jednostavnijih vrsta.

Svaka se poruka osim poruka QUICK može dodatno skratiti tako da se, zapravo na sličan način kao i u osmišljavanju baš poruke QUICK, upotrebljava binarno kodiranje i izbace suvišni znakovi. Ni kodiranje poruke SELECT ne ide u drugačijem smjeru, i ovdje se može reći da su to dva osnovna načela kodiranja primijenjena za kraćenje poruka – binarno kodiranje i izbacivanje suvišnoga.

Važno je reći i da se ovo smanjivanje poruka može upotrebljavati, ali i ne mora. Zamisao je da se krene s korisniku razumljivim porukama, a kad sustav proradi, poruke se idu nadograditi kad je uklanjanje pogrješaka gotovo, koje bi bilo puno teže da se je odmah pristupilo kraćenju. Poruke se mogu dakle razmjerno jednostavno dekodirati ako su kodirane, a ako nisu, lako se može ustanoviti da nisu. Naime, čim poruka u sebi ima znak u rasponu \x80-\xFF izvan naziva u dvostrukim navodnicima i znakovnih nizova, odmah se vidi da je kodirana, obično odmah na početku. Slijedi opis kodiranja za pojedine vrste poruka.

Za poruku DATA može se uočiti nekoliko stvari koje se mogu izbaciti da se ne navode svaki put. Recimo, kao i za poruku QUICK, postoje podrazumijevani stupci. Kao što ondje postoji podrazumijevani stupac „d“, ovdje, ako se ne navedu stupci a više ih je, podrazumijevaju se stupci „d1“, „d2“, „d3“… Kao i u „d“, biraju se što kraći, idealno jednoznamenkasti, nazivi proširaka ovih stupaca. Navede li se samo jedan stupac, recimo „stupac“, podrazumijeva se da se stupci zovu „stupac1“, „stupac2“, „stupac3“ itd. Dakle tri načina kraćenja postoje samo tu.

Tako se može izostaviti dio „zaglavlja“ tablice ili cijelo, odnosno sve lijevo od „=“ s njim. Osim toga, neki literali prilično su dugi, pa recimo vremenski literal tipa TIMESTAMP u standardu jezika SQL ima dosta viška znakova i u mnogim se bazama to može kraće napisati. Ovdje će se primijeniti kodiranje ključnih riječi (po Prilogu A) i tako npr. „TIMESTAMP '<vremenski biljeg>'“ može postati „\xEF'<vremenski biljeg>'“, jer je ovdje i razmak suvišan za dekodiranje ostatka. Isto vrijedi i za tip TIME i za tip DATE i za tip INTERVAL, dokle se u brojčanim i znakovnim literalima nema što kratiti.

Podrazumijevani stupac „t“ isto se najčešće izbacuje iz poruke i jednostavno se upotrebljava implicitno vremensko potpisivanje, tj. upotrebljava se trenutačno vrijeme u primanju poruke u odredišnom čvoru. U dugačkim binarnim literalima postoji mogućnost

kodiranja na način da se između „X“ i „'“ napiše koliko je velik literal u bajtima te se on napiše u binarnom obliku. Tim se literal može smanjiti i za gotovo 50 %.

Treba naglasiti da se cijelo vrijeme inzistira na pisanju literalâ i upitâ po standardu jezika SQL. Naime, svaka baza podataka ima svoju sintaksu i ne može se ni na jednu oslanjati, nego samo na standard. I u standardu je specificirano da se ključne riječi moraju pisati velikim slovima, bez obzira na to što se veličina slova u upitima zanemaruje u svim bazama, osim u znakovnim nizovima i nazivima u dvostrukim navodnicima. Nazivi stupaca i nazivi tablica, pisani na običan način, prevode se u velika slova po standardu, a recimo u bazi PostgreSQL u mala slova, ali se u svakom slučaju ignorira njihova veličina u sustavima DBMS.

U predloženom se sustavu mora držati toga da se ključne riječi pišu velikim slovima a nazivi odnosno identifikatori malim slovima, što će omogućiti jednostavno „instantno“ dekodiranje. Npr., ako poruka `DATA` počinje malim slovom, radi se o nazivu stupca, i ne može se raditi ni o kojoj drugoj vrsti poruke. Zapravo postoje samo dva slučaja gdje se ne može iz prvoga znaka odmah zaključiti vrsta poruke. To su prvo slovo „`T`“ gdje može biti poruka `DATA` bez zaglavlja s literalom tipa `TIME` ili `TIMESTAMP`, ili poruka `SELECT` napisana u obliku `TABLE`. Kao drugo, slovo „`U`“ može biti poruka `DATA` s onim zapisom standarda Unicode na početku, ili poruka `UNSUBSCRIBE`, poruka `UNSUBSCRIBE_ALL`. Tad se treba pogledati još jedan znak naprijed, odnosno je li „`I`“ ili „`A`“ u prvom ili „`&`“ ili „`N`“ u drugom slučaju – a ako su poruke binarno kodirane, toga problema nema.

Govoreći o standardu jezika SQL, treba voditi računa i o tom da se, kad se provjerava radi li se o ispravno napisanoj vrsti poruke, drži do podjele velika-mala slova, pa recimo ako se šalje „`select` <nešto>`;`“ umjesto „`SELECT` <nešto>`;`“ da se poruka odbije jer se tako potiče korisnik da drži do potrebne ispravne sintakse.

S druge strane, kad se obavljaju sigurnosne provjere nečega što se ide slati u bazu, treba se zanemariti veliko i malo slovo jer će baza u većini ostvarivanja ignorirati veliko i malo slovo, kako je i rečeno. Pa tako, npr., za injekciju jezika SQL treba se provjeriti da se nije injektirala recimo naredba `DELETE`, `delete`, `DeLeTe`… u upit `SELECT`. Po standardu i da se tako injektira naredba ne bi radila, ali mnoge baze podupiru i takve upite gdje bi to radilo. Npr., baza PostgreSQL podupire nešto tipa „`SELECT (DELETE FROM tablica RETURNING 1);`“. Tako da, i što se tiče takvih naredaba, isto vrijedi – kad

se provjerava je li poruka ispravna, odbaciti sve što nije u standardu, a u sigurnosnim provjerama provjeravati i stvari koje u standardu nisu ali jesu u upotrebljavanoj bazi.

Rečeno je već da se poruka `QUICK` ne može još skratiti, a da se poruka `SELECT` obrađuje u sljedećem potpoglavlju. Ondje će se još obraditi i poruka `SELECT_SUBSCRIBE`, a sad slijedi opis kraćenja sporednih vrsta poruka.

Za poruku `UNSUBSCRIBE` dovoljno ju je samo identificirati jednim znakom i slijedi identifikator pretplate u tekstovnom obliku. Tako da ona zapravo od „`UNSUBSCRIBE 1;`“ postaje „`\xF71`“, dakle 2 bajta, i više nije ni potrebno. Za poruku `UNSUBSCRIBE_ALL` kodira se i ključna riječ `ALL`, a zašto ključna riječ `UNSUBSCRIBE` postaje znak `0xF7` a ključna riječ `ALL` znak `0x85` bit će objašnjeno odmah u sljedećem potpoglavlju.

Za poruku `ACKNOWLEDGMENT` isto je dovoljna samo identifikacija i polje `ID` poruke koja se potvrđuje. Tu je dovoljno treće slovo, „`K`“, i jedan bajt za polje `ID`, pa npr. „`ACKNOWLEDGMENT 0x16;`“ postaje „`K\x16`“. Kao ni prije ni poslije, ni s čim se drugim po standardu jezika SQL opisivane poruke ne mogu zamijeniti.

Na sličan način „`PAYLOAD_ERROR 0x16 "injekcija jezika SQL otkrivena";`“ postaje „`R\x16injekcija jezika SQL otkrivena`“ a „`OPERATION_UNSUPPORTED 0x16 "surečenica PERCENT nepoduprta";`“ postaje „`N\x16surečenica PERCENT nepoduprta`“. Ovdje se jednostavno uzima važno slovo.

Konačno, poruka `HELLO` može, osim ako se ne upotrebljava spojno usmjereni protokol, biti potpuno prazna, 0 bajtova, jer ne nosi nikakav koristan podatak. Tu dolazi do problema gdje se 1-bajtna poruka može tumačiti kao poruka `QUICK` bez zaglavlja ili poruka `HELLO` sa 1-bajtnim zaglavljem, pa se dodatno definira da poruka `HELLO` ne može biti velika ukupno 1 bajt nego su takve 1-bajtne poruke rezervirane isključivo za poruke `QUICK`. Stoga se poruka „`\x16`“ ne dekodira kao `HD=0x16, PL="HELLO;"`, nego kao `HD=0x00, PL="d=22;"` a poruka `HELLO` ili nema ništa ili ima zaglavlje te još neko polje, npr. „`\x80\x00`“ za polje `ID` vrijednosti `0x00`.

Kako razlikovati binarno kodiranu ključnu riječ od dijela upita, kako odabrati koja se riječ kako kodira, koje to implikacije ima na druge programske jezike itd. objašnjeno je u sljedećem potpoglavlju.

3.6. Kodiranje jezika SQL

Ključne riječi iz jezika SQL zamjenjuju se kodom – jednim bajtom u rasponu `0x80-0xFF`, čim se dosta smanjuje veličina poruke. Primjerice, upit „`SELECT a, b, c FROM tablica WHERE a > b AND b > c ORDER BY c;`“ kodira se kao „`X1a,b,cX2tablicaX3a>bX4b>cX5X6c`“ gdje se znakom „`Xi`“ obilježava heksadekadski broj kojim se kodira ključna riječ, npr., umjesto `X2` konkretno piše se „`\xAC`“. Ovdje se upotrebljava i izbacivanje suvišnih graničnika.

Na ovom mjestu naglasila bi se jedna sitnica o zapisu. Naime, svi znakovni nizovi pisali su se ovako: „`niz`“, a u njima, kao što se vidi, mogu biti i binarno kodirani znakovi, zapisani heksadekadski. Iako se u programskom jeziku C/C++, koji služi kao izvorno nadahnuće za takav zapis, svaki „`\x`“ može upotrijebiti za više znakova, u ovom će se radu uvijek upotrebljavati samo za jedan. Npr., ono što bi u jeziku C/C++ bilo `"\xABCD"` ovdje će se pisati „`\xAB\xCD`“, čim se izbjegavaju i dvoznačnosti. Npr., u C/C++ „`\xabej`“ pisalo bi se `"\xab""ej"` zato što *parser* „jede“ sve što može biti heksadekadska znamenka i onda kad dođe do znaka `j` „više ne sluša“.

Heksadekadski će se brojevi u ovom radu radi jednolikosti uvijek kao brojevi pisati velikim slovima, iako ni baze ni programski jezici ne razlikuju tu velika i mala slova. standard jezika SQL svojom formalnom gramatikom razlikuje, što je još jedan razlog. Za heksadekadske se brojeve inače cijelo vrijeme upotrebljava standardni zapis `0x`, a za binarne zapis `0b`. Oktalnih brojeva, kao ni čistih znakova – *char* – u ovom radu ne će biti.

Najvažniji je dio kodiranja jezika SQL kodiranje ključnih riječi kao što su `SELECT` ili `FROM`, pa će ono biti najprije obrađeno. Proučavanjem upita `SELECT` došlo se je do skupa višeznakovnih operatora, ključnih riječi i skupnih funkcija koji se može očekivati. Pod skupne se ovdje misli na agregatne, npr. funkciju *prosjek* (engl. Average, `AVG`) i funkciju *brojenje* (engl. `COUNT`). Pri tom su zanemareni: proširci za objektnu usmjerenost i za „prozorske“ (engl. *window*) i „vremenske“ (engl. *temporal*) mogućnosti, uzorci redaka (engl. *row pattern*), višeznakovne neskupne funkcije i proširci za zapis JSON. Naime, sve to često nije poduprto u komercijalnim bazama, ili se razlikuje od jedne do druge, a to se događa zato što su to razmjerno novi dodatci standardu jezika SQL.

Ukupno je u jeziku SQL, odnosno skici standarda SQL:2016 po kojoj se je radilo, 11 višeznakovnih simbola, 219 predbilježenih ključnih riječi (engl. *reserved keyword*) i 250

nepredbilježenih ključnih riječi (engl. *unreserved keyword*). Recimo, ugradbene funkcije kao npr. `CAST` (hrv. *pretvori tip*) ili `EXTRACT` (hrv. *izvuci dio*) spadaju u ovu drugu skupinu, a rečeno je da se od njih razmatraju samo skupne. Od svega toga je izabrano 128 stvari za kodiranje, čim se je popunio cijeli raspon kodiranja. Standard koji se je gledao jest dakle standard SQL:2016, čija se besplatna radna inačica može, odnosno se je mogla, naći se na Internetu na mjestu [88], ili kupiti po cijeni od 180 CHF ako se inzistira na pravoj inačici. Čini se da to potonje nije potrebno za dalji rad, pogotovo jer ni standard SQL:2016 ne donosi znatne novosti [89] a kamo li standard SQL:2019. U svakom slučaju, u upitima se očekuje standard SQL:2016, što dakle ne ograničava korisnika skoro ništa.

upit jezika SQL po standardu je zapravo u obliku *američki standardni kod za razmjenu informacija* (engl. American Standard Code for Information Interchange, ASCII), osim dijelova gdje se može upotrebljavati standard Unicode. Tako zapravo raspon oblika ASCII od broja `128` do broja `255` na većini upita ostaje neupotrijebljen, i to je bilo nadahnuće za uporabu toga raspona za kodiranje. Ispis tih znakova u ovom je radu u heksadekadskom obliku, iako bi se mogao očekivati i zapis *standard 8859-1 Međunarodne organizacije za standardizaciju* (engl. International Organization for Standardization, standard 8859-1, ISO 8859-1) [90]. Slično se može primijeniti i na druge jezike kao što je C ili C++, kojima je isto sve osim identifikatorâ i znakovnih nizova u obliku ASCII. Popis zamjena ključnih riječi jezika SQL binarnim znakovima može se naći u Prilogu A. Zapisivanje binarnih znakova u upitu ne priječi zapisivanje standarda Unicode na svojim mjestima, a standard Unicode se svakako zapisuje samo u obliku *Unicodeovo tekstno obličje, 8-bitno* (engl. Unicode Text Format, 8-bit, UTF-8) [91].

Treba se reći i da se je izvorno pokušalo naći smislenije kodiranje riječî, npr. „`S`“ za „`SELECT`“ ili recimo „`\x`<128 + char za slovo S; heksadekadski>“ za nj, ali tu se je pojavilo više problema. Recimo, naći smisleno kodiranje za svaku skraćenu riječ je nemoguće a i pretraživanje po abecedi neporedanoga popisa zamijenjenih riječi tijekom programiranja ostvarivanja vrlo je teško.

Osim kodiranja ključnih riječi, kao i za poruku `DATA`, suvišno se znakovlje ispušta. U upitu jezika SQL najčešće su suvišni svi razmaci. Naime, ako se pišu ključne riječi velikim slovom, a nazivi izvan dvostrukih navodnika malim, ostaje vrlo malen broj situacija gdje se dvije riječi jednake veličine slova nalaze jedna do druge. U ključnim riječima to se može češće dogoditi: može ih biti i više jedna za drugom, recimo „`GROUP BY DISTINCT`“ ili

„FULL OUTER JOIN“ kao u tablici u Prilogu A, ali se to već rješava kodiranjem. Kad se kodiraju sve riječi koje se trebaju kodirati, ostaje možda samo nekoliko situacija u ugradbenim funkcijama gdje će se to dogoditi, ako i uopće. To je uopće moguće zato što postoje ključne riječi unutar ugradbenih funkcija, koje se ne kodiraju, jer ih ima previše (ugradbenih funkcija). Naime, neke su ključne riječi unutar ugradbenih funkcija i standardne riječi koje se kodiraju. Npr. u funkciji EXTRACT koja služi za izvlačenje dijelova vremenskoga literala može se pojaviti riječ FROM, riječ SECOND, riječ MINUTE, riječ DATE i tako dalje.

Što se tiče situacija gdje mogu doći dva naziva jedan do drugoga – koliko se može utvrditi po standardu – to se može dogoditi samo u preimenovanju tablica i stupaca, odnosno npr. u „SELECT stupac novo_ime_za_stupac FROM tablica novo_ime_za_tablicu INNER JOIN tablica2 novo_ime_za_tablicu2(novo_ime_za_njezin_stupac);“, što se lako može izbjegnuti, bilo ostavljanjem razmaka, bilo bolje dodavanjem ključne riječi „AS“, pri čem je i jedno i drugo samo jedan bajt.

Dodatno se mogu kodirati i nazivi tablica. Svaka tablica ima prilično dugačak naziv „t“ + 16 znakova od broja 0 do broja 9 i slova a do slova f, pa se tih 16 znakova mogu zapisati binarno kao 8 bajtova poslije toga „t“. Kako prepoznati radi li se o 16 običnih ili 8 kodiranih? Dobro pitanje, jer se ne može znati osim ovako: ako je cijeli upit kodiran, onda je očito i to kodirano. Već se po prvom znaku upita vidi je li kodiranje uključeno, a što se tiče prepoznavanja gdje se nalazi naziv tablice, po standardu se naziv nalazi samo iza „FROM“ i „JOIN“. To vrijedi samo ako se zanemari da se može upotrebljavati u upitu za „kvalificiranje“ naziva kao npr. tablica.stupac i ako se zanemari da se može napisati „FROM tablica1, tablica2“ (ovdje bi u slučaju ispuštanja dijela FROM bilo i drugih problema). To nisu problemi jer će uz kvalificiranje svaki smisleni programer upotrebljavati i preimenovanje tablice radi kraćenja cijeloga upita, a ono potonje stvaranje Kartezijeva umnoška je zastarjela sintaksa koja se više ne preporučuje, nego pisanje „tablica1 CROSS JOIN tablica2“, što je kodirano 2 bajta, odnosno samo jedan ako se izostavi šumna riječ (engl. *noise word*) JOIN (na popis riječi iza kojih dolaze nazivi dodaje se ovdje ključna riječ CROSS). Na kraju, predupiti WITH ne mogu se imenovati kako god. Ako počinju slovom t i oni se mogu pomiješati sa skraćenim nazivima tablica.

Ako je surečenica FROM, npr. „FROM tablica“, izostavljena u potpunosti a recimo ključna riječ JOIN i dalje postoji, podrazumijeva se da piše „FROM t<polje DST>“, tako da se u traženju podataka od nekoga čvora može izbaciti još najmanje 10 bajtova iz upita, 1 za razmak i 9 za naziv.

Piše li samo „FROM“, ali ne i tablica poslije te riječi, onda se podrazumijeva da piše „FROM t<polje SRC>“. Nije odabrano obrnuto, gore polje SRC a ovdje polje DST, jer se češće očekuje traženje tuđih nego svojih podataka od drugoga čvora, a to se u poglavlju 4 vidi i u primjerima. Na ovaj se način poruka SELECT kojom se traže svi podatci od nekoga čvora može skratiti na samo 2 bajta, odnosno „<binarni kod za ključnu riječ SELECT>*“. U upitu se razmjerno lako može ustvrditi ima li ili nema surečenice FROM, bez stvaranja sintaksnih stabala. Samo se pogleda je li se došlo do jedne od surečenica koja može doći poslije surečenice FROM. To su surečenica JOIN, WHERE, GROUP, HAVING, ORDER, ili se je samo došlo do kraja cijeloga upita. Ustanovi li se tako da ga nema, ubaci se „FROM t<polje DST>“. Ako je prazna surečenica FROM, vidi se da je to još lakše uočiti: samo se pogleda što joj slijedi.

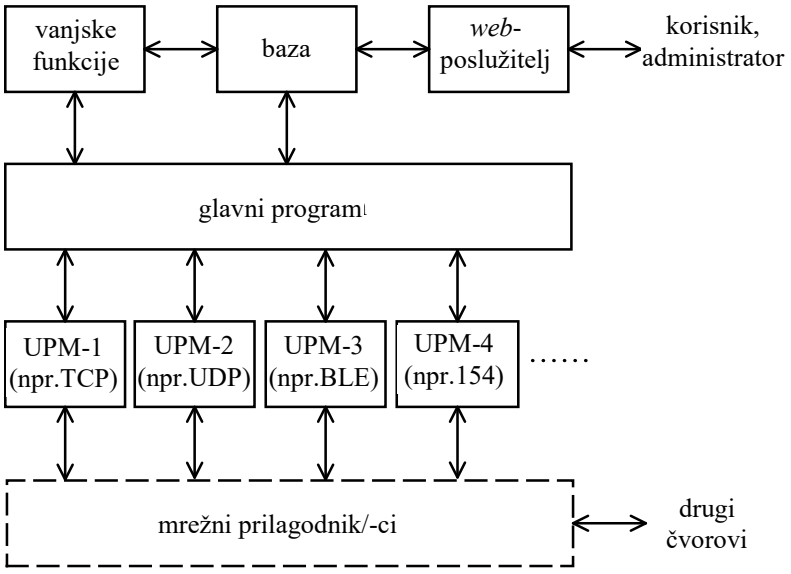
Znak je točka-zarez u upitu potpuno suvišan, jer se po standardu samo jedan upit može izvoditi istodobno. Još jedna stvar koja će se izbaciti jesu komentari, jednoredčani ili višeredčani, koji mogu postojati u neskraćenoj inačici upita. Korisnost im je vrlo upitna jer će se prije upotrebljavati komentari u jeziku u kojem se programiraju poruke nego oni, ali kako ne utječu ni na što ne zabranjuju se.

Primjeri kodiranja navedeni u scenarijima poglavlja 4 pokazat će da se upit bez poteškoća smanjuje na polovicu duljine. Uzevši u obzir da se ima vrlo moćno oruđe za slanje upita, kraće upite od toga nije realno očekivati za jednake mogućnosti u nekim drugim rješenjima, a i teško se može naći jednako moćan jezik u IoT-u.

U nastavku ovoga poglavlja (modelu čvora, sustavu pravila, …) za opis se upotrebljava izgrađeni prototip. Naravno da bi se za uporabu navedenoga metaprotokola mogle izraditi i drukčija ostvarivanja, interno izgrađena drukčije. Međutim, na prototipu su pokazane sve željene mogućnosti predložene arhitekture koja se sastoji i od potpore metaprotokola za razmjenu poruka i od mogućnosti čvorova kroz sustave pravila i slične operacije (postavke, dinamičke promjene načina rada ili uloga i sl.).

3.7. Programska potpora u čvoru

Model potpore može se vidjeti na Sl. 3.5. Na njoj se može vidjeti sljedeće: osnovne su sastavnice modela glavni program i moduli nižih protokola, ovdje obilježeni kraticom UPM.



Slika 3.5. Model potpore

Ovaj je model konkretizacija modela čvora, gdje se npr. svi podatci koji su prije opisani pohranjuju u bazu, rukovanje događajima se izvodi kroz glavni program, sučelje za *web* se izvodi preko poslužitelja itd.

Modula UPM može biti koliko je potrebno, i ostvareno u čvoru, a mogu ostvarivati protokol po želji. Ovdje su, kao i u prototipu, za sad ostvarena ova četiri modula, a to su prvo modul protokola TCP koji upotrebljava protokol *sigurnost prijenosnoga sloja* (engl. Transport Layer Security, TLS) [92] za sigurnost ako ju poruka treba i modul protokola UDP koji upotrebljava protokol *datagramska sigurnost prijenosnoga sloja* (engl. Datagram Transport Layer Security, DTLS) [93] za sigurnost ako ju poruka treba. Druga su dva modul protokola BLE i modul protokola 154, ali za sada bez uporabe sigurnosnih mehanizama. Moduli UPM omogućuju uporabu metaprotokola s tim nižim protokolima, odnosno učahurivanje u njih i raščahurivanje iz njih, kao i izvlačenje informacija iz zaglavljâ i ubacivanje informacija u njih – zaglavlja nižega protokola.

Kako se trebaju odabrati vrata za prijenosne protokole, odlučeno je da će to u prototipu biti 44000 jer je to razmjerno pamtljiv broj, a u konvencionalnom rasponu i po organizaciji *autoritet internetskih dodijeljenih brojeva* (engl. Internet Assigned Numbers Authority,

IANA) slobodan [94] za rezervaciju. Vrata 44001 s istim svojstvima upotrebljavaju se za sigurnu komunikaciju.

Moduli UPM se mogu uključiti i isključiti u konfiguraciji čvora. Recimo, oblak vjerojatno ne će upotrebljavati podatkovne nego prijenosne protokole, ali usmjernik će vjerojatno upotrebljavati oboje; najjedostavniji čvorovi vjerojatno samo jedan podatkovni protokol. Modul UPM može biti na kojem god sloju modela *međupovezivost otvorenih sustava* (engl. Open Systems Interconnection, OSI). Protokol TCP i protokol UDP, kao i protokol BLE i protokol 154, po dva su protokola vrlo različitih slojeva, tj. prijenosnoga odnosno podatkovnoga sloja. Zato su odabrani za prikaz koncepta ujedinjavanja više slojeva metaprotokolom.

Modul UPM teorijski može izravno upotrebljavati mrežnu opremu čvora (npr. mrežnu karticu preko upravljačkih programa) ili upotrebljavati mrežne usluge sustava (npr. postojeći mrežni podsustav), kao u prototipnoj implementaciji.

Prilagodnici i protokoli dolaze u razmatranje u određivanju *proširenoga jedinstvenoga identifikatora* (engl. Extended Unique Identifier, EUI) čvora kad on nije eksplicitno napisan. Na istom računalu može biti više istih vrsta prilagodnika, recimo više njih protokola Ethernet ako se radi o poslužitelju ili više njih protokola BLE ako se radi o usmjerniku i tad može jedan čvor imati više jednakovrijednih identifikatora.

Moduli UPM moraju znati, dodatno, i kako razaslati poruku i kako shvatiti da je bila razaslana, ako to podupiru. Recimo, protokol TCP ne može po naravi stvari podupirati ikakvo razašiljanje a i OS odbija takav pokušaj, protokol UDP može ako OS to podupire, npr. na OS-u Linux. Ostala dva upotrebljavat će ga vrlo često. Razašiljanje ima veze i sa sigurnošću, jer se poruka ne može asimetrično kriptirati javnim ključem primatelja ako primatelj nije zadan. Postoji mogućnost i da se poruka razašalje nižim protokolom, ali poljem `DST` namijeni točno određenomu čvoru. To služi kad se ne zna koja je „fizička“ adresa odredišta, ali se zna njegov identifikator EUI.

U potpuno ostvarenom modelu prema Sl. 3.5, kakav je i onaj u prototipu, glavni program povezuje se s bazom. Baza čuva sustav pravila, korisnike, u više kategorija, pohranjene poruke, pretplatne okidače, interne procedure, podatke o daljinskim čvorovima itd. Baza se može upotrebljavati izravno ili preko vanjskih funkcija (engl. *external functions*).

Vanjske funkcije mogu biti ostvarene u jeziku C, jeziku C++ ili nekom drugom jeziku više razine, a služit će pozivanju iz baze. Npr., za sustav Postgres to je uz pomoć knjižnice

`libpq` u programskom jeziku C, `libq++` u programskom jeziku C++ ili neke slične. Naime, glavni program može preko odgovarajuće knjižnice pozivati što god mu treba u bazi. Baza sama ne može drugačije pozivati funkcije glavnoga programa nego preko vanjskih funkcija, a to je, recimo, kad treba „okinuti“ pretplatu, izvesti radnju s poslužitelja za *web*, obaviti složenije operacije od onih jezika SQL… Jedina alternativa bila bi pozivanje naredaba iz naredbene ljuske iz baze, ali onda bi se moralo to nekako i primiti u glavnom programu. To se ne čini kao bolje rješenje od ovoga.

Sučelje za korisnike i administratore bilo bi najbolje ostvariti poslužiteljem za *web*, kao i na Sl. 3.5. To vrijedi i za samo jednoga korisnika i jednoga administratora, i za samo jednoga glavnoga administratora od ukupnoga broja korisnika, i za ostale konfiguracije. Uz pomoć takva sučelja korisnici mogu dohvaćati podatke, slati naredbe čvorovima, i sl. U prototipu konkretno može se pristupiti pohranjenim porukama, mogu se injektirati poruke odnosno stvoriti poruke kao da su došle od negdje, što je vrlo korisno za ulančavanje pravilâ, ili se mogu slati ručno nekomu čvoru, ili upravljati pravilima i konfiguracijom čvora.

U jednostavnijim se čvorovima mogu izostaviti neki elementi modela čvora (tj. njegove implementacije) ili upotrebljavati samo neki elementi s vlastitim programom. Primjerice, jednostavniji čvor ne će upotrebljavati sučelje za *web* jer bi to previše trošilo njegovu bateriju ili procesor. U takvim slučajevima konfiguracija će se na početku pohraniti u bazu i najvjerojatnije se više ne će mijenjati. Teorijski je moguće da se konfiguracija sustava pravila dinamički mijenja ovisno o primljenim porukama ako korisnik zna kako su pravila pohranjena, ali taj pristup ovdje ne će biti potanje istražen. Ako je čvor još jednostavniji i nema bazu, onda se sve „uprogramira“ u glavni program, s tim da se ne će poduprijeti sve moguće operacije; nego, recimo, samo podupirati fiksan skup upita koji se mogu dobiti, a za sve drugo vraćati poruku `OPERATION_UNSUPPORTED`.

Iako izostavljanje elemenata u modelu čvora nije implementirano, on jest izgrađen modularno, i u nekoj nadogradnji implementacije moglo bi se definirati što uključiti a što ne, odnosno, pripraviti određene dijelove za uključivanje u druge programe.

U izgradnji pojedinoga čvora pojedini moduli UPM se mogu u sustavu uključivati i isključivati. Jednostavni čvorovi ne moraju imati bazu, pa ni poslužitelj, a čak i glavni program može biti zamijenjen nekim što ne ostvaruje cijeli metaprotokol, ali taj program svejedno može upotrebljavati module UPM. Moduli UPM ne ovise o glavnom programu i

zamišljeno je da se prevode kao posebne knjižnice. Na taj način postignula bi se maksimalna ponovna upotrebljivost (engl. *reusability*) sustava. Za komunikaciju se između modulâ UPM i glavnoga programa mogu upotrebljavati redovi poruka, kao u prototipnoj implementaciji.

3.8. Sustav pravila

Sustav se pravila u prototipu nalazi pohranjen u tablici „`rules`“. U konkretnom ostvarivanju modela nije nužno imati ga stvarno u bazi, ali je vjerojatno tako najjednostavnije, pogotovo jer su neki njegovi dijelovi upiti jezika SQL ili izrazi jezika SQL pa baza treba i za njih.

Svako se pravilo sastoji od sljedećih dijelova, prikazanih na Sl. 3.6:

- vlasnik;
- identifikator;
- vrsta i filtar kad se aktivira;
- radnja s porukom i promjena poruke ako je uključena;
- dodatna radnja s upitom ili naredbom;
- dodatna radnja sa slanjem ili injektiranjem nove poruke stvorene priloženim upitom ili naredbom;
- identifikator za aktiviranje;
- identifikator za deaktiviranje;
- zastavica je li ovo pravilo aktivno;
- vrijeme zadnjega pokretanja (za periodična pravila).

Neki od tih dijelova imaju i dodatne elemente. Identifikator je broj tipa `INTEGER` koji je ujedno i prioritet pravila i, zajedno s vlasnikom, ključ tablice. Pravila se ispituju po redoslijedu od najmanjega do najvećega broja pravila za korisnika čiji je čvor poslao poruku u slučaju primanja, odnosno za korisnika čijemu se čvoru šalje poruka u slučaju slanja.

You are authorized to view (edit) rules for all users.
Viewing table "rules".
Table ordered by username ascending and id ascending.

For username	this is rule number	It is activated	(filter)	It instructs to	(modification)	to execute this	(query/command 1)	and to form a new msg from this	(query/command 2)
		◉ on sending when: ○ on receiving when: ○ every this amount of seconds:		◉ drop message ○ modify message with this: ○ do nothing		◉ SQL query: ○ bash command: ○ (execute nothing)		◉ query and send it: ○ command and send it: ○ query and inject it: ○ command and inject it: ○ (form nothing)	
root	0	◉ on sending when: ○ on receiving when: ○ every this amount of seconds:	3600	○ drop message ○ modify message with this: ◉ do nothing		○ SQL query: ○ bash command: ◉ (execute nothing)		○ query and send it: ◉ command and send it: ○ query and inject it: ○ command and inject it: ○ (form nothing)	procitaj_senzor.s

d	using protocol	and address	using insecure port	and secure port	using CCF	and ACF	using broadcast	and override implicit rules	Also activate rule number	Also deactivate rule number	Is active?	Last run on:	(actions)
					☐	☐	☐	☐ .			☐	LOCALTIMESTAMP(0)	INSERT reset TRUNCATE
	tcp	00000000c0a8			☑	☑	☐	☐ .			☑	2022-02-02 11:11:11 +01	UPDATE reset DELETE

If "SELECT <filter>;" evaluates to TRUE, the filter is triggered. You can use column names HD, ID, LEN, DST, SRC, PL, CRC, proto, addr, CCF, ACF, encrypted (read-only), signed (read-only), broadcast, override, insecure and secure. Appropriate FROM is automatically appended.
Modification is performed like "UPDATE message SET <semicolon-separated command 1>; UPDATE message SET <semicolon-separated command 2>; <...>;".
During SQL queries the current message is stored in table "message" and columns HD, ID, LEN, DST, SRC, PL, CRC, proto, addr, CCF, ACF,encrypted (read-only), signed (read-only), broadcast, override, insecure and secure.
bash commands are NOT executed as /root/, but as the user who started the database.
Filter can be either a number or a string.
Leaving a field empty indicates NULL value.
Deactivating a rule deletes its timer. Changing a period does not.
Id must be unique for user. Smaller value indicates bigger priority.
When broadcasting a message any "addr" is ignored.
On send and receive rules "last_run" is meaningless.
Strings are written without excess quotations, e.g., proto = 'tcp'.

Slika 3.6. Primjer implementacije sustava pravila

Done

Vrsta definira kada treba provjeriti uporabu pravila dok filtar specificira potankosti provjere. Vrsta može biti „nad svakom odlaznom porukom“ (na Sl. 3.6 engl. *when receiving*, hrv. *kad prima*), „nad svakom dolaznom porukom“ (na Sl. 3.6 engl. *when sending*, hrv. *kad šalje*) i „periodično“ (na Sl. 3.6 engl. *every this amount of seconds*, hrv. *svakih ovoliko sekunda*). Filtar može biti filtar nad porukom ili samo broj sekunda za periodična pravila. Filtar se piše pojednostavljeno, tako da se piše uvjet, koji može biti i izraz jezika SQL ako treba, umjesto upita. Npr. „`DST = X'abababababcdcdcdcd'`“, ili „`proto = 'tcp'`“, ili zastavica „`ENCRYPTED`“, ili kombinacija logičkim operatorima, a pri čem se efektivno pokreće „`SELECT` <filtar> `FROM poruka;`“ i gleda rezultat. Izračuna li se filtar u vrijednost `TRUE` onda se pravilo izvodi (po ostatku definicije pravila).

Sigurnosni se problem injektiranja jezika SQL kroz filtar ne može pojaviti jer se nedvosmisleno izvodi samo naredba `SELECT`. Podupire li baza naredbu `INSERT` ili slične naredbe za mijenjanje podataka kao podnaredbe, onda to treba provjeriti u programu koji izvodi pravila. Mogući su problem i pristup podatcima kojima konkretan korisnik ne smije pristupiti, tako da se svakako treba imati na umu korisnik koji je vlasnik pravila i gledati u bazi koja izvodi naredbe automatski smije li korisnik pristupiti nekoj tablici. Naime, svaka tablica ima pristupna prava.

Radnje s porukom i promjene poruke mogu biti: „promijeni poruku sljedećom promjenom“ (na Sl. 3.6 engl. *modify with*, hrv. *preinači uz*), „odbaci poruku“ (na Sl. 3.6 engl. *drop message*, hrv. *odbaci poruku*), „ne čini ništa navedeno nego nastavi obradbu ove poruke“ (na Sl. 3.6 engl. *do nothing*, hrv. *čini ništa*). Promjena se poruke definira skraćenim upitom jezika SQL kao i filtar, gdje promjena može biti npr. „`DST = X'cdcdcdcdabababab'`“, „`proto = 'udp'`“, „`ENCRYPT`“, ili više njih odvojeno točka-zarezima, npr. „`SRC = X'babadedadecaceca'; broadcast = TRUE; SIGN`“. Efektivno se izvodi „`UPDATE poruka SET` <prva promjena>`; UPDATE poruka SET` <druga promjena>`;…;`“, npr. „`UPDATE poruka SET SRC = X'babadedadecaceca'; UPDATE poruka SET broadcast = TRUE; CALL ENCRYPT();`“. Ni ovdje se ne može dogoditi problem s injektiranjem, ali se može pristupiti podatcima kojima taj korisnik nema pristup, ako se to ne provjerava, najbolje u izvođenju naredaba automatski.

Sljedeće je polje „dodatna radnja s upitom ili naredbom“ gdje se može izvesti neki upit jezika SQL ili naredba ljuske `bash` nakon prethodne radnje i promjene poruke (ako je bila zadana), a prije nastavka obradbe poruke. Radnja se može izvesti i poslije završetka obradbe neke poruke, ali to nije poduprto izravno, nego se može aktivirati neko pravilo s periodom `0`, čija će obradba krenuti poslije obradbe poruke, a koje će samo sebe deaktivirati.

Po tom ide „dodatna radnja sa slanjem ili injektiranjem nove poruke stvorene priloženim upitom ili naredbom“ pri čem se radi o slanju ili injektiranje poruke generirane *ad-hoc*, a koja se može generirati upitom jezika SQL ili naredbom jezika `bash`. Još se mora specificirati i protokol i adresa drugoga čvora, koji mogu biti vrijednost `NULL` da se izvuku iz poruke u obradbi, da bi se poruka mogla poslati. Nadalje, treba odabrati uključuje li se u protokol uporaba povjerljivosti i autentičnosti u protokolu, što se ovdje zove *zastavica povjerljivoga kanala* (engl. confidential channel flag, `CCF`) i *zastavica autentičnoga kanala* (engl. authentic channel flag, `ACF`), i konačno je li uključeno razašiljanje i jesu li isključena implicitna sigurnosna pravila, kao i moguća posebna nesigurna ili sigurna vrata za slušanje (engl. *custom insecure/secure listen port*) prigodom slanja prijenosnim protokolom. Poruka se može proslijediti i nekomu programu, uz pomoć naprednijih upita.

Sljedeća su dva polja identifikatori drugih pravila koja treba aktivirati (na Sl. 3.6 engl. *activate*, hrv. *aktiviraj*) ili deaktivirati (na Sl. 3.6 engl. *deactivate*, hrv. *deaktiviraj*) ili vrijednosti `NULL` kad se to ne čini. Zadnje je polje zastavica je li ovo pravilo aktivirano (na Sl. 3.6 engl. *is active*, hrv. *je aktivno*). Nije li onda se ne provjerava ni izvodi.

Svako pravilo ima sva opisana polja, ali se ne moraju sva upotrebljavati ako imaju praznu vrijednost. Uz pravila potrebno je da u sustavu postoji i popis korisnika i njihovih ovlasti. Ovlasti uključuju i smiju li pregledavati i mijenjati svoja pravila, bez obzira na ulogu.

Pravila se izvode u programu kad se ispune „učinak“ i „filtar“, slijedno po identifikatoru od manjega. Više pravila može zadovoljiti jednak uvjet, ili jednako pravilo imati više ponovljenih definicija s drugačijim uvjetom. Tako se može dobiti i pravilo koje se pokreće i prigodom slanja i prigodom primanja i periodično. Naravno, u periodičnima neka polja nemaju smisla jer ne postoji poruka u obradbi.

Za obradbe je zadan uređen niz pravila od najmanjega identifikatora do najvećega (prioritet je obrnut). Niz se osvježava kad se neko pravilo aktivira/deaktivira. Aktiviranje/ deaktiviranje manjega identifikatora od trenutačnoga u obradbi može služiti kao

aktiviranje/deaktiviranje uz odgodu, tek za sljedeći događaj, dokle se aktiviranje/ deaktiviranje većega identifikatora može smatrati aktiviranjem/deaktiviranjem odmah za tekući događaj. Vraćanje na manji identifikator, naime, nije moguće.

3.9. Upravljanje čvorom

Osim sustava pravila postoje još mnogi parametri koje treba konfigurirati za čvor. Zamišljeno je da se svaki napredniji čvor može konfigurirati grafičkim sučeljem. To može biti sučeljem za *web*, kao što je i ostvareno u predloženom prototipu, izravnim postavljanjem zapisane konfiguracije (u bazi).

Konfigurirati se mogu: korisnici, tablice, pravila, višekategorijska pristupna prava tablicama odnosno čvorovima, niži protokoli, mrežni prilagodnici, javni certifikati, privatni ključevi, podatci o daljinskim čvorovima, i konačno postavke za komunikaciju s drugim čvorovima u nastavku obilježene sintagmom „korisničke metaprotokolske postavke“. Svaki od ovih članova konfiguracije zapisan je kao najmanje jedna posebna tablica u bazi, npr. *users* (hrv. *korisnici*), *rules* (hrv. *pravila*), *table_owner* (hrv. *tablica-vlasnik*), *table_reader* (hrv. *tablica-čitač*).

Korisnik mora biti ili „javni korisnik“ (na slikama engl. *public*, hrv. *javni*) ili „običan korisnik“ ili „običan administrator“ ili „korijenski administrator“ (na slikama engl. *root*, hrv. *korijenski*). Korijenski je administrator samo jedan, ne može se obrisati i ima sve ovlasti. Običan korisnik odnosno običan administrator može konfigurirati članove konfiguracije čvora koje mu običan administrator odnosno korijenski administrator dopusti, obično prigodom stvaranja njegova računa, s tim da obični administrator može konfigurirati te dijelove i za sve obične korisnike, za razliku od običnoga korisnika. Javni je korisnik isto samo jedan, ne može se obrisati i njegove dijelove mogu konfigurirati svi kojima je to dopušteno. U trenutačnom modelu ono što se može dopustiti ili zabraniti jest: pregledavanje i mijenjanje tablica, izravno slanje poruka, izravno injektiranje poruka, pregledavanje i mijenjanje pravila, izravno slanje upita u bazu, mogućnost prijave na sučelje, pregledavanje i mijenjanje podataka o čvorovima, pregledavanje i mijenjanje postavaka, pregledavanje i mijenjanje pristupnih prava za čvorove, ručno izvršavanje periodičnih pravila, pregledavanje i mijenjanje drugih korisnika (za obične administratore – običnih korisnika i javnoga korisnika, za obične korisnike – samo javnoga korisnika), pregledavanje i mijenjanje nižih protokola i prilagodnika, pregledavanje i mijenjanje

certifikata i privatnih ključeva, pregledavanje i mijenjanje samoga sebe (pregledavanje pravâ, mijenjanje imena i zaporke), te pregledavanje i mijenjanje konfiguracije u ime drugih korisnika.

Tablice u kojima su primljeni podatci od drugih čvorova mogu se pregledavati i mijenjati. Pristupna prava njima su imena korisnika koji smiju pristupiti određenoj tablici naredbom `SELECT` za čitanje ili drugima za pisanje. Prigodom upita `SELECT` u bazi se može ustanoviti kojim se tablicama pristupa, a ni za druge naredbe nije ni malo teže. Pristup tablici „t<identifikator EUI čvora>“ za čitanje odnosno pisanje implicitno omogućuje i „pristup“ za čitanje odnosno pisanje i svemu drugomu za taj čvor u čvoru u kojem se konfigurira.

Pod nižim protokolima misli se na uključivanje ili isključivanje modula UPM za protokole koji prenose metaprotokol, npr. protokol TCP, protokol UDP, BLE, 154, HTTP, MQTT i druge. Mrežni se prilagodnici, kao pridružena konfiguracija, isto mogu uključiti ili isključiti po potrebi. Primjerice, ako se radi o čvoru koji glumi oblak, nema razloga zašto bi mu radili bežični prilagodnici, osim možda ako mu se ne može pristupiti preko mobilnih mreža a ima karticu *pretplatnički identifikacijski modul* (engl. Subscriber Identity Module, SIM).

Pod pregledavanjem podataka o daljinskim čvorovima misli se na pregled podataka koje čuva sam program, a to su: trenutačno polje `ID` poruke prema nekomu čvoru, podatci za provjeru duplikata i podatci za čuvanje rute. Recimo, tu se može ručno konfigurirati kako pristupiti nekomu daljinskomu čvoru ili provjeriti kojim protokolom i kojom rutom mu se trenutačno pristupa.

Pod slanjem i injektiranjem poruka misli se na mehanizam koji je moguć kroz pravila. Pod izravnim se slanjem upita u bazu misli na, recimo, ručno čišćenje tablica tijekom ispravljanja pogrješaka u sustavu. Jedna stvar koju ovdje svakako treba osigurati jest da korisnik, ako ima pravo slati upite u bazu, nema pravo promijeniti podatke o sebi i dati si drugačija prava. Pitanje je koliko bi to utjecalo ako bi ionako mogao slati kakve bilo upite u bazu. Zato se provjeravaju i prava nad tablicama, i to kroz sam sustav DBMS.

Pod korisničkim metaprotokolskim postavkama čvora trenutačno su definirani: razdoblje za otkrivanje duplikata, razdoblje koliko vrijede mrežne rute prije odbacivanja, uključenost prosljeđivanja poruka, uključenost učenja rute, uključenost sigurnosnih provjera za slanje, uključenost sigurnosnih provjera za primanje, identifikator EUI podrazumijevanoga

odredišta, identifikator EUI tekućega čvora, kao i moguća posebna nesigurna ili sigurna vrata za slušanje (engl. *custom insecure/secure listen port*) prigodom slanja prijenosnim protokolom. Radi li se o čvoru u oblaku, najvjerojatnije će se željeti isključiti provjera duplikata jer to rješavaju niži čvorovi, ili će se isključiti podrazumijevano odredište jer je to najviši čvor u hijerarhiji. Prikaz mogućnosti može se vidjeti na Sl. 3.7. Na Sl. 3.8 odmah poslije vidi se implementacija pristupnih prava (dodavanje, mijenjanje, brisanje, brisanje svih čitača). Na Sl. 3.9 odmah poslije vidi se implementacija metapodataka čvora (dodavanje, mijenjanje, brisanje podataka za duplikate, identifikaciju, rute).

Račun korijenskoga administratora „`root`“ i javnoga korisnika „`public`“ postoji odmah prigodom inicijalizacije. Dodavanje korisnika, a pogotovo „običnih“ administratora, nije nužno ako se radi o jednostavnijem čvoru. Ako se radi o čvoru koji glumi oblak, očekuje se da će se na nj spajati mnogi korisnici, svaki sa svojim čvorovima odnosno tablicama. Treba se paziti da budu ispravno konfigurirani i ne daju nikakva suvišna prava nad sustavom korisnicima kojima to ne treba.

Upravljanje sustavom gdje ima puno korisnika i čvorova može se prilično zakomplicirati i korijenski administrator će vjerojatno željeti u tom slučaju i ručno mijenjati stvari u bazi. To može izravnim slanjem upita kroz sučelje i ne treba mu nikakvo posebno spajanje na bazu. Npr., ako treba učiniti neki čvor privatnim, da ne briše jednoga po jednoga korisnika pregledavača može samo poslati upit „`DELETE FROM table_reader WHERE tablename = 'tabcdabcdabcdabcd';`“.

3.10. Sigurnosne radnje

Sigurnost se u metaprotokolu sastoji od povjerljivosti i autentičnosti. Povjerljivost se ostvaruje kriptiranjem, tako da se sadržaj poruka skrije od čvorova koji ga prenose, osim ako ga iz nekoga razloga trebaju čitati. Autentičnost se ostvaruje potpisivanjem poruke, ili digitalnim potpisom. Ovi se termini upotrebljavaju na način uobičajen u IoT-u, kao npr. u protokolu 154, gdje je tajnost (engl. *secrecy*) dovoljna za povjerljivost.

You are authorized to view (edit) configuration for all users.
Viewing table "configuration".
Table ordered by username ascending.

Username	Forward messages?	Use LAN switch algorithm?	Duplicate expiration in seconds	Address expiration in seconds	Trust sending?	Trust receiving?	Default gateway	My EUI	Insecure port	Secure port	Actions
public	☑	☑	600	36000	☐	☐			44000	44001	UPDATE reset
root	☑	☑	600	36000	☐	☐			44000	44001	UPDATE reset

Write gateway (empty if unused) and EUI (empty if default) as a binary string, e.g., abababababababab.
Write numbers as an integer, e.g., 11.
Empty field indicates NULL value.

Done

Slika 3.7. Primjer implementacije postavaka čvora

You are authorized to view (edit) permissions for all tables.
Viewing tables "table_reader" and "table_owner", table readers shown first.
Tables ordered by tablename ascending and username ascending.

Tablename	Username	Actions
		INSERT reader reset TRUNCATE readers
t80a589fffe380477	public	UPDATE owner reset

Write tablename and username as a string, e.g., table and root.
Done

Slika 3.8. Primjer implementacije pristupnih prava

You are authorized to view (edit) everything here.

View destination for this SRC (and when was the last message with which ID received for that DST, used for duplicates): cdcdcdcdcdcdcdcd (remove) (add) Write destination as a binary string, e.g., abababababababab.
Destinations ordered by address ascending.

Outgoing ID for this SRC: 22 change reset randomize Write id as an integer, e.g., 11.

View protocol for this SRC (and when was the last message with which src received over that proto, used for routes): udp (remove) (add) Write protocol as a string, e.g., tcp.
Protocols ordered by name ascending.

Done

Slika 3.9. Primjer implementacije metapodataka čvora

Sigurnost je ostvarena na razmjerno minimalistički način: u smislu da postoji, ali da postoje i bolje mogućnosti na koje se ovaj metaprotokol može osloniti, ako je to potrebno. To znači da se nipošto ne želi ovim metaprotokolom konkurirati npr. protokolu TLS 1.3 i protokolu DTLS 1.3, trenutačno najvažnijim sigurnosnim protokolima, ako treba zaštititi komunikaciju protokola TCP ili protokola UDP između dvaju čvorova.

Zapravo se želi reći da, kao što i drugi dijelovi metaprotokola „uskaču“ kad ih niži protokoli ne podupiru, a to je zbog sredstvima vrlo ograničenih čvorova, i sigurnost ovdje može poslužiti u takve svrhe „uskakanja“. Kad se radi o naprednijim i napadima izloženijim čvorovima, nema razloga zašto se ne bi upotrebljavala sigurnost koja je već uključena u niži protokol, makar kao poseban sloj kao što je to protokol TLS iznad protokola TCP, posebno ako se taj protokol već vrlo uspješno upotrebljava.

Događa li se komunikacija u zatvorenu sustavu, gdje se svim čvorovima i poveznicama vjeruje, za sigurnost nema potrebe; ne mora se upotrebljavati ni kriptiranje ni potpisivanje. Neka se zamisli sustav lokalne mreže koja je fizički odvojena od svega drugoga. Odvojena može biti bilo prostorom ako se upotrebljavaju bežični protokoli, bilo lokotima na kabelima ako se upotrebljavaju žični protokoli. U takvoj mreži sigurnost samo smeta – ako se svim čvorovima vjeruje, to jest. Isto vrijedi i ako je protokol siguran sam po sebi, kao u ovom primjeru [95], ili ako se upotrebljavaju sigurnosni mehanizmi podatkovnih protokola.

Kad komunikacija iziđe iz zatvorena sustava i krene k odredištu kakvom nesigurnom mrežom, sigurnost je u nekoj mjeri potrebna odmah. Takva nesigurna mreža može biti sam Internet ili, karikirano, satelit koji komunicira s drugom lokalnom mrežom. Odnosno, treba makar uključiti potpisivanje poruka da netko zlonamjeran ne pokuša injektirati lažne poruke u mrežu. Postoji li mogućnost presretanja poruka, a one nose osjetljive podatke, treba se upotrebljavati i kriptiranje. U nižim protokolima često nije moguće praktično definirati samo uporabu potpisa ili samo kriptiranje. Dapače, u protokolu TLS i protokolu DTLS se to i ne preporučuje [96]. Tad je potrebno uključiti oboje, i kriptiranje i potpisivanje, čim je potrebno i samo jedno od njih.

U implementaciji se upotrebljavaju algoritam *napredni enkripcijski standard, 128-bitan* (engl. Advanced Encryption Standard, 128-bit, AES-128) [97] za simetrično kriptiranje blokova podataka, algoritam *Rivest-Shamir-Adleman, 1024-bitan* (engl. Rivest-Shamir-Adleman, 1024-bit, RSA-1024) – s nadopunjanjem čistoga teksta iz *kriptografskih*

standarda javnih ključeva #1 (engl. Public-Key Cryptography Standards #1, PKCS #1) [98] – za asimetrično kriptiranje vrlo kratkih nizova: simetričnih ključeva za algoritam AES i sažetaka. Algoritam *sigurni sažetkovni algoritam, 256-bitan* (engl. Secure Hash Algorithm, 256-bit, SHA-256) – s nadopunjanjem čistoga teksta iz *zahtjeva za komentare 6234* (engl. Request for Comments 6234, RFC 6234) [99] – upotrebljava se za sažimanje u svrhu potpisivanja, algoritam *šifreno ulančavanje blokova* (engl. Cipher Block Chaining, CBC) za način kriptiranja toka podataka; algoritam *kriptografski standardi javnih ključeva #7* (engl. Public-Key Cryptography Standards #7, PKCS #7) [100] za kriptografsko nadopunjanje blokova. Algoritam AES-128 ne smatra se dovoljno sigurnim za tzv. *top secret* [101], pa u slučaju takve potrebe bolje je upogoniti dovoljno jaku sigurnost u nižem protokolu, ali svi se ovi algoritmi još uvijek smatraju jednima od najsigurnijih algoritama i ne postoje isplativi napadi na same algoritme, nego samo na njihove implementacije, iako nisu kvantno otporni kao [102]; slično [103] vrijedi i za protokol TLS 1.2 i protokol DTLS 1.2 koji se ovdje upotrebljavaju.

Odabrani algoritmi za kriptiranje i sažimanje nisu prilagođeni ograničenim čvorovima. Prvo, jer je zamisao da takvi upotrebljavaju usluge nižih protokola (npr. protokol BLE) za ostvarivanje sigurnosti u lokalnoj mreži, a da, u prelasku k globalnoj, čvor koji je granica po potrebi upotrebljava algoritme metaprotokola. Nadalje, uporaba algoritama povećava duljinu poruke što opet nepovoljno utječe na ograničene čvorove. Čvorovi koji nisu ograničeni najčešće nemaju ni problema s vezom i povećana poruka im ne čini problem.

U budućem se radu na implementaciji modela može dodati mogućnost odabira algoritama, ali tad treba taj odabir nekako zapisati i u porukama koje ga upotrebljavaju, kao što to npr. čini protokol TLS. To bi onda tražilo i odgovarajuće izmjene u obličju poruka, tj. trebalo bi proširiti obličja navedena u nastavku. U te svrhe postoje i drugi znakovi koji mogu „okidati“ te algoritme.

Digitalna se omotnica (engl. *digital envelope*) upotrebljava za ostvarivanje povjerljivosti. Tijelo poruke koja sadržava digitalnu omotnicu prikazano je u Tablici 3.I.

Tablica 3.I. Sadržaj digitalne omotnice

znak „@“	poruka kriptirana simetrično algoritmom AES	simetrični ključ kriptiran asimetrično algoritmom RSA	inicijalizacijski vektor IV

Vektor IV je dugačak 16 bajtova jer se upotrebljava algoritam AES-128. Kriptirani ključ je dugačak 128 bajtova jer se upotrebljava algoritam RSA-1024 sa standardnim nadopunjanjem. Duljina prije kriptiranja je 16 bajtova. Veličinu kriptirane poruke nije potrebno definirati; može se izračunati iz polja `LEN` i veličine svih drugih članova. Sva su polja osim prvoga binarna. Simetrični ili tajni ključ i vektor IV generiraju se nasumično u čvoru koji šalje poruku, po mogućnosti samom knjižnicom za sigurnost poduprtom sklopovskim generatorima slučajnosti. Naravno, po definiciji digitalne omotnice, za asimetrično kriptiranje služi javni ključ njezina primatelja. Simetrično se kriptira cijela izvorna poruka, od zaglavlja do kraja tijela. Digitalna omotnica, u kojoj se nalazi kriptirana izvorna poruka, stavlja se kao tijelo nove poruke s novim poljem `ID` i ostalim članovima nove poruke. Pravilima se može izmijeniti i polje `SRC`, tako da se skrije pošiljatelj do dekripcije, i sl.

Digitalno potpisana (engl. *digitally signed*) poruka, tj. njezino tijelo, prikazana je u Tablici 3.II.

Tablica 3.II. Sadržaj digitalnoga potpisa

znak „#“	sažetak poruke kriptiran asimetrično, to jest digitalni potpis	izvorno tijelo

Potpis je dugačak 128 bajtova zbog uporabe algoritma RSA-1024 sa standardnim nadopunjanjem. Potpisuje se cijela poruka od zaglavlja do kraja izvornoga tijela, odnosno, sve što prethodi potpisu i slijedi za njim. Pošiljatelj kriptira sažetak svojim privatnim ključem, a odredišni ga čvor dekriptira javnim ključem pošiljatelja. Veličina samoga sažetka od 32 B ovdje nema veliku ulogu jer se sažetak ionako kriptira nadopunjen na ukupno 128 B.

Znakovi „`@`“ i „`#`“ po standardu ne upotrebljavaju se u jeziku SQL iako ih baze mogu upotrebljavati za neke svrhe [104]. Isto tako, ti se znakovi ne upotrebljavaju nigdje drugo ni u metaprotokolu, tako da ne može doći do zabune oko toga je li poruka osigurana ili nije. Potpis je ovdje stavljen na početak poruke da se odmah vidi je li poruka potpisana ili nije jer bi se inače morala pretraživati poruka za znakom `#`.

„Digitalnim pečatom“ zove se kombinacija digitalnoga potpisa i digitalne omotnice, koja je samo njihov slijed <potpis><omotnica>, prema Tablici 3.III.

Tablica 3.III. Sadržaj digitalnoga pečata

znak „#“	ostatak potpisa prema tablici 3.II	znak „@“	ostatak omotnice prema tablici 3.I

Potpisuje se drugačije, samo omotnica. Programiranjem ili sustavom pravila može se ostvariti i višestruko omatanje i potpisivanje, ali to rijetko ima smisla. Da je ovdje omotnica na početku, nikako se ne bi moglo ustanoviti gdje završava koji dio u Tablici 3.I jer njezini dijelovi mogu imati binarno kodiran ovaj znak `#`.

U stvaranju se pravila mogu čitati zastavice „`ENCRYPTED`“ (hrv. *kriptirana*) i „`SIGNED`“ (hrv. *potpisana*), tj. obje kad je potrebno upotrebljavati ili provjeriti „digitalni pečat“. Posebnim naredbama „`ENCRYPT`“ (hrv. *kriptiraj*), „`DECRYPT`“ (engl. *dekriptiraj*), „`SIGN`“ (engl. *potpiši*) i „`VERIFY`“ (hrv. *provjeri*) omogućuje se omatanje i „razmatanje“, odnosno potpisivanje i provjera potpisa, u sustavu pravila. Dovoljno je postaviti čvor `SRC` i čvor `DST` u poljima i pozvati te naredbe poljem „modifikacija poruke“ da se to izvede automatski u pravilu; tj. one pripadaju nadgradnji sustava pravila posebnim ugradbenim mogućnostima.

Konfiguracija čvorova koji upotrebljavaju navedene sigurnosne mehanizme nešto je zahtjevnija jer oni trebaju imati privatne i javne ključeve, a javne treba i na neki drugi sigurni način dostaviti drugim čvorovima, možda i posebno oblikovanim porukama.

Kad je to moguće trebaju se upotrebljavati mehanizmi nižih protokola, npr. protokola TCP, protokola BLE, protokola HTTPS. Zato se sigurnost zahtijevana bitovima `C` i `A` u zaglavlju može ostvariti uporabom i tih nižih protokola. Drugi čvorovi koji prenose poruku trebaju poštovati bit `C` i bit `A`, a ako ne mogu, jednostavno odbaciti tu poruku. To vrijedi samo ako nisu postavljena eksplicitna pravila koja to rješavaju na koji drugi način. Npr. može se gledati ide li poruka kroz lokalnu mrežu ili ne, i tako razlučiti „sigurnu“ od „nesigurne“ okoline.

Sigurnost u metaprotokolu može funkcionirati i tako da samo odredišni čvor može dekriptirati poruku, ali se isto preporučuje zaštita nižim protokolom između svakih dvaju čvorova kad je to moguće. Naime, ako je uspostavljena sigurna ruta, recimo stvar-usmjernik-oblak-korisnik, između svakih se dvaju čvorova npr. upotrebljava protokol TLS pa nitko između ne može ništa ni pokušati, ali se upotrebljava i osiguravanje metaprotokolom, pa međučvorovi mogu prenositi poruku, ali ne i čitati ju ili mijenjati.

Treba reći i da se sigurnost može ostvariti i samo na dijelu rute, ako je preostali dio siguran sam po sebi, ili ako iz kojega drugoga razloga pravila tako govore. Pravila mogu pružiti „posredničku“ (engl. *proxy*) sigurnost. Razmotrit će se prvo čemu bi to uopće služilo.

Recimo da se ima poruka koja kreće iz lokalne mreže jedne stvari, k pametnijemu čvoru, primjerice usmjerniku. On prosljeđuje poruku složajem TCP/IP, k sljedećemu čvoru IoT-a. Između njih može biti i običnih usmjernika koji upotrebljavaju složaj TCP/IP za usmjeravanje paketa, i sigurno će ih i biti jer poruka putuje Internetom. Poruka ide dalje preko čvorova IoT-a do odredišta. Odredište može biti neki drugi pametniji čvor ili samo obična stvar u kojoj drugoj lokalnoj mreži.

Sigurnost koja se zahtijeva u zaglavlju mora se poštivati u svakom skoku između dvaju čvorova IoT-a koji upotrebljavaju navedeni metaprotokol. Inače, ako je poruka primljena, a sigurnost definirana u zaglavlju nije poštivana, čvor bi tu poruku trebao odbaciti, osim ako posebnim pravilima nije premošteno drukčije.

Problem na koji treba računati jest da je vrlo vjerojatno da će stvari IoT-a, odnosno najniži čvorovi, imati ograničenosti na sredstva, a pogotovo što se tiče kriptografije. Iako danas moderni procesori imaju sklopovski ostvareno dosta toga [105], u jednostavnim se čvorovima upotrebljavaju jednostavniji procesori, i puno manje memorije. Pokazat će se da kriptografija u nekim slučajima nije ni potrebna, nego će se nadomjestiti „posredničkom sigurnošću“.

Naime, radi se o tom da metaprotokol omogućuje da se sigurnost zahtijeva čak i ako se ne može osigurati u samom čvoru. Ta sigurnost može biti namijenjena izlasku iz trenutačne sigurne mreže, ili, opet, kako se god namjesti pravilima. Raščlanit će se prvo koje su dvije vrste sigurnosnih pravila.

Prvo, postoje implicitna sigurnosna pravila koja su vrlo jednostavna i koja su ugrađena u metaprotokol. Osim implicitnih za čvor mogu biti definirana i dodatna eksplicitna pravila. Slična se metodologija upotrebljava cijelo vrijeme; u metaprotokol su općenito ugrađene jednostavne podrazumijevane stvari, koje se mogu premostiti dodatnom konfiguracijom. Tako se mogu premostiti i implicitna pravila za sigurnost.

Za razmatranje jednostavnih sigurnosnih pravila neka se započne najprije s jednostavnim čvorovima ili „stvarima“. Oni se mogu konfigurirati da potpuno zaobiđu sigurnosne zahtjeve i jednostavno svima „vjeruju“. Takvi čvorovi mogu prosljeđivati sve poruke bez promjena – osim ako nisu namijenjene samo njima – i tako omogućuju da te poruke

nenamijenjene njima dođu do onoga komu jesu namijenjene. To će posebno poslužiti ako usmjernik nije izravno dohvatljiv (engl. *reachable*) nego je predaleko i mora se do njega doći u više skokova. Takvo ponašanje čvora može biti konfigurirano u programskoj potpori ili jednostavno isprogramirano u glavnom programu ako nema mogućnosti konfiguriranja.

Složeniji čvorovi mogu raščlaniti poruku i osigurati ju na neki način ako treba. To može biti nižim protokolom, ili metaprotokolom, ili obama. Kad se tako razmotre i jednostavniji i složeniji čvorovi, kao i postojanje eksplicitnih pravila, dolazi se do sljedećega modela.

Algoritam za podrazumijevanu provjeru prigodom slanja poruka prikazan je na Sl. 3.9. Pseudokod je napisan više-manje kao u programskom jeziku C. Argument je „`poruka`“ poruka koja se prenosi, argument je „`odredište`“ adresa sljedećega čvora IoT-a u lancu po upotrijebljenom nižem protokolu, dokle je taj niži protokol opisan trećim argumentom „`niži`“.

```
provjeri_slanje(poruka, odredište, niži) {
  ako (konfiguracija.vjeruj_za_slanje || poruka.premosti_implicitna
      || !(poruka.HD.C || poruka.HD.A) || poruka.ručno_osigurana()) {
    pošalji poruku na odredište;
  inače ako (niži.može_osigurati(odredište, poruka)) {
    pošalji poruku uz sigurnosne mehanizme protokola niži na odredište;
  inače {
    odbaci poruku;
  }
}
```

Slika 3.9. Implicitne sigurnosne provjere prije slanja poruke

Jednostavan se čvor tako može konfigurirati da sve prosljeđuje uz postavku „`vjeruj_(svima_)za_slanje`“. Pravila se mogu premostiti i zastavicom „`premosti_implicitna(_pravila)`“ koja se može postaviti u slanju ili injektiranju poruke (ako je ta poruka bila stvorena kroz sučelje ili pravilom). Složeniji će čvorovi prvo provjeriti eksplicitna pravila, a onda i bit `C` i bit `A`. Ako su te zastavice poruke postavljene, a sigurnost je osigurana metaprotokolom, onda poruka isto ide naprijed. U protivnom se pokušava zaštititi makar nižim protokolom. Nije li ni to moguće, poruka se odbacuje.

Slična provjera ide prigodom primanja svih poruka. Algoritam je prikazan na Sl. 3.10, gdje argument „izvorište“ čini adresu prošloga čvora IoT-a u lancu ali opet u nižem protokolu koji se tu upotrebljava.

```
provjeri_primanje(poruka, izvorište, niži) {
  ako (konfiguracija.vjeruj_za_primanje || poruka.premosti_implicitna
      || !(poruka.HD.C || poruka.HD.A) || poruka.ručno_osigurana
      || niži.bilo_osigurano(izvorište, poruka)) {
    prihvati poruku na obradbu;
  } inače {
    odbaci poruku;
  }
}
```

Slika 3.10. Implicitne sigurnosne provjere prije primanja poruke

Drugi su dio u ostvarivanju sigurnosti eksplicitna pravila. Ona se najčešće upotrebljavaju kad poruke izlaze iz lokalne mreže ili ulaze u nju izvana. Ta sigurnosna pravila nisu poseban dio sustava pravila, nego se definiraju unutar njega uz pomoć posebnih naredaba `ENCRYPT`, `SIGN`, i sl.

Razmotrit će se moguća uporaba eksplicitnih pravila u sustavu sa sigurnom lokalnom i nesigurnom globalnom mrežom. Primjer pravila u usmjerniku koji se nalazi na granici (povezuje lokalnu mrežu s nekim čvorovima izvan nje) može biti ovakav: ako je primljena poruka iz lokalne mreže za neki čvor izvan nje, ali nije obilježena uporaba sigurnosti u zaglavlju te poruke, ona se jednostavno tiho odbacuje. Slično se pravilo može definirati za poruke koje dolaze u obrnutom smjeru, od čvorova izvan lokalne mreže.

S druge strane, pravilom se može definirati da se poruka u usmjerniku osigura prije slanja izvan lokalne mreže, ako se porukom zahtijeva sigurnost (ili i bez obzira na to). Recimo, ako je bit `C` postavljen, pozove se naredba `ENCRYPT` i to će automatski kriptirati poruku uz pomoć javnoga ključa čvora `DST` ako on postoji u sustavu. Nije li u sustavu takav ključ dodan, pravilo se ne će uspjeti izvršiti i poruka će se odbaciti implicitno.

Komplementarno pravilo može dekriptirati poruku i provjeriti potpis poruke koja ide u suprotnom smjeru, tj. dolazi od čvora izvan lokalne mreže. Tima se dvama pravilima osigurava „posrednička“ sigurnost, gdje usmjernik ostvaruje sigurnost umjesto stvari, a koji je sigurno bolje opravljen za kriptografiju od njih, ali i za uopće pohranu različitih

ključeva. Tad je sigurnost ovisna o sigurnosti toga posrednika – ako napadač uspije prodrijeti u nj onda može kompromitirati sve stvari spojene na nj.

Nadalje, postoji mogućnost kombiniranja usluga posrednika i samih stvari koje imaju makar neke kriptografske mogućnosti. Npr., one mogu potpisati poruku što će omogućiti razinu neporecivosti od same stvari.

Govoreći, s druge strane mreže, o primatelju, treba reći i da bi on trebao imati slična pravila na svojoj strani, odnosno u usmjerniku ako je primatelj stavljen iza njega. Na taj način ni pošiljatelj ni primatelj ne moraju ostvariti sigurnost – a mogu ju imati bez ikakvih promjena u nižem protokolu ili metaprotokolu što se tiče njih. Samo što onda moraju vjerovati čvorovima koji ostvaruju sigurnost umjesto njih. To ne ovisi previše o sigurnosti same lokalne mreže, jer je to druga kategorija: ako se ne vjeruje čvoru kojemu se sljedećemu šalje, zaludu sigurnost poveznice do njega.

„Pouzdane“ se i „nepouzdane“ mreže mogu npr. razlučiti po nižem protokolu. Naime, sustav pravila rukuje i tom informacijom prigodom obradbe poruka (u tekstu „`proto`“). Upotrebljava li se takvo lučenje, može se uzeti da bežični protokoli čine lokalnu mrežu a žični globalnu. Npr., tako se za neki sustav može definirati da se protokolu BLE može vjerovati, jer je došao iz izravne blizine, a protokolu TCP ne, jer je došao preko nesigurnoga Interneta.

U sustavu pravila ima i drugih mogućnosti, pa se mogu gledati i adrese nižega protokola (u tekstu „`addr`“). Naime, protokol TCP se šalje protokolom IP, a protokol IP podupire i lokalne i globalne adrese. Ako je lokalna mreža protokola IP pouzdana, onda se može njoj vjerovati na takav način, a Internetu, odnosno globalnim adresama, ne, sve uporabom pravilâ.

Primjerice, filtar za protokol može glasiti „`proto = 'tcp'`“, a filtar za adresu „`SUBSTRING(addr FROM 1 FOR 2) = X'C0A8'`“, jer su prva dva bajta lokalne adrese `192` i `168`, što je heksadekadski znak `C0` i znak `A8` [106]. Filtri se za sve spominjane potrebe mogu kombinirati običnom riječju „`AND`“ u jeziku SQL, kao i drugim logičkim operatorima iz toga jezika.

Isključivanje se sigurnosti može ostvariti pravilom koje premošćuje implicitna pravila i tako prihvaća ili prosljeđuje sve poruke koje dođu u čvor (u kojem će se slučaju u internoj

reprezentaciji poruke postaviti zastavica „`premosti_implicitna_pravila`“). Opet, to ima smisla samo ako je lokalna mreža fizički odvojena od mogućih napadača.

3.11. Usmjeravanje poruka

Usmjeravanje ostvareno metaprotokolom ne će biti potrebno ako je između početnoga i završnoga čvora poruke već uspostavljena izravna veza nižim protokolom. Primjerice, ako su oba čvora na lokalnoj mreži i paket s porukom izravno ili preko mrežne opreme dolazi nepromijenjen do odredišta. Slično je ako su oba čvora povezana preko protokola TCP, protokola UDP ili viših protokola. U ostalim će slučajima biti potrebno i usmjeravanje. Ono sa sobom donosi i neke probleme, kao što je to otkrivanje dvostrukih poruka.

U identifikaciji se poruke upotrebljava i njezin identifikator iz polja `ID`. Identifikator se nasumično generira kad se prvi put šalje poruka čvoru – ili prvi put odgovara čvoru. Svaka sljedeća poruka sadržava identifikator koji je za `1` veći od onoga u prethodnoj. Uvećavanje se čini po modulu `256` tako da nakon identifikatora `255` ide identifikator `0`.

Jedinstvena identifikacija poruke, koja se osim polja `ID` sastoji i od polja `SRC` i `DST` (implicitno izračunanih ako nisu dio same poruke) omogućuje otkrivanje duplikata. To je posebno važno u usmjeravanju poruka, pogotovo ako se više takvih poruka kreće različitim rutama. Međutim, zbog toga treba od svake poslane poruke na neko vrijeme pamtiti i identifikaciju te poruke. Nadalje, zbog uporabe takve identifikacije koja ne mora biti lokacijski ovisna, čvor se jednostavno može premještati iz jedne lokalne mreže u drugu, bez ikakve promjene u tom čvoru ili ostalim čvorovima u tim mrežama. Algoritam koji se u drugim mrežama, primjerice mrežama protokola IP, upotrebljava za sprječavanje petljâ, jest mehanizam *vrijeme života* (engl. Time to Live, TTL) [107], koji smanjuje brojač u zaglavlju paketa za jedan svaki put kad paket dođe do sljedećega čvora; kad taj broj dođe do ništice paket se odbacuje. Ipak, usmjeravanje uz pomoć mehanizma TTL ne osigurava više ruta iste poruke do cilja. To jest, protokol IP ga omogućuje istodobno i preširoko i preusko, tako da može slobodno postojati više ruta koje imaju zajedničke dijelove u slučaju da u nekom trenutku više čvorova uhvati poruku, ali ti zajednički dijelovi moraju obvezno biti na samom kraju rute i tjeraju taj skup izgledati kao stablo. To proizlazi iz definicije usmjeravanja protokola IP gdje nema otkrivanja duplikata i ima samo jedan mogući sljedeći skok k odredištu. Usmjeravanje uz pomoć polja `ID` osigurava postojanje više ruta na jednostavniji način. Odnosno, metaprotokol „spljošćuje“ (engl. *flatten*) te zajedničke

dijelove u samo jedan jer ne prosljeđuje više puta istu poruku jednakom rutom, a isto tako i može proslijediti poruku na više odredišta pa skup ruta više ne mora izgledati uvijek kao stablo.

Uporaba je mehanizma TTL jednostavna i ne traži pohranu podataka o nedavno poslanim porukama, kao što je to potrebno za opisani postupak s identifikatorima. Međutim, s obzirom na to na se vrijednost TTL mijenja svakim skokom, to bi otežalo uporabu kriptiranja i potpisivanja poruke.

Provjera poljâ `SRC`, `DST` i `ID` omogućuje da se ista poruka ne proslijedi dva puta. Očekuje se da jednostavna stvar u razmjerno kratkom vremenu, recimo 60 s, ne će generirati 256 ili više različitih poruka za jednako odredište. Ako je ista poruka dva puta došla do nekoga čvora, ali preko dvaju različitih čvorova, i to u kratkom vremenskom razdoblju u kojem se još čuvaju zapisi proslijeđenih poruka, onda se takva poruka smatra dvostrukom porukom i odbacuje.

Čvorovi radi uštede energije ne moraju uvijek raditi, odnosno biti komunikacijski aktivni. Stoga poruka može ići različitim rutama od početnoga do odredišnoga čvora, ovisno o vremenu slanja ili drugim vanjskim čimbenicima. To može ići bilo razašiljanjem ili jednostavnim višestrukim slanjem drugim čvorovima. Naime, kad čvor šalje, ili prosljeđuje, poruku, sljedeći čvor koji tu poruku treba dobiti – identificiran po polju `DST` – može biti dostupan na više načina. Ti su načini znani čvoru preko prijašnjih razmjena poruka, ili se za to najprije mogu poslati poruke `HELLO`, ili se oni na neki drugi način dostaviti čvoru (npr. ručno).

Nadalje, ovisno o pravilima, poruka se može proslijediti ili na sve rute ili možda na samo jednu. Podrazumijevano se šalje svima, ali pravila mogu i to promijeniti. Recimo, može se provjeriti koja je najsvježija ruta i poslati na nju. Za takva pravila programer bi trebao znati interne strukture u bazi, jer za sad to nije zamišljeno kao nešto pruženo korisniku kao npr. naredbe `ENCRYPT` i `SIGN`.

Napredniji čvorovi moraju pamtiti te neke podatke o daljinskim čvorovima. Prvo, koje je zadnje polje `ID` poslano kojemu čvoru od ovoga čvora, za identifikaciju poruke. Drugo, kad je koje polje `ID` za koji čvor zadnji put dobiveno od kojega čvora, u kraćem razdoblju za otkrivanje duplikata. Treće, kad je preko kojega protokola i koje adrese zadnji put primljena poruka od kojega čvora, u duljem razdoblju za pamćenje ruta.

Još napredniji čvorovi mogli bi imati neku svoju dodatnu metriku i izložiti ju zapravila u nekoj tablici. S druge strane najjednostavniji čvorovi mogu uvijek upotrebljavati razašiljanje (svima) i ništa drugo ne moraju znati o rutama. Razašiljanje se može uključiti u pravilima ili jednostavno ručno upotrijebiti za slanje nekim nižim protokolom.

Kad se upotrebljava razašiljanje neki vidovi sigurnosti mogu isto biti ostvareni nižim protokolima, ako ih oni podupiru. Nije li to dovoljno, potrebno je upotrebljavati kriptiranje i/ili potpisivanje koje omogućuje metaprotokol.

3.12. Ostvarene radnje Interneta stvari

U potpoglavlju su 3.1 opisane potrebne radnje za sustave IoT-a. U nastavku je opisano kako se one mogu ostvariti predloženom arhitekturom, tj. metaprotokolom i modelom čvora.

3.12.1. Slanje podataka

Slanje podataka ostvareno je porukom `DATA`, koja se može slati kao samostalna poruka, kao odgovor na upit i kao odgovor na pretplatu. Samostalna poruka može biti generirana i periodičnim pravilom ili kad se primi neki okidač u tijelu ili zaglavlju poruke. Slanje podataka vrlo je raznovrsno u smislu da se mogu upotrebljavati bilo koji tipovi podataka jezika SQL. To pokriva više-manje sve smislene tipove i sigurno je jednako raznoliko koliko i svi modeli koji su temeljeni na programerskim tipovima podataka. Može se poslati jedan podatak, cijeli redak ili cijela tablica podataka. Ne može se poslati više tablica, ali to nije ni potrebno, pogotovo jer se upitom tablice mogu kombinirati. Ne može se ni poslati istodobno više po tipu različitih redaka podataka, ali to nije ni previše smisleno. Umjesto toga može se poslati jedan binarni skupni podatak koji se na drugoj strani dekodira kako god, ili rijetko popunjena (engl. *sparse*) tablica.

3.12.2. Slanje podataka što kraćom porukom

Slanje je podataka što kraćom porukom ostvareno porukom tipa `QUICK`. Ona je zapravo posebno kodirana poruka tipa `DATA` i automatski se mijenja iz jednoga oblika u drugi tijekom slanja odnosno primanja. Poruka omogućuje najbrže moguće slanje podataka s jednoga jednostavnoga senzora koji može takvom porukom poslati jednu 1-bajtnu vrijednost. Ta vrijednost ne mora biti izravna fizička veličina kao npr. temperatura u Celzijevim stupnjima, nego može značiti nešto drugo. Preslikavanje vrijednosti iz domene

uređaja u 1-bajtni broj u svakom je slučaju povoljnije za potrošnju energije nego slanje više bajtova umjesto jednoga. Npr., dokle se jačina struje za rad antene mikrokontrolera mjeri u miliamperima (mA), običan se rad mjeri u mikroamperima (μA) [108].

3.12.3. Zahtijevanje podataka

Zahtijevanje podataka ostvareno je porukom `SELECT`. Upit može biti od vrlo jednostavna do vrlo složena, što omogućuje uporaba naredbe `SELECT` jezika SQL. Primjerice, upit se može slati jednostavnoj stvari kad se želi odčitavanje senzora, ali i složenijoj stvari ili kakvu čvoru od kojega se može tražiti dohvat podataka iz baze. Kad se to upari s predloženim kraćenjem poruka, dobiva se sažeto i moćno oruđe za dohvat podataka.

3.12.4. Zahtijevanje obrađenih podataka

Zahtijevanje obrađenih podataka isto je ostvareno porukom `SELECT`. Puno se toga može uvrstiti u upit `SELECT`, zapravo cijeli standard SQL:2016, i izmišljati alternativno rješenje ne čini se potrebno. Upit `SELECT` se može izravno poslati bazi ili, ako je jednostavan upit, obraditi i bez nje. Pošalje li se presložen upit prejednostavnu čvoru, takva se poruka „odbija“ porukom `OPERATION_UNSUPPORTED`.

3.12.5. Pretplata na podatke

Pretplata se ostvaruje porukom `SELECT_SUBSCRIBE`, što je zapravo proširena poruka `SELECT`. Pretplatiti se može na što bilo, uključujući posebno obrađenu kombinaciju podataka više čvorova. Mehanizmi jezika SQL kao što su „(`CREATE`) `TRIGGER`“ i „(`CREATE`) `VIEW`“ omogućuju da se pretplata odvija skoro automatski.

3.12.6. Odjava s pretplate

Odjava se s pretplate ostvaruje porukama `UNSUBSCRIBE` i `UNSUBSCRIBE_ALL`; prvom se odjavljuje sa samo jedne pretplate identificirane rednim brojem, dokle se drugom odjavljuje sa svih pretplata u tom čvoru.

3.12.7. Ostvarivanje povjerljivosti

U zaglavlju svake poruke piše zahtijeva li se povjerljivost od prvoga, početnoga, izvorišta, čvora `SRC`, do zadnjega, konačnoga, odredišta, čvora `DST` ili ne. Zahtijeva li se, to se može ostvariti nižim protokolom, što je preporučljivo, ili se to može ostvariti kriptiranjem poruke

kroz metaprotokol. Preporučuje se i uporaba obaju načina, ako je moguće, jer niži protokol obično ne može osigurati povjerljivost na cijelom putu, a metaprotokol često nije toliko sofisticiran kao on pa je bolje ne izlagati ga zlonamjernim čvorovima.

3.12.8. Ostvarivanje autentičnosti

Slično ostvarivanju povjerljivosti ostvaruje se i autentičnost, ako je zastavica `A` postavljena u zaglavlju poruke. Autentičnost se ostvaruje kroz digitalni potpis.

3.12.9. Potvrđivanje poruke

Potvrđivanje je poruke ostvareno porukom `ACKNOWLEDGMENT`, koja se šalje kao odgovor na primljenu poruku, ako je ona imala postavljen bit `K` u zaglavlju.

3.12.10. Upravljanje pogrješkama

Nisu li elementi poruke (ne uključujući tijelo) dobro oblikovani, poruka se odbacuje bez dojave pogrješke. Za pogrješke u tijelu poruke može se upotrebljavati poruka tipa `PAYLOAD_ERROR`. Ne može li odredišni čvor izvršiti zahtijevanu operaciju, odbijanje može vratiti porukom tipa `OPERATION_UNSUPPORTED`.

3.12.11. Upravljanje informacijama o daljinskim čvorovima

U informacije o daljinskim čvorovima u jednom čvoru spadaju podatci za duplikate, podatci za identifikaciju i podatci za rutu. Ruta se do nekoga čvora automatski ažurira primitkom poruke od toga čvora, a mogu se upotrebljavati i poruke `HELLO`. Isto tako, te se informacije mogu u čvor upisati ručno, npr. u početnom postavljanju čvora, ili poslije na neki način (preko sučelja za *web*, spajanja preko posebna priključka, npr. upravljačke serijske veze, i sl.). Registracija čvora u drugom čvoru događa se automatski (slanjem poruka) ili ručno (kroz sučelje). Deregistracija se događa slično automatski (istekom rute) ili ručno (kroz sučelje).

3.12.12. Upravljanja podatcima nekoga daljinskoga čvora

Pohrana podataka iz primljenih poruka nekoga drugoga čvora u bazu može biti automatska, ako to čvor podupire. Brisanje nije predviđeno jer svi podatci imaju vremenski biljeg po kojem se razlikuju stari podatci od novih. Ako je potrebno brisanje, to se može ostvariti vremenskim pravilom, ili pravilom na neku posebnu poruku, ili ručno.

3.12.13. Postavljanje okidača

Postavljanje okidača izvodi se kroz sustav pravila. Okidač može biti slanje poruke, primanje poruke ili vrijeme odnosno period. Okidač može biti jednostavan izraz, a može i uključivati znanje programiranja u jeziku SQL. Za ostvarivanje se pretplate upotrebljavaju okidači koji se postavljaju na zahtjev za pretplatu (nije ih potrebno ručno definirati).

3.12.14. Upravljanje pravima

Upravljanje pravima izvodi se kroz konfiguraciju čvora. Svaka tablica ima vlasnika i popis korisnika koji joj mogu pristupiti za čitanje podataka. Ako je za neku tablicu čitač, odnosno vlasnik, postavljen na javnoga korisnika, to obilježava da mogu svi čitati, odnosno pisati, podatke u tu tablicu. Za pristup podatcima kroz eksplicitna pravila gledaju se vlasnici tih pravila i njihova pristupna prava.

3.13. Turing-potpunost

Treba naglasiti da je predloženi model podataka i poruka Turing-potpun, bez obzira na to što se cijelo vrijeme dodaju novi podatci. To bi trebalo biti jasno iz rasprave o vremenskim biljezima u prošlom potpoglavlju.

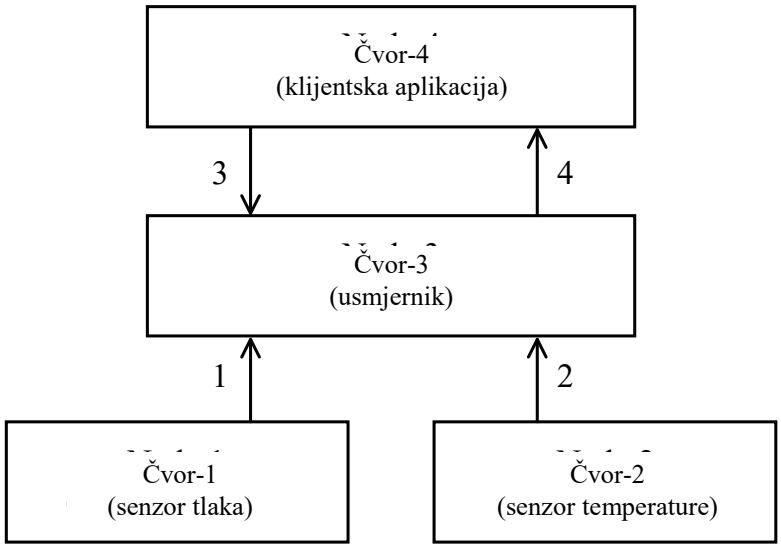
Ovaj je model „Turing-potpun“ u smislu u kojem je takav i svaki popularni programski jezik kao C, C++, C#, Java… Odnosno, u smislu da bi mogao izvesti sve što i Turingov stroj kad ne bi bio ograničen i vremenom i prostorom. Zapravo je i ovaj sustav i ti programski jezici nešto što se može zvati „LBA-potpun“ [109], odnosno potpun s obzirom na *linearni ograničeni automat* (engl. linear bounded automaton, LBA). Automat LBA je ograničeniji model od Turingova stroja u smislu da ima konačnu vrpcu: Turingov stroj ima beskonačnu, što je neostvarivo u stvarnom svijetu. Usprkos tomu, izneseni bi se dokaz podudarao s dokazom koji se upotrebljava za te programske jezike kad se izrijekom gleda Turingov stroj. Odnosno, pokazalo bi se da se Turingov stroj može simulirati ovim sustavom, zanemarujući nužnokonačnu vrpcu kojom će taj „stroj“ rukovati.

4. PRIMJERI SCENARIJA

U ovom je poglavlju prikazano nekoliko scenarija uporabe koji prikazuju tipična ponašanja sustava. Svaki scenarij ima nekoliko čvorova koji međusobno komuniciraju predloženim metaprokotolom. Mogućnosti su čvorova različite; neki imaju instaliranu prototipnu programsku potporu, a drugi ručno ostvaruju samo dio metaprotokola. Svaki čvor ima neku pridijeljenu ulogu, npr. senzora ili posrednika ili poslužitelja.

4.1. Primjer 1. Osnovne operacije

Prvi primjer scenarija prikazan je na Sl. 4.1 i sastoji se od četiriju čvorova. Pri tom čvorovi Čvor-1 i Čvor-2 su jednostavne stvari IoT-a koje komuniciraju nekim podatkovnim protokolom s Čvorom-3, primjerice protokolom BLE. Čvor-3 i Čvor-4 međusobno komuniciraju složajem TCP/IP. U ovom se primjeru ne razmatra sigurnost te se ona ne će ni upotrebljavati.



Slika 4.1. Primjer jednostavna sustava

Čvor-1 čini senzor tlaka koji svakih 60 min šalje trenutačni tlak Čvoru-3. Slično, Čvor-2 svakih 60 min odčitava temperaturu i to šalje Čvoru-3.

Čvor-3 služi kao agregator podataka od obaju čvorova. Najvjerojatnije bi to bio neki usmjernik IoT-a uključen u struju, za razliku od ovih dvaju senzora koji bi radili na baterije i bili prijenosni. Čvor-4 ima ulogu korisnika (aplikacije za korisnika) kojega s vremena na

vrijeme zanimaju podatci senzora, u izvornom ili obrađenom obliku. Prvo će se razmotriti slanje podataka od senzorâ k usmjerniku.

Scenarij koji se ovdje razmatra sastoji se od više poruka. Prva, poruka 1 na Sl. 4.1, poslana je iz Čvora-1. Čvor-3 pri tom pohranjuje i podatke iz te poruke svojom podrazumijevanom radnjom pohrane. Čvor pohrani i metapodatke o dostupnosti Čvora-1. Na sličan način i Čvor-2 šalje svoje podatke: poruku 2 na Sl. 4.1. Navedene razmjene poruka i operacije ponavljaju se svakih približno 60 min. Nakon nekoliko dana korisnik, tj. Čvor-4, šalje poruku Čvoru-3 u kojoj traži odčitane podatke za određeno razdoblje. Čvor-3 mu odgovara traženim podatcima.

Prva je poruka – poruka `DATA` koja, radi ilustracije metaprotokola, upotrebljava sva njegova polja. Ta je poruka prikazana u Tablici 4.I. S obzirom na to da sama poruka na razini metaprotokola sadržava sve elemente, nije potrebno ništa izvlačiti iz nižega sloja (npr. protokola BLE), tako da se za ovu poruku oblik te poruke na nižem sloju ne će razmatrati.

Tablica 4.I. Elementi poruke 1

Član	Vrijednost
Zaglavlje (`HD`)	`0b11111000`
Identifikator poruke (`ID`)	`0x48`
Duljina tijela (`LEN`)	`0x000A`
EUI odredišta (`DST`)	`0x333333FFFE333333`
EUI izvorišta (`SRC`)	`0x111111FFFE111111`
Tijelo poruke (`PL`)	„`tlak=1034;`“
Zaštitni kod (`CRC`)	`0xD5CA2B5C` (izračunano, nevažno)

Prvo polje poruke je zaglavlje. U zaglavlju poruke postavljene su jedinice za nazočnost svih elemenata poruke: identifikator, duljina, adrese odredišta i izvorišta, (tijelo poruke i) zaštitni kod. Adrese odredišta i izvorišta najčešće su preuzete iz nižega sloja, iako ne mora biti tako. Tijelo poruke „`tlak=1034;`“ sadržava 10 znakova što je i navedeno u elementu duljine tijela. S obzirom na to da tijelo poruke započinje znakom „`t`“ poruka ne može biti ništa drugo nego poruka `DATA`. U ovom scenariju Čvor-3 unaprijed zna što mu čvorovi

šalju i jedinica za tlak ovdje nije neophodna. U općenitijem slučaju ta bi se jedinica mogla dodati u poruku ili izravno na vrijednost („`tlak='1034 hPa';`“) ili kao dodatno polje („`tlak,jedinica=1034,'hPa';`“).

Kad primi prvu poruku Čvor-3 automatski stvara tablicu u koju će pohranjivati podatke Čvora-1. Postoji li već tablica za taj čvor, treba provjeriti je li u njoj stupac „`tlak`“. Nije li, on se doda, a ako jest, provjeri se slažu li se tipovi i treba li ih proširiti. Po metaprotokolu je naziv tablice „`t111111fffe111111`“. Stupci u tablici jesu „`t`“ za vremenski biljeg i stupac „`tlak`“ po dobivenom podatku u tijelu poruke. Po dobivenoj vrijednosti (`1034`) utvrđuje se tip toga stupca (`NUMERIC(4, 0)`).

Nije li riječ o prvoj poslanoj komunikaciji, ta tablica već postoji na Čvoru-3 pa se ne mora stvarati. Ako je tablica postojala, ali nije imala stupac „`tlak`“, on se stvara. Primljena se poruka efektivno proširuje na „`tlak,t=1034,LOCALTIMESTAMP(4);`“ prije transformacije u naredbu `INSERT INTO t111111fffe111111(tlak, t) VALUES(1034, LOCALTIMESTAMP(4));`.

Za prethodnu se je poruku `DATA` sve osim tijela moglo i ispustiti jer se je moglo izvesti iz same poruke (nižega sloja). Slijedi opis takve komunikacije na primjeru Čvora-2.

Čvor-2, za razliku od Čvora-1, šalje svoje odčitavanje ne porukom `DATA`, nego porukom `QUICK`. Poruka `QUICK` ima samo 1 bajt, a ispušta se sve ostalo, tako da je poruka ukupno velika samo 1 bajt (ne računajući podatke nižih slojeva). Neka Čvor-2 treba poslati poruku koja bi u obliku poruke `DATA` bila: „`d=20;`“. Poruka `QUICK` sadržava samo 8-bitnu vrijednost koja mora biti u rasponu `0x00-0xFF`, tj. „`\x14`“. Primjer takve poruke vidi se u Tablici 4.II. U ovom primjeru za „niži sloj“ upotrebljava se protokol BLE. Vrsta paketa koji se šalje jest tip *oglašavajuće nespojivo javljanje* (engl. Advertising Non-Connectable Indication, `ADV_NONCONN_IND`), najjednostavniji paket protokola BLE za razašiljanje poruke bez tzv. mogućnosti skeniranja (engl. *scannability*) toga uređaja i bez tzv. mogućnosti povezivanja (engl. *connectability*) na uređaj. Treba reći da se takvim paketom ne mogu slati preduge poruke, nego tomu služi tip *oglašavajuće prošireno javljanje* (engl. Advertising Extended Indication, `ADV_EXT_IND`) iz protokola BLE 5.0 koji može poslati do 255 B. Inače se može slati najviše 31 B. U slučaju još duljih poruka trebalo bi uspostaviti vezu protokola BLE uparivanjem dvaju uređaja, ali pitanje je koliko je to korisno u ovom slučaju.

Tablica 4.II. Elementi paketa protokola BLE koji prenosi poruku 2

Član	**Vrijednost**
Preambula (`Preamble`)	(1 bajt)
Pristupna adresa (`Access Address`)	(4 bajta)
Zaglavlje (`Header`)	(2 bajta)
Adresa oglašivača (`AdvA`)	`0x222222222222`
Podatci oglašivača (`AdvData`)	`0x14` (poruka 2 – poruka `QUICK`)
Zaštitni kod (`CRC`)	(3 bajta)

Čvor-3 gleda prvo duljinu cijele poruke na razini metaprotokola, a to je element *oglašivačevi podatci* (engl. Advertiser Data, `AdvData`) iz Tablice 4.II), i kako je on 1 bajt, zaključuje da je to poruka `QUICK`. Tad se sve ispuštene informacije moraju ili izvući iz nižega protokola ili izračunati ili staviti na podrazumijevane vrijednosti. Prvo, zaglavlje se stavi na vrijednost `0b00000000` jer je to podrazumijevana vrijednost. Polje `LEN` se izvlači iz nižega protokola (vrijednost `1`). Identifikator izvorišta se dobije iz polja `AdvA` uz algoritam pretvorbe u identifikator EUI-64 iz identifikatora EUI-48. Konkretno, to bi bilo ubacivanje 2-bajtne vrijednosti `0xFFFE` u sredinu. Identifikator odredišta stavi se na identifikator EUI-64 Čvora-3 koji se pročita iz njegova sučelja protokola BLE, ali se i dalje pamti da je poruka bila razaslana.

Polje `CRC` se izračunava iz cijele 1-bajtne poruke, a njegov se prvi bajt upotrebljava kao polje `ID` poruke. Podatak se „`\x14`“ preobličuje u „`d=20;`“ i zapiše u tablicu „`t222222fffe222222`“, odnosno u njezine stupce „`d`“ i „`t`“ ubace se vrijednosti „`20`“ i „`LOCALTIMESTAMP(4)`“.

U sljedećim primjerima ne će se više raščlanjivati konkretni niži protokol, a ni polja metaprotokola, nego samo njegovo tijelo poruke koje je bitno za demonstraciju nekih svojstava metaprotokola.

Sljedeći važan događaj jest kad Čvor-4 zahtijeva agregirane podatke Čvora-1 i Čvora-2 za svoje potrebe (poruka 3 na Sl. 4.1). Konkretno, zanimaju ga odčitavanja senzorâ na dan 1. siječnja 2022. između 10h i 14h. Za to mu treba poruka `SELECT`, a ona u tekstualnom obliku metaprotokola glasi kako slijedi.

```
SELECT ALL t1.tl AS tl, t2.d AS te FROM t111111fffe111111 AS
t1 INNER JOIN t222222fffe222222 AS t2 ON CAST(t1.t AS DATE) =
CAST(t2.t AS DATE) AND EXTRACT(HOUR FROM t1.t) = EXTRACT(HOUR
FROM t2.t) WHERE CAST(t1.t AS DATE) = DATE '2022-01-01' AND
EXTRACT(HOUR FROM t1.t) BETWEEN 10 AND 14 ORDER BY t1.t ASC;
```

Ova poruka konkretno ima 307 bajtova i prilično je nezgrapna jer u standardu jezika SQL rad s vremenskim biljezima nije prejednostavan za pisanje. Ovdje se tablice spajaju na osnovi jednaka sata, ali je to dvostruki uvjet jer treba specificirati i jednak dan.

S druge strane ako se poruka kodira po opisanim pravilima zamjene ključnih riječi jednim bajtom i izbacivanja suvišnoga, dobiva se sljedeći znakovni niz.

```
\xDE\x85t1.tl\x87tl,t2.d\x87te\xACt\x11\x11\x11\xFF\xFE\x11\x
11\x11\x87t1\xB3\xB7t\x22\x22\x22\xFF\xFE\x22\x22\x22\x87t2\x
CECAST(t1.t\x87\x9B)=CAST(t2.t\x87\x9B)\x84EXTRACT(\xB1\xACt1
.t)=EXTRACT(\xB1\xACt2.t)\xFCCAST(t1.t\x87\x9B)=\x9B'2022-01-
01'\x84EXTRACT(\xB1\xACt1.t)\x8C10\x8414\xD1\x8Ft1.t\x88
```

Taj niz ima 159 bajtova i očito je za oko pola manji. Nije čitak običnomu korisniku, kao što se može i očekivati, pogotovo ako se ne ispisuje u ovakvu obliku nego kao oblik ISO 8859-1. Čvor-3 kao napredniji čvor ovo može dekodirati i izvršiti taj upit `SELECT`. Po odgovoru iz baze taj će se odgovor upakirati u poruku `DATA`, odnosno nešto ovako kako slijedi.

```
tl,te=1030,19;1031,20;1032,20;1031,21;1030,21;“
```

4.2. Primjer 2. Operacija pretplate

U drugom primjeru upotrebljava se ista slika sustava, Sl. 4.1, ali zanemarujući nacrtane strjelice. Naime, ovdje će se razmatrati malo naprednija funkcionalnost, a to je mogućnost pretplate na podatke. Ovdje se Čvor-4 želi pretplatiti na podatke u Čvoru-3 tako da dobije obavijest kad senzor temperature javi temperaturu veću od 20 °C. Da bi se to postignulo, on šalje poruku `SELECT_SUBSCRIBE` Čvoru-3 u sljedećem obliku:

```
SELECT ALL t1.d FROM t222222fffe222222 AS t1 WHERE t1.d > 20
                        SUBSCRIBE 1;
```

„ALL“ je svagdje do sad u tekstu „šumna riječ“, ali se piše da bi se razlikovalo od sintagme „SELECT DISTINCT“. Jasno je i da se ova poruka može skratiti, ali to ovdje nije potrebno.

Radi detekcije promjena Čvor-3 će na ovakvu poruku postaviti okidač jezika SQL: „CREATE TRIGGER AFTER UPDATE ON t222222fffe222222(…);“.

Na okidač se pozove funkcija SQL-a koja provjeri treba li nešto poslati pretplatniku. Da bi to bilo moguće ostvariti potrebno je pamtiti što je već poslano, tj. upotrebljavati „razliku“.

Uz pomoć baze se okidači pretplate mogu ostvariti na sljedeći način: pohrani se upit u bazi kao virtualna tablica VIEW jezika SQL i kroza nju se gleda jesu li se rezultati promijenili. Naime, prigodom primitka poruke SELECT_SUBSCRIBE izvede se upit SELECT i pohrani u novu tablicu, a kad se nešto promijeni u dinamički osvježenoj tablici VIEW u odnosu na tu tablicu, pokreće se vanjska funkcija koja će rezultate poslati pretplatniku.

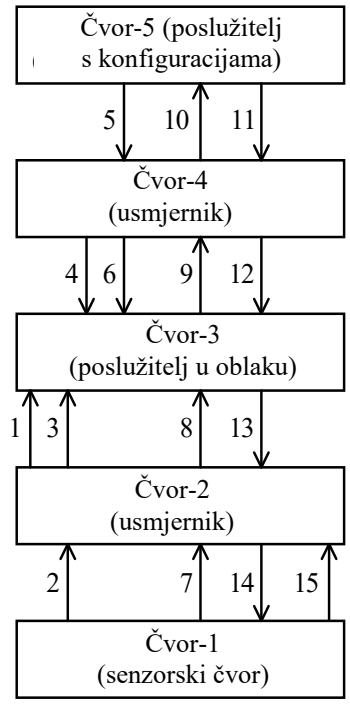
Pod pretpostavkom da su u bazi bile vrijednosti 19 20 21 22 22 23 23 22 22 21 i sad dođe vrijednost 21, razlika je samo vrijednost 21, jer su stari rezultat bile vrijednosti 21 22 22 23 23 22 22 21 a novi su vrijednosti 21 22 23 23 22 22 21 21. Dođe li vrijednost 20, ništa se ne mijenja jer se ne mijenja ni stari ni novi rezultat.

Kad Čvor-4 poželi odjaviti pretplatu, poslat će poruku „UNSUBSCRIBE 1;“ Čvoru-3. Umjesto slanja poruke UNSUBSCRIBE može se poslati i poruka UNSUBSCRIBE_ALL.

Dodatno, ako se želi biti siguran da je odjava primljena i razumljena, za poruku UNSUBSCRIBE će u zaglavlju biti postavljen bit K i Čvor-3 će poslati potvrdu. Potvrda će biti poruka ACKNOWLEDGMENT u tekstualnom obliku npr. „ACKNOWLEDGMENT 0x01;“, odnosno samo „K\x01“ ako se šalje u binarnom obliku.

4.3. Primjer 3. Raspodijeljeni sustav

Smisao je sljedećih dvaju primjera pokazati da se predložena arhitektura ne odnosi samo na lokalne mreže. Sustav na kojem se razrađuje sljedeći primjer prikazan je na Sl. 4.2. Ondje se vidi višerazinski raspodijeljen sustav gdje je Čvor-3 na vrhu poredaka a Čvor-1 i Čvor-5 na dnu. Čvor-2 i Čvor-4 su međučvorovi između njih. Čvor-3 je zato prikazan u sredini slike.



Slika 4.2. Primjer raspodijeljenoga sustava

Čvor-1 je neki senzor, Čvor-2 njegov usmjernik u istoj lokalnoj mreži, Čvor-3 podrazumijevani globalni usmjernik za lokalne usmjernike. Čvor-5 je neki čvor u drugoj lokalnoj mreži čiji je lokalni usmjernik Čvor-4. Čvor-5 ima određene konfiguracijske podatke ili nešto drugo korisno za Čvor-1 i ostale čvorove. Recimo da ti podatci nisu u oblaku zato što se mijenjaju ovisno o stanju u lokalnoj mreži. Čvor-3 i za tu lokalnu mrežu isto služi kao globalni povezivač s ostalima – vjerojatno se nalazi u oblaku i ima statičku adresu protokola IP. Ona je zapisana u konfiguraciji usmjernikâ kao podrazumijevano odredište.

U lokalnoj mreži čvorovi su povezani i zemljopisno, tako da se u ovom primjeru između Čvora-1 i Čvora-2 – odnosno Čvora-5 i Čvora-4 – upotrebljava neki bežični podatkovni protokol kao što je protokol BLE. Čvor-3 se nalazi u Internetu i do njega se u svakom slučaju dolazi preko nekoga protokola iznad protokola IP. To može biti protokol TCP, protokol UDP, protokol HTTP, kombinacija *sigurni HTTP* (engl. HTTP Secure, HTTPS) ili nešto drugo. Za dalje razmatranje konkretni protokoli nisu ni važni. Ovdje se samo želi istaknuti da protokoli mogu biti različiti, na isti način kao što su različite i razine. Naime, metaprotokol može sve to premostiti i upotrebljavati se na svim razinama i preko svih protokola. Naravno, ako ima odgovarajuće module UPM za učahurivanje.

Čvor-3 početno ne mora poznavati adrese ostalih čvorova (ili i da postoje). Njihove adrese doznat će kad od njih dobije poruke. S druge strane, Čvor-2 i Čvor-4 moraju poznavati adresu Čvora-3, unesenu na neki način u te čvorove.

Da bi se niži čvorovi „javili“ višima, trebaju im nešto poslati. Nemaju li ništa korisno, uvijek mogu poslati poruku `HELLO`, samo da im obznane svoje postojanje.

U ovom se scenariju upravo poruke `HELLO` upotrebljavaju da bi se čvorovi najavili svojim usmjernicima odmah na početku svojega rada. Poredak slanja poruka se vidi na Sl. 4.2: Čvor-2 se javi Čvoru-3 (poruka 1), Čvor-1 Čvoru-2 (poruka 2), Čvor-2 proslijedi to Čvoru-3 (3). Prosljeđivanje ide kroz implicitno ponašanje sustava gdje se pozdravne poruke prosljeđuju podrazumijevanim odredištima, ili po dodatno postavljenu pravilu. Tim je Čvor-2 najavio Čvoru-3 da je Čvor-1 dostupan preko njega, Čvora-2.

Na drugoj se strani sustava događa slično. Čvor-4 se javi Čvoru-3 (4), Čvor-5 Čvoru-4 (5), to bude proslijeđeno Čvoru-3 (6), i tim je početni dio gotov. Pošto se je tim konfiguriralo usmjeravanje, slijedi konkretan rad. U ovom scenariju Čvor-1 periodično šalje svoje vrijednosti Čvoru-2. U prikazanom se scenariju veličina perioda (za pojedino vremensko razdoblje, npr. određeni dan) definira u Čvoru-5. Stoga Čvor-1 mora od Čvora-5 zatražiti taj podatak.

Čvor-1 ne može izravno komunicirati s Čvorom-5, čak on i ne mora znati gdje je taj čvor, u lokalnoj mreži ili negdje drugdje. Stoga on poruku koja kao odredište (polje `DST`) ima Čvor-5 pošalje (neizravno, razaslano) i Čvor-2 ju primi. Čvor-2 ne mora poznavati adresu Čvora-5, tj. Čvora-4 koji je veza put Čvora-5, nego on poruku (8) prosljeđuje Čvoru-3 koji bi trebao znati za Čvor-5. Isto tako, ako ni Čvor-2 ne zna je li možda Čvor-5 u lokalnoj mreži, on bi poruku opet mogao poslati neizravno i u lokalnu mrežu (možda i prije nego li šalje Čvoru-3) jer je poruka bila razaslana nižim protokolom. Ali odatle odgovor ne će dobiti, pa će se u nastavku razmatrati samo odgovori koje prima od Čvora-3.

Čvor-3 zna gdje se nalazi Čvor-5 (preko prijašnjih poruka `HELLO`) i neizmijenjenu poruku prosljeđuje Čvoru-4 koji ju onda konačno prosljeđuje Čvoru-5 koji je i naveden kao odredište ove poruke (polje `DST`). U komunikaciji se između svakih dvaju čvorova upotrebljava protokol koji se je i prethodno upotrebljavao u komunikaciji. Npr. protokol BLE između Čvora-1 i Čvora-2 te između Čvora-4 i Čvora-5, kombinacija HTTPS između Čvora-2 i Čvora-3 te između Čvora-3 i Čvora-4. Paket koji se šalje sadržava poruku `SELECT`:

```
SELECT ALL period FROM konfiguracija WHERE "čvor_id" =
X'111111FFFE111111';
```

Čvor-5 sadržava posebnu tablicu „`konfiguracija`“, koja je ručno stvorena i popunjena podatcima za svaki senzor koji je u njoj identificiran svojim identifikatorom EUI. Čvor-5 najprije provjeri valjanost upita `SELECT` i onda proslijedi upit k bazi, gdje se provjere i prava pristupa toj tablici od zadanoga čvora (ako su ona poduprta).

Za ovaj konkretan zahtjev baza vrati odgovor da je „`period`“ jednak „`20`“. To se upakira u poruku `DATA` „`period=20;`“, a identifikatori su sad obrnuti. Čvor `DST` je Čvor-1, čvor `SRC` je Čvor-5. Poruka se pošalje njegovu podrazumijevanomu usmjerniku – Čvoru-4 (11) – ali ne razaslano, nego samo njemu. Naime, malo je prije bilo zabilježeno da se preko te poveznice može doći do Čvora-1, zato što je odatle došla poruka od toga čvora. Istim se dakle protokolom i vrati u drugom smjeru, a slično čini i Čvor-4. On zna da poruku treba poslati Čvoru-3, iako bi to učinio i da ne zna, jer je on njegov usmjernik (12).

Čvor-3 dalje zna da treba proslijediti poruku Čvoru-2 (13), a on prosljeđuje Čvoru-1 (14). Ruta je, kako je rečeno, bila uspostavljena i postoji, pogotovo u tako kratku razdoblju gdje još nije ni blizu „istekla“.

Čvor-1 sad zna svoj period i može početi slati podatke (15). Šalje ih svakih 20 min, jer on zna vremensku jedinicu. Naime, već je govoreno da se jedinice ne zapisuju. Prigodom slanja svojih odčitavanja može se poslužiti porukom `DATA` kao što je „`"odčitavanje"=123;`“, ili porukom `QUICK` „`\x7B`“ („`d=123;`“).

4.4. Primjer 4. Sigurnost i privatnost

Mogućnosti su ostvarivanja sigurnosti kroz predloženu arhitekturu ovdje prikazane kroz sljedeći scenarij. U tom se scenariju upotrebljava sigurnost nižega protokola kad se to može, a ručno osiguravanje poruke mehanizmima metaprotokola kad to ne bude moguće, odnosno kad to ne bude dovoljno.

Čvorovi su isti kao na Sl. 4.2, ali s drugim scenarijem. U ovom se scenariju traži sigurnost poruke poslane od Čvora-1 do Čvora-5, tj. da nitko ne može pročitati ili promijeniti te podatke osim Čvora-5.

U sljedećim se dvama primjerima pretpostavlja slanje poruke od Čvora-1 do Čvora-5. Tijelo poruke neka je „`podatak=56;`“. Dva primjera razlikovat će se u tom što u prvom

Čvor-1 i Čvor-5 mogu sami ostvariti sigurnost dokle se u drugom primjeru za to oslanjaju na prvi sljedeći čvor.

U prvom primjeru Čvor-1 sam ne upotrebljava nikakvu sigurnost, ali ju traži u prijenosu. Stoga je u Čvoru-2 konfigurirana posrednička sigurnost iz potpoglavlja 3.10, tj. pravila koja se pokreću kad se primi poruka od Čvora-1 u kojoj su postavljene zastavice `C` i `A` te komplementarna pravila kad izvana dođe poruka za Čvor-1. Prva pravila glase:

```
ako (SRC = Čvor-1 && HD.C) ENCRYPT; i
ako (SRC = Čvor-1 && HD.A) SIGN;,
```

gdje je uvjet u zagradama konkretni filtar, a naredba s točka-zarezom konkretna modifikacija poruke. Slično, komplementarna pravila imala bi „`DST`“ i „`DECRYPT`“, odnosno „`DST`“ i „`VERIFY`“ u obrnutom redoslijedu. Adrese bi čvorova trebale biti u heksadekadskom obliku jezika SQL umjesto naziva `Čvor-1`.

Čvor-2 djeluje u ime Čvora-1 i mora imati potrebne ključeve. To znači da, ako kriptira, mora imati javni ključ Čvora-5; ako dekriptira, privatni ključ Čvora-1. Alternativno, mogao bi obavljati njegove radnje u svoje ime. Npr. ako Čvor-1 želi nešto dohvatiti od Čvora-5, može i Čvor-2 to dohvatiti pa mu proslijediti. Ali za takvo bi nešto trebala dodatna nešto složenija pravila s mogućnošću rada korak po korak.

Čvor-5 u ovom primjeru sam rješava sigurnost, ne delegira ju Čvoru-4. Što se tiče veze između Čvora-1 i Čvora-2, odnosno Čvora-4 i Čvora-5, pretpostavlja se da su njihove veze sigurne, odnosno da je sigurnost ostvarena fizičkom nedostupnošću ili nižim protokolom, npr. kriptiranom vezom preko protokola BLE.

U drugom primjeru Čvor-1 pošalje nezaštićenu poruku Čvoru-2, ali s postavljenim zastavicama `C`, `A` i `K`. Čvor-2 postupa po navedenim pravilima, zaštićuje poruku ključevima Čvora-1 i Čvora-5 te prosljeđuje poruku Čvoru-3. Čvor-3 sam ne može pogledati što prenosi, ali za nj to ionako nije namijenjeno, on to prosljeđuje Čvoru-4 koji na sličan način to prosljeđuje Čvoru-5. Čvor-5 na to odgovara potvrdom, opet ju osiguravši na sličan način tako da ju može otpakirati samo Čvor-1, tj. Čvor-2 njegovim ključevima. Obrnutim putom poruka dolazi do Čvora-2 koji ju raspakira i prosljeđuje Čvoru-1, čim je završeno sigurno slanje poruke uz potvrdu da je ona i primljena.

U metaprotokol nisu uključena automatska ponovna odašiljanja u slučaju pogrješaka. Stoga bi i pogrješke sigurnosti trebalo detektirati/javljati dodatnim pravilima, koja bi

primjerice mogla ustanoviti da se radi o problemu sigurnosti, a ne vezi (npr. obične poruke `HELLO` prolaze, a osigurane ne). Pri tom bi najbolje bilo upotrebljavati poruke tipa `PAYLOAD_ERROR` ili `OPERATION_UNSUPPORTED` za dojavu komunikacijskih problema.

Kad bi Čvor-1 i Čvor-5 izravno komunicirali nekim „nižim“ protokolom koji podupire privatnost, npr. protokolom TLS, onda bi se taj protokol mogao upotrijebiti i ne bi trebalo aktivirati mogućnosti metaprotokola. U ovom primjeru ti čvorovi ne komuniciraju izravno nego preko ostalih čvorova u mreži (Čvor-2, Čvor-3 i Čvor-4). Kad bi se i njima dopustio pristup podatku, onda bi se mogli upotrebljavati sigurnosni mehanizmi veza između njih, ako postoje (veza Čvor-2 i Čvor-3 te Čvor-3 i Čvor-4) i na taj način spriječiti da se sadržaji otkriju ili omogući izmjena nekim drugim računalima koja prenose te podatke ali nisu dio metaprotokola (npr. različiti usmjernici koji povezuju te čvorove).

4.5. Primjer 5. Pojednostavljena izgradnja aplikacija programskom potporom

Sljedeći primjer prikazuje proširivanje metaprotokola programskom potporom (tzv. *middlewareom*) koja omogućuje automatizaciju rada s njim. Kao što se u OS-ima iznad mrežnoga sloja odnosno npr. sloja protokola IP izgrađuju npr. protokol TCP i protokol UDP za dodatne mogućnosti, tako se ovdje dodatne mogućnosti ostvaruju programskom potporom koja omogućuje upravljanje porukama. To upravljanje može biti stvaranje poruka, provjera pristignulih poruka, izvlačenje podataka iz poruka, slanje i primanje poruka, sustav pravila itd.

U uređajima ograničenima sredstvima takva programska potpora sadržavat će samo neke jednostavne operacije koje su potrebne, a na ostale zahtjeve odgovarati porukama `OPERATION_UNSUPPORTED`.

Na najjednostavnijim uređajima ta se potpora može prikazati jednom programskom petljom. Odčita li neki uređaj svakih 60 min ili na zahtjev svoj fizički senzor i tu vrijednost šalje svojemu usmjerniku, to se može ostvariti sljedećim programom na Sl. 4.3.

```
program {
    registriraj_događaj(NA_UPIT, na_upit);
    čini {
        pošalji_podrazumijevano(pročitaj_senzor());
        čekaj 60 min;
    }
}
na_upit(upit) {
    pošalji_podrazumijevano(pročitaj_senzor()));
}
```

Slika 4.3 Primjer programa

Operacije su `registriraj_događaj` i `pošalji_podrazumijevano` dio zamišljene programske potpore. Prva povezuje pozivanje funkcije na upit za događaj primitka poruke s upitom, a druga šalje zadani podatak preko podrazumijevane veze s podrazumijevanim elementima poruke (uz novu vrijednost senzora).

U složenijim sustavima gdje se može instalirati naprednija potpora kao što je implementirani prototip – zapravo nije ni potrebno programiranje nego se sve može zamijeniti pravilima i popratnim operacijama ponuđenima korisniku. Osnovne operacije mogu biti čitanje fizičkoga senzora u tablicu, pisanje vrijednosti na aktuator, slanje podataka drugim čvorovima, metaoperacije kao što su uključivanje ili isključivanje drugoga ili ovoga pravila, a sve to na neki događaj sa senzorom ili na neku poruku.

Pravila se mogu definirati u obliku relacijske tablice, i to ili ručno ili konfiguracijom za *web*. U sljedećim primjerima pravila će se opisati pseudokodom, a onda i relacijskim modelom. Dakle, uređaj opisan prijašnjim pseudokodom može se promijeniti u sustav pravila kako slijedi na Sl. 4.4.

```
PRAVILO 1:
tip = PERIODIČNO
period = 3600 (sekunda)
radnja:
    PROČITAJ_SENZOR(TEMP_25, TABLICA_0)
    POŠALJI_PODATKE(PODRAZUMIJEVANI_USMJ, TABLICA_0)

PRAVILO 2:
tip = U_PRIMANJU
FILTAR(PORUKA.tip = POR_TIP_SELECT)
```

```
radnja:
    PROČITAJ_SENZOR(TEMP_25, TABLICA_0)
    POŠALJI_DATA(PODRAZUMIJEVANI_USMJ, TABLICA_0)

PRAVILO 3:
tip = U_PRIMANJU
FILTAR(PORUKA.tip = POR_TIP_HELLO)
FILTAR(PORUKA.zastavice & POR_ZAS_ACK != 0)
radnja:
    POŠALJI_PORUKU(PORUKA.izvor, HELLO_PORUKA)
```

Slika 4.4. Primjer pravila 1, 2 i 3

Pravilo 1 periodično šalje odčitane podatke podrazumijevanomu usmjerniku – zamjenjuje petlju iz pseudokoda. U relacijskom obliku zapisuje se ovako na Sl. 4.5.

korisnik = `'root'`

identifikator = `1`

vrsta = PERIODIČNO

filtar = `3600`

radnja1 = NIŠTA

modifikacija = `NULL`

radnja2 = NIŠTA

radnja2_upit_naredba = `NULL`

radnja3 = POŠALJI_PORUKU_STVORENU_IZ_NAREDBE

radnja3_upit_naredba = `'procitaj_senzor.sh'`

radnja3_protokol = `'tcp'`

radnja3_adresa = `X'`<heksadekadski zapis adrese protokola IP>`'`

radnja3_CCF = `FALSE`

radnja3_ACF = `FALSE`

radnja3_razašiljanje = `FALSE`

radnja3_premošćivanje = `FALSE`

radnja3_nesigurni = `NULL`

radnja3_sigurni = `NULL`

aktiviranje = `NULL`

deaktiviranje = `NULL`

aktivno = `TRUE`

zadnji_put_pokrenuto = `TIMESTAMP '`<vrijeme zadnjega pokretanja>`'`

Slika 4.5. Primjer pravila 1 u konkretnom zapisu

Ovdje naredba za čitanje senzora vraća poruku u metaprotokolovu obliku, odnosno npr. 2 bajta „\0<bajt vrijednosti>“. Dalje, pravilo 2 šalje onomu tko pita jednak odgovor kao i pravilo 1. U relacijskom obliku zapisuje se ovako na Sl. 4.6.

korisnik = 'root'

identifikator = 2

vrsta = PRIMANJE

filtar = 'SUBSTRING(PL FROM 1 FOR 1) IN (X''53'', X''DE'')' (počinje li upit slovom S ili kodom za SELECT)

radnja1 = NIŠTA

modifikacija = NULL

radnja2 = NIŠTA

radnja2_upit_naredba = NULL

radnja3 = POŠALJI_PORUKU_STVORENU_IZ_NAREDBE

radnja3_upit_naredba = 'procitaj_senzor.sh'

radnja3_protokol = NULL

radnja3_adresa = NULL

radnja3_CCF = FALSE

radnja3_ACF = FALSE

radnja3_razašiljanje = FALSE

radnja3_premošćivanje = FALSE

radnja3_nesigurni = NULL

radnja3_sigurni = NULL

aktiviranje = NULL

deaktiviranje = NULL

aktivno = TRUE

zadnji_put_pokrenuto = NULL

Slika 4.6. Primjer pravila 2 u konkretnom zapisu

Postavljeni će filtar iz tijela poruke pogledati početni bajt i vidjeti je li poruka SELECT. Pravilo 3 odgovara porukama HELLO na poruke HELLO i poruke ACKNOWLEDGMENT. Ovo potonje nije uobičajena radnja, nego je samo za ilustraciju. Vidi Sl. 4.7.

korisnik = 'root'

identifikator = 3

vrsta = PRIMANJE

filtar = 'OCTET_LENGTH(PL) = 0 OR MOD(128, MOD(64, MOD(32, MOD(16, CAST(HD AS INTEGER))))) > 8' (ako nema tijela ili je postavljen bit K)

radnja1 = NIŠTA

modifikacija = NULL

radnja2 = NIŠTA

radnja2_upit_naredba = `NULL`

radnja3 = POŠALJI_NOVU_PORUKU_STVORENU_IZ_UPITA

radnja3_upit_naredba = `'SELECT X'''';'` (stvaranje prazne poruke)

radnja3_protokol = `NULL`

radnja3_adresa = `NULL`

radnja3_CCF = `NULL`

radnja3_ACF = `NULL`

radnja3_razašiljanje = `NULL`

radnja3_premošćivanje = `NULL`

radnja3_nesigurni = `NULL`

radnja3_sigurni = `NULL`

aktiviranje = `NULL`

deaktiviranje = `NULL`

aktivno = `TRUE`

zadnji_put_pokrenuto = `NULL`

Slika 4.7. Primjer pravila 3 u konkretnom zapisu

Programska potpora za naprednije čvorove kao što su to usmjernici IoT-a imat će naprednije mogućnosti za filtriranje i prosljeđivanje, kao i informacije iz baze. To su recimo rute kojima poruke idu. Tu bi moglo biti pravilo npr. na Sl. 4.8.

```
PRAVILO 4:
tip = U_PRIMANJU
FILTAR(TREBA_PROSLIJEDITI_OD(PORUKA.SRC))
FILTAR(TREBA_PROSLIJEDITI_K(PORUKA.DST))
radnja:
    IZVRŠI_SQL(„SELECT pravo_odredište FROM tablica_prosljeđivanja WHERE
identifikator_izvorišta = “ + PORUKA.SRC, REZULTAT_0);
    PROSLIJEDI_PORUKU(PORUKA, REZULTAT_0);
```

Slika 4.8. Primjer pravila 4

U relacijskom obliku pravilo glasi kao na Sl. 4.9.

korisnik = 'root'

vrsta = PRIMANJE

identifikator = 4

filtar = 'EXISTS(SELECT TRUE FROM treba_proslijediti_od WHERE adresa = SRC) AND EXISTS(SELECT TRUE FROM treba_proslijediti_k WHERE adresa = DST)'

radnja1 = NIŠTA

modifikacija = 'DST = (SELECT pravo_odredište FROM tablica_prosljeđivanja WHERE identifikator_izvorišta = SRC)'

radnja2 = NIŠTA

radnja2_upit_naredba = NULL

radnja3 = NIŠTA

radnja3_upit_naredba = NULL

radnja3_protokol = NULL

radnja3_adresa = NULL

radnja3_CCF = NULL

radnja3_ACF = NULL

radnja3_razašiljanje = NULL

radnja3_premošćivanje = NULL

radnja3_sigurni = NULL

radnja3_nesigurni = NULL

aktiviranje = NULL

deaktiviranje = NULL

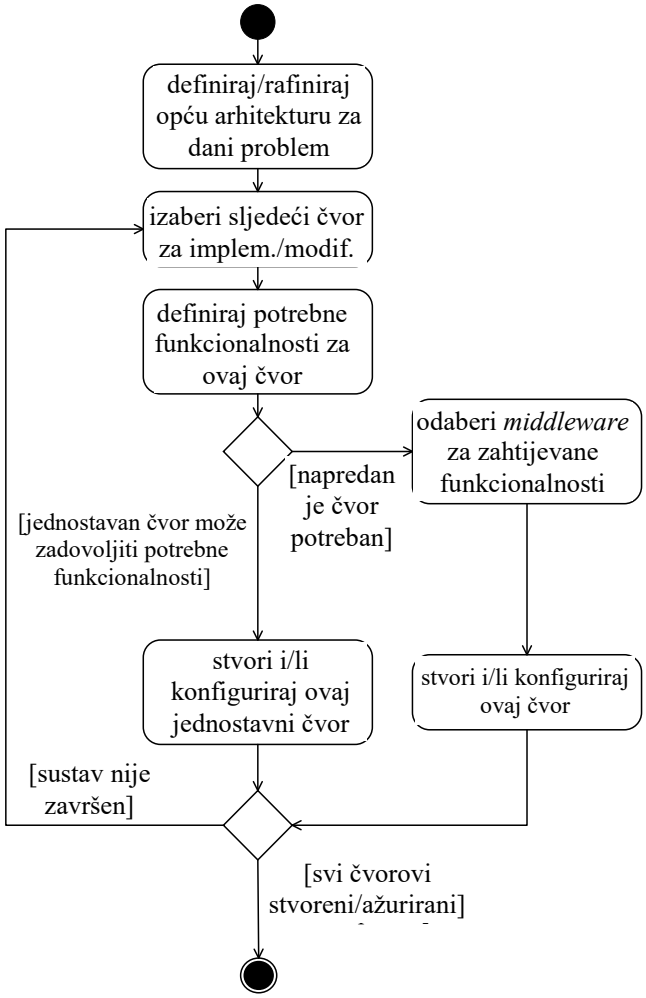
aktivno = TRUE

zadnji_put_pokrenuto = NULL

Slika 4.9. Primjer pravila 4 u konkretnom zapisu

4.6. Vodič za ostvarivanje arhitekture

Na Sl. 4.10 prikazan je dijagram aktivnosti (engl. *activity diagram*) *ujedinjenoga modelnoga jezika* (engl. Unified Modeling Language, UML) koji pokazuje kako najjednostavnije izraditi nov sustav IoT-a uporabom predložene arhitekture.



Slika 4.10. Vodič za ostvarivanje arhitekture

Prvi je korak u stvaranju nova sustava definiranje njegove arhitekture ili rafiniranje postojeće. Stvara li se nov sustav bez uzorâ, može se krenuti i od jednoga čvora koji obavlja jednu radnju. Npr., ako se ima sustav gdje se trebaju prikupljati podatci od čvorova-senzorâ i prikazivati korisniku, može se krenuti od čvora koji prikuplja podatke. Stvara li se sustav rafiniranjem drugoga sustava, kreće se od čvorova koji već jesu definirani.

Sljedeći je korak dodavanje ili nadograđivanje čvorova. Npr., ako se na početku ima samo čvor koji prikuplja podatke, može se dodavati jedan po jedan čvor čiji se podatci prikupljaju.

U dodavanju čvora treba definirati i njegovu programsku potporu. Za najjednostavnije čvorove „stvari“ mogu se upotrebljavati jednostavni programi kao što je to obična petlja. Napredniji čvorovi upotrebljavat će posebnu potporu (engl. *middleware*) kao što je ova iz

predloženoga prototipa. Ona može upotrebljavati brojna implicitna pravila ili se može konfigurirati dodatnim pravilima.

Proces dodavanja ili nadogradnje čvorova ponavlja se sve dokle se ne dobije odgovarajuća razina funkcionalnosti. Ta se razina može poslije i promijeniti, pa se čvorovi mogu poslije i oduzimati ili pojednostaviti, bez većih promjena na drugim čvorovima, u nekim situacijama i bez ikakvih; kao i kad se ide u suprotnom smjeru, „naprijed“ s dodavanjem ili nadograđivanjem.

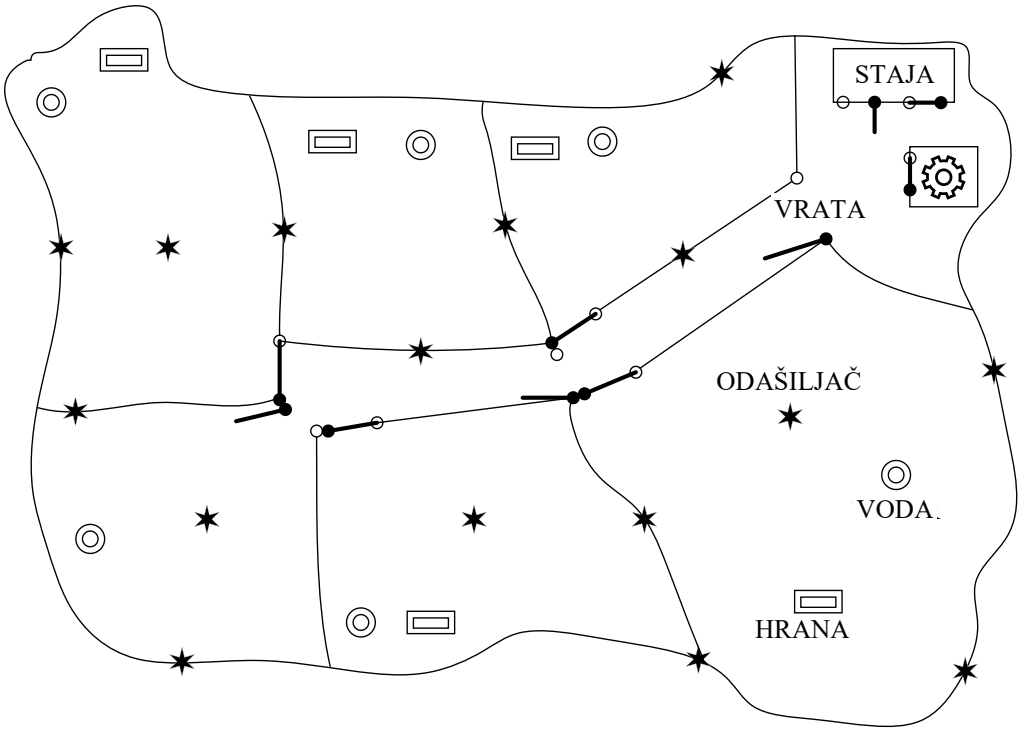
Isti se postupak može upotrebljavati i u slučaju već postojećega sustava, ako je taj sustav izgrađen u skladu s ovim modelom. U svakom slučaju, promjena jednoga čvora najčešće ne utječe na druge, zbog brojnih implicitnih pravila kao što su automatska pohrana podataka, automatski njihov dohvat, i ostale operacije koje se događaju u čvoru koji ostvaruje puni metaprotokol.

Ovaj je model u nekim crtama sličan samomu Internetu. Ondje kad se doda čvor u sustav – a to može biti i jednostavno uključivanje Interneta na mobitelu – čvor dobiva adresu protokola IP. On s vremenom uđe u usmjerne tablice usmjernikâ, ali nijedan se drugi čvor ne mora promijeniti da bi novomu čvoru izišao u susret. Na isti način ni ovdje se ne mora ništa posebno mijenjati ako su implicitna pravila dovoljna.

4.7. Pametno imanje

U ovom potpoglavlju prikazuje se primjer sustava IoT-a u svrhu prikaza nekih prednostî predložene arhitekture. Te se prednosti vide u osmišljavanju sustava, olakšanom uključivanju vrlo ograničenih čvorova, kao i upotrebljivosti i fleksibilnosti.

Primjer sustava prikazan je na Sl. 4.11. On se odnosi na pametno imanje (farmu, ranč) negdje na selu. Na imanju postoje mnoge životinje, a to mogu biti krave, ovce i sl. sustav IoT-a u ovom se primjeru sastoji od pametnih uređaja (čvorova, stvarî) koji su ugrađene u fizičke objekte. To mogu biti ogrlice koje životinje nose, pojilišta, hranilišta, vrata za životinje, samostalni senzori, glavno središte za obradbu životinja kao što je pametno muzilište itd. Postoji i poslužiteljski čvor s korisničkim aplikacijama za nadgledanje nekih dijelova.



Slika 4.11. Primjer sustava pametnoga imanja

U ovom primjeru imanje je podijeljeno na velika zemljišta gdje se životinje kreću. Onuda se mogu slobodno kretati, jesti, piti, odmarati se, spavati itd. Do zemljištâ se dolazi ograđenim prolazima, a te prolaze otvaraju i zatvaraju pametna vrata. U određene svrhe treba životinje usmjeravati u različita područja po danu i to se izvodi automatski. Pametna hranilišta i pojilišta postoje na određenim područjima ako su ondje potrebna. Na nekim objektima, npr. ogradama, postoje i samostalni senzori koji prate kuda se životinje kreću tijekom dana.

Pametna vrata spojena su žicom s vanjskim svijetom i primaju naredbe od poslužitelja. To mogu biti poruke `DATA` u obliku „`radnja='otvori';`“ ili `'zatvori'` ili `'stanje'`. Veza s vanjskim svijetom ne mora biti stalna nego se može uspostaviti nekoliko puta po satu, ako vrata nisu spojena na izvor stalnoga napajanja. Osim upravljanja vratima čvor isto tako bilježi i koje su životinje prošle kraj njega nekom kratkodometnom komunikacijom. Pametnoj je ogrlici za to dovoljno s vremena na vrijeme odaslati poruku `HELLO`. Kad ta vrata prime takvu poruku, dekodiraju identifikator životinje iz njezine adrese i pohrane ga, vremenski potpisanoga. Kad se poslije uspostavi veza s poslužiteljem, uz pomoć poslužiteljeva upita vrata će poslati te podatke preko poruka `DATA`. One će glasiti npr. „`vrijeme,"id_životinje"=`<vr1>`,`<id1>`;`<vr2>`,`<id2>`;…;`“. U ovima su potrebni navodnici da se ne bi „`ž`“ (`U+017E`) dekodiralo kao ključna riječ. Nema

li veze s poslužiteljem, vrata se mogu namjestiti na otvaranje i zatvaranje u određena doba dana, što se može konfigurirati pravilima.

Pametni čvorovi u senzorima, vratima i ostalim pametnim nepomičnim objektima-lokacijama periodično će odašiljati svoje postojanje. To će za razliku od vrlo ograničenih ogrlica činiti preko poruka `DATA` kao što su „`lokacija='<identifikator>';`“, za implicitno lociranje, tako da primatelj u ogrlici ne mora imati nikakva eksplicitna pravila za rad s porukama. Ta ogrlica pohranjuje poruke da bi ih poslije dala na vratima staje, kad ju ona pita sve podatke uz pomoć poruke „`SELECT *;`“. Nije li to dovoljno za praćenje životinja, na zemljišta se može dodati nekoliko složenijih čvorova koji imaju naprednije komunikacijske mogućnosti. Npr., nekim samostalnim senzorima ili nekim pametnim objektima može se dodati kartica SIM. Nije li ni to dovoljno, mogu postojati i „jače“ ogrlice koje imaju i *globalni pozicijski sustav* (engl. Global Positioning System, GPS), a prate obližnje životinje koje odašilju poruke `HELLO`. Ako su dakle komunikacijski naprednije mogu se spojiti na poslužitelj s vremena na vrijeme i poslati mu sve, ili čekati da ih on pita.

Pametna hranilišta i pojilišta isto otkrivaju nazočnost životinja, kad se aktiviraju nekim mehaničkim senzorom, slušajući njihova odašiljanja poruka `HELLO`. Dodatno, kad se životinja otkrije, provjeri se je li sve dobro opskrbljeno za njihov tolik broj i pusti još npr. vode ako se vidi da nje nedostaje.

Ovaj je predstavljeni sustav vrlo labavo povezan. Često ni dodavanje novoga čvora ne uzrokuje nikakve potrebne promjene na drugim čvorovima. Možda jedino u tom slučaju treba nešto promijeniti na poslužitelju; npr., kad se doda nov samostalan senzor, unese mu se lokacija na poslužitelj i to je to. Slično vrijedi i za druge čvorove ako je njihova konkretna fizička lokacija važna. Kad se čvor ukloni, nema nikakvih promjena, nigdje, pa tako ni na poslužitelju. Što se tiče samoga programiranja čvora, ono se može slobodno izvesti samo jedan put i to prije uključivanja u sustav. Pa tako se neki vrlo ograničeni čvorovi poput samostalnih senzora mogu ostvariti vrlo ograničenim mikrokontrolerima jedan put isprogramiranima. Dodavanje ili mijenjanje uloge nekomu čvoru ne zahtijeva široke promjene, a možda i nikakve. Npr. stanje hranilišta ili pojilišta mogu odašiljati kao nešto novo jednostavnim dodavanjem polja u lokacijsku poruku. Npr., to može biti proširena poruka „`lokacija,voda,baterija='w421',73,45;`“, gdje je lokaciji `'w421'` dodano stanje vode od `73` % i baterije od `45` %. Tu poruku primala bi ogrlica i

isto kao i prije implicitno pohranjivala u proširenoj tablici za taj objekt. Poslije te podatke može dati onomu komu trebaju. U tom slučaju jedina je promjena na izvorištu podataka i vjerojatno poslužitelju koji s tim podatcima radi. Ne prilagodi li se poslužitelj, onda se podatci samo pohranjuju automatski a ne obrađuju.

Svi spominjani podatci mogu se pohraniti na samo jedan poslužitelj koji može biti u lokalnoj mreži, ali i na Internetu, ili biti poslani na više njih. Npr., podatci pametnih hraništa mogu se proslijediti poslužitelju tvrtke koja se bavi opskrbom imanja hranom i sličnim. Ti podatci se zadržavaju i za gospodara imanja i on ih može raščlaniti uz pomoć sučelja za *web*. Recimo, može vidjeti gdje se troši koliko hrane pa zaključiti gdje ima dovoljno prirodne hrane i kamo poslati životinje. Čak se i više imanja istodobno istoga vlasnika može kontrolirati samo jednim poslužiteljem, ili nekolicinom njih povezanih.

Za sad se nije spominjala sigurnost u ovom sustavu jer se u praksi ne događaju veći sigurnosni problemi. Oni nastaju ako imanje graniči s nekim drugim imanjem drugoga vlasnika koji upotrebljava identičan sustav IoT-a. Tad se trebaju razlučiti vlastite životinje od tuđih, a to se može na više načina.

Dostaje li u tom slučaju samo razlika između čvorova, obično je dovoljno samo filtrirati podatke u poslužiteljima koji ih kupe. U pojilištima ili hraništima može se rukovoditi popisom životinja tekućega imanja, ili filtrom nad njihovim identifikatorima. Tako se izbjegava aktiviranje za neke tuđe životinje koje su samo uhvaćene signalom, ali se ne nalaze na samom imanju nego s druge strane ograda.

Nije li to sve skupa dovoljno, postoje naravno i druge sigurnosne mogućnosti. Sigurnost se za početak može ostvariti nižim protokolom i u slučaju komunikacije jednim skokom (engl. *single-hop*) to je i optimalno rješenje. Naime, tako se optimizira potrošnja sredstava u ograničenim čvorovima gdje se sigurnost prepušta čipovima koji se bave komunikacijom. Npr., ako je tu potreban neki ključ za komunikaciju, on se prije početka komunikacije u njima konfigurira. Postoji li u sustavu komunikacija preko više skokova, potrebna je naprednija sigurnost s kraja na kraj jer možda postoji sumnja da je neki čvor kompromitiran. U tom slučaju mogu se upotrijebiti mehanizmi metaprotokola u smislu digitalne omotnice ili digitalnoga potpisa. To bi onda zahtijevalo predrazmjenu svih potrebnih ključeva prije početka komunikacije. Na taj se način mogu osigurati poruke bez straha da će ih netko pročitati ili promijeniti, kako je objašnjeno u sigurnosnoj raščlambi u

sljedećem poglavlju, jer je to smanjivanje sigurnosti nemoguće ako su mehanizmi ispravno implementirani.

5. EVALUACIJA PREDLOŽENE ARHITEKTURE

U ovom poglavlju razmotrit će se dobre i slabe strane predložene arhitekture. Kroz usporedbu sa sličnim sustavima iz znanstvene literature, ali i s komercijalnim i besplatnim sustavima za IoT, pokazat će se i gdje predložena arhitektura ima prednosti, a gdje možda i nema.

Usporedba s literaturom već je učinjena u potpoglavlju 2.2. Ponovit će se najvažnije razlike. Članci iz literature razvijeni su samo za jednu svrhu, zahtijevaju da korisnik nauči poseban jezik, namijenjeni su samo za lokalne mreže, imaju dodatan apstrakcijski sloj za komunikaciju, imaju kruto odvojene kategorije svojih čvorova, ne podupiru dinamičku promjenu uloge ili složenosti čvora, ne omogućuju ubaciti jednostavan čvor i onda ga dotjeravati, i sl. Predložena je arhitektura zamišljena za mreže IoT-a opće namjene, upotrebljava se samo jezik SQL, porukama se izravno rukuje, mijenjanje postavaka čvora može biti vrlo jednostavno, čvoru se mogu dodavati i oduzimati funkcionalnosti, na sigurnost se misli od početka, nudi se i ušteda energije i sl.

U rješenjima iz znanstvene literature postoje neke česte značajke koje u sustavu IoT-a nisu poželjne – makar što se tiče prosječna korisnika. Te se značajke pretežno odnose na fleksibilnost primjene i jednostavnost uporabe. Naravno, ne može se reći da sva rješenja imaju iste probleme. Naime, iako su vrlo slična rješenja, opet su i vrlo različita. Ipak, ono što se može reći jest da se neki problemi pojavljuju baš u svima njima. To su, primjerice, namijenjenost samo za lokalne mreže ili kruto odvojene kategorije čvorova, što smanjuje fleksibilnost sustava. Ti svi problemi mogu biti manje ili više važni korisniku, ili može njihova kombinacija biti važna. Jesu li mu važni, ili će se primijeniti predloženi sustav ili će se opet ići s nekim sustavom za precizno određene namjene gdje će ti problemi biti manje istaknuti od drugih.

Prikaz prednostî predloženoga sustava nikako ne bi bio potpun bez usporedbe s komercijalnim rješenjima. Ta su rješenja isto navedena u poglavlju 1, a ovdje će se samo ponoviti da su to potpuni skupovi primjenskih programa temeljeni na oblaku (engl. *cloud-based complete full-stack application sets*) [110]. Oni omogućuju, između ostaloga,

napredne analize prikupljenih podataka i napredne grafičke prikaze tih analiza. Takvi sustavi već imaju mnoge korisnike za koje ti sustavi dobro rade.

Ono što se odmah uočava jest da, kao i sustavi iz literature s kojima se uspoređuje, komercijalni sustavi dijele određene osobine koje mogu biti nepoželjne korisnicima. Npr., te arhitekture imaju krute kategorije čvorova s hijerarhijskom ogranizacijom – od najnižih „stvari“ do najvišega „oblaka“. Za mnoge scenarije takvi su sustavi idealni, ali, naravno, ne i za sve. U sljedećem potpoglavlju usporedit će se potanje načini izgradnje sustava za takve tradicionalne arhitekture s načinom izgradnje za predloženu arhitekturu.

Osim jednostavnosti izgradnje sustava pitanje je i fleksibilnosti primjene. Ti komercijalni sustavi nisu izgrađeni samo za posebnu namjenu kao mnogi iz literature, ali oni još uvijek nisu prikladni za jednostavnije sustave IoT-a koji samo žele razmijeniti podatke bez instalacije više različitih programskih potpora i sučelja. Stoga se, koliko god takvi sustavi bili moćni, ovdje ne smatraju dovoljno općenitima za sve primjene.

S druge strane, ono što čini predloženu arhitekturu općenitijom jest i mogućnost kombiniranja s drugim, pa i komercijalnim rješenjima. Npr., može se imati lokalna mreža u kojoj se upotrebljava jednostavna komunikacija, a onda usmjernik sve potrebno prosljeđuje u oblak, gdje se može upotrebljavati naprednija raščlamba podataka ili napredniji grafički prikaz. To je posebno zgodno ako je potrebno automatizirati neku vrstu strojnoga učenja nad podatcima, ili recimo korisnikov nadzor čvorova u stvarnom vremenu. Naime, predloženi model dovoljno je općenit da je moguća suradnja s drugim sustavima, a strojno je učenje s druge strane nešto normalno u komercijalnim rješenjima. Potrebno je samo prilagoditi prijenos podataka iz tablica u drugi sustav i natrag. To može biti ostvareno i modulom UPM, npr., za protokol MQTT, ili vanjskim programom, i sl.

Osim komercijalnih rješenja postoje i besplatna rješenja IoT-a. Primjer je jednoga potpora Node-RED [111]. Potpora Node-RED je u biti uređivač pravila (engl. *rule editor*) koja se definiraju preko internetskoga preglednika, donekle slično nekim znanstvenim rješenjima. Uz pomoć potpore Node-RED mogu se grafički slagati kompozicije funkcija jezika Javascript što omogućuje fleksibilnost. Program koji se tako složi zove se „tijek“ (engl. *flow*) i pohranjen je u obliku zapisa JSON. On se poslije stavlja na uređaje IoT-a koji podupiru tehnologiju Node.js jezika Javascript. Potpora Node-RED podupire više različitih vrsta čvorova „stvari“, (npr. Opto, Raspberry, Siemens…). Isto tako može uspostaviti vezu

s više različitih „oblaka“, (npr. AT&T, Cisco, Nokia…). Poseban je naglasak zapravo na „rubnim“ uređajima u mreži (engl. *edge computing*) i povezivanju s oblakom.

Potpora Node-RED se dakle brine najviše oko komunikacije između čvora i poslužitelja, pa ne omogućuje puno veću fleksibilnost organizacije čvorova i povezivosti između njih kao što se predlaže u arhitekturi u ovom radu. Nadalje, potpora Node-RED se ne može staviti na čvorove ograničene sredstvima zbog jezika JavaScript i potrebne dodatne potpore za nj, za razliku od predložene arhitekture. Slično tomu komunikacija među čvorovima u sustavu nije optimizirana za jednostavne čvorove i nezahtjevne poruke, za razliku od ovoga prijedloga.

Dodatno bi se trebalo spomenuti da bi komercijalno rješenje zbog svoje naprednosti moglo i odbiti korisnike, primjerice zbog cijene ili složenoga stavljanja u pogon (engl. *deployment*).

Ne očekuje se da običan korisnik sam može složiti višerazinske sustave u predloženoj arhitekturi, kao što se to ne očekuje ni u komercijalnim rješenjima. Ipak, već izrađene sustave može mijenjati na načine na koje to ne bi mogao u njima. Npr., ako treba promijeniti ulogu čvora, u uporabi komercijalnih rješenja najčešće će trebati zamijeniti cjelokupnu programsku potporu na tom čvoru, dokle će uz uporabu predložene arhitekture dovoljno biti uključiti ili isključiti neke dijelove potpore. K tomu, izradba nekoga novoga sustava/čvora kopiranjem nekoga postojećega uz neke male izmjene jednostavnija je, jer se ne mora voditi računa o različitim programskim potporama za različite čvorove, nego samo o drugačijoj konfiguraciji jedne od njih. Na taj način, po mišljenju autora, zapravo izgradnja sustavâ postaje dostupna običnu korisniku ili makar nekima od njih.

Naprednije analize i prikazi nisu dio modela iako su jednostavnije analize i prikazi već uključeni u prototipnu implementaciju. Kad bi takve stvari u konkretnom slučaju bile potrebne, mogle bi se dodati kao „modul“ potpore u čvor u smislu dodavanja složenih pravila i mogućnosti pokretanja vanjskih programa. Kako je naglasak bio cijelo vrijeme na jednostavnijim sustavima, to se ne će dalje razrađivati.

5.1. Usporedba izgradnje i prilagodbe sustava

Sustavi u kojima postoji očita prednost predloženoga nad postojećim rješenjima oni su sustavi gdje je važna neprekinuta evolucija i prilagodba. Kao i u skoro svakom sustavu,

očekuje se da će korisnici mijenjati svoje zahtjeve u skladu s naučenim mogućnostima sustava, izravno ili dijeljenjem iskustava s drugim korisnicima.

Ne postoji li već oblikovana zamisao kako treba izgledati novi sustav, predložena se arhitektura može upotrijebiti za izgradnju sustava na sljedeći način. Prva točka u stvaranju nova sustava bio bi čvor-usmjernik. Taj bi čvor za početak prikupljao sve podatke iz lokalne mreže – sve što mu se pošalje određenim podatkovnim protokolom i to pohranjuje u bazu.

Sljedeći bi koraci bili dodavanje čvorova u lokalnoj mreži, jedan po jedan. To za početak mogu biti samo čvorovi koji s vremena na vrijeme razašalju 1 bajt, uvećan za zaglavlja podatkovnoga protokola, možda i fizičkoga sloja, tomu usmjerniku. Ništa drugo za početnu verifikaciju sustava nije potrebno. Usmjernik za početak samo pohranjuje podatke, ali može i uz pomoć sustava pravila obaviti i neku radnju, kao što je javljanje nekomu drugomu lokalnomu čvoru da se je nešto dogodilo, primjerice, nekomu aktuatoru.

Doda li se u nekom trenu kakav napredniji čvor koji nudi složenu obradbu podataka ili složeni grafički prikaz, vjerojatno negdje u oblaku – a možda to bude i neko od komercijalnih rješenja – usmjernik se nadogradi pravilima koja prosljeđuju poruke i to možda samo neke tomu čvoru, umjesto obradbe, čim se dio trenutačne uloge usmjernika prebaci na oblak. U tom slučaju stvari IoT-a ne moraju biti svijesne mreže onkraj usmjernika. One mogu nastaviti slati svoje podatke, odgovarati na poruke usmjernika ili obavljati druge „lokalne“ operacije, bez drugih znanja.

Dodavanjem nekoga važnijega čvora za mrežu, kao što je nov usmjernik ili čvor koji obrađuje podatke i donosi odluke, ne bi se trebali znatno promijeniti postojeći povezani čvorovi, tj. promjene koje na njima treba učiniti zbog toga dodanoga čvora ne bi trebale biti složene. Naime, takvo mijenjanje čvora može biti, npr., da čvor sad može slati više podataka ili zahtijevati podatke. Svejedno to mijenjanje ne bi trebalo zahtijevati da se neki drugi lokalni čvor mijenja, nego samo možda dodati neka pravila u usmjernik.

Jednom izgrađena početna lokalna mreža može se dalje nadograđivati. Recimo, treba štedjeti energiju pa treba skratiti poruke, ili se mreža treba izvući iz trenutačnoga nadziranoga okružja pa se treba dodati sigurnost, ili se trebaju dodati pravila za filtriranje neželjenih poruka, ili se trebaju dodati novi čvorovi, ili se trebaju dodati različiti korisnici na usmjernik. Sve je to poduprto predloženom arhitekturom; lokalna se mreža može dalje

razvijati dodavanjem ili uklanjanjem čvorova, mijenjanjem konfiguracije ili povezanosti čvorova, stvaranjem podmreža, mijenjanjem uloga čvorova, i sl.

Sljedeći bi korak bio povezanost s daljinskim čvorovima – čvorovima u nekoj drugoj lokalnoj mreži IoT-a. Ako je lokalna mreža dio većega sustava, ta će povezanost svakako biti potrebna. Ta se povezanost može ostvariti i samo promjenom jednoga čvora preko pravila koja s njom rade. Primjerice, u usmjerniku se dodaju pravila za posredničku sigurnost ili u nekim slučajima ni to ne mora biti potrebno nego se može upotrijebiti automatsko usmjeravanje kao što je prikazano u Primjeru 3 i 4 u potpoglavlju 4.3 i 4.4. U slučaju potrebe velikih promjena u sustavu očekuju se i veće promjene u čvorovima.

Može se dakle reći da povezanost s drugom mrežom zahtijeva minimalne promjene u lokalnoj mreži. Dodatno, razvoj lokalne mreže može biti neprekinut i nesmetan promjenama u povezanosti s drugim mrežama, kao i promjenama u tim mrežama.

Ako bi se za ovakve sustave, u kojima je potrebno stalno razvijanje, upotrijebile tradicionalne arhitekture, mogući su sljedeći problemi koji će otežati razvoj. Prvi jest istodobna uporaba većega broja tehnologija, ako bi one uopće omogućivale razvoj odgovarajuće mrežne povezivosti, i ako bi složenost tih tehnologija bila prikladna za sredstvima ograničene uređaje IoT-a. Drugi problem jest mogu li se novi zahtjevi uopće riješiti početnom arhitekturom. Nadalje, mogu li se različiti modeli podataka na različitim razinama ujediniti na zadovoljavajuć način. Onda, u tradicionalnim će arhitekturama najvjerojatnije biti potrebno zamijeniti cijelu programsku potporu na čvoru kad se promijeni njegova uloga ili čak i manji dio njegova rada.

Neka se uzme s druge strane da u sustavu nije potreban neprekinut razvoj, a sustav odgovara nekoj tradicionalnoj shemi pa recimo, samo treba jedan usmjernik prikupiti podatke od više čvorova i prikazati ih korisniku. Tad bi bilo bolje uzeti takvo ostvarivanje. Npr., ako se trebaju prikupljati podatci za atletske treninge, bolje je uzeti u potpoglavlju 3.1 opisan sustav koji je već pripravljen za to. Ne bi bilo razumno napadati problem općenitom tehnologijom ako se sigurno zna da potrebama precizno odgovara određeni sustav i da se zahtjevi poslije ne će mijenjati.

Ovom se predloženom arhitekturom nesumnjivo podupire današnji naglasak na brzom stvaranju i prototipiranju mreža IoT-a. To se može zvati i prodirućim računarstvom (engl. *pervasive computing*) [112]. To nije stvaranje složenih sustava ni iz čega, nego počinjanje od najjednostavnijih, najmanjih, sustava po trenutačnim potrebama, a onda optimiziranje

ili proširivanje ili usložnjavanje ili osiguravanje sustava malo-pomalo. Ako i ne bude potrebno ništa od tih nadogradnja, brzo stvaranje najjednostavnijih sustava vrlo je jaka strana predložene arhitekture.

Dodavanje čvora u postojeći sustav obično zahtijeva veće promjene u arhitekturama IoT-a, dokle u predloženoj zbog implicitnih pravila najčešće ne će biti potrebna nikakva promjena. Naime, za dodani čvor u početku mogu djelovati samo takva implicitna pravila. Poslije, ako to nije dovoljno, mogu se dodati i eksplicitna pravila.

Osim u dodavanju čvora postoje i velike razlike u tom što su u tradicionalnim arhitekturama sve razine kruto odvojene i komuniciraju različitim sučeljima, a obično i različitim protokolima. Na primjeru složaja „stvar-usmjernik-poslužitelj-korisnik“ očekuju se 4 različite programske potpore i 3 sučelja među njima, kao u komercijalnim uslugama. Možda, čak, i svaka potpora s drugačijim podatkovnim modelima. U predloženoj arhitekturi nema nikakva kruta odvajanja i čvor može imati više od jedne uloge. Recimo, arhitektura se može „spljoštiti“ tako da „usmjeravanje“ i „posluživanje“ budu u jednom čvoru, ili „razliti“ tako da usmjernik preuzme možda i samo dio uloge poslužitelja.

S druge strane u predloženoj se arhitekturi očekuje samo jedno sučelje metaprotokola i moguće samo jedna programska potpora. O nižim protokolima komunikacija uopće ne ovisi, tj. ortogonalna je na njih i oni se mogu „u letu“ zamijeniti, osim kad treba skratiti poruke pa moduli rukuju izvlačenjem i ubacivanjem podataka iz koda.

Važno je zato naglasiti da je broj odvojenih dediciranih čvorova, programskih potpora i komunikacijskih sučelja puno manji nego inače, a to se donekle može i vidjeti iz Tablice 5.I gdje se razmatraju potrebne radnje u dodavanju nova čvora u sustav. Iako je u tablici naglasak na drugim stvarima, vidljivo je isto tako i da se ne radi s kategorijama nego s ulogama i da se zahvaljujući općenitosti potpore i sučelja, uz, naravno, implicitna pravila, stvari znatno pojednostavljuju.

Tablica 5.I. Operacije povezane s dodatkom jednostavna čvora u postojeći sustav IoT-a

Arhitektura	**Razina stvari**	**Razina usmjernika**	**Razina poslužitelja**	**Razina korisnika**
„Obična“ 4-slojna arhitektura	-operacija stvari -konfiguracija za komunikaciju s daljinskim čvorom	*-opcionalna konfiguracija usmjernika za obradbu i prosljeđivanje poruka*	-ubacivanje parametara dodanoga čvora u poslužiteljev sustav *-opcionalna logika više razine*	-ubacivanje postavaka za komunikaciju i obradbu poruka za dodani čvor na korisnikovu aplikaciju
Predložena arhitektura	-operacija čvora s ulogom stvari *-opcionalna konfiguracija za komunikaciju s daljinskim čvorom ako implicitne postavke nisu dovoljne*	*-opcionalna konfiguracija čvora s ulogom usmjernika za obradbu i prosljeđivanje poruka ako implicitne postavke nisu dovoljne*	*-opcionalna konfiguracija dodanoga čvora u čvoru s ulogom poslužitelja ako implicitne postavke nisu dovoljne* *-opcionalna logika više razine*	*-opcionalno ubacivanje postavaka dodanoga čvora za komunikaciju i obradbu poruka ako implicitne postavke nisu dovoljne*

Mnoge operacije koje će u tradicionalnoj 4-razinskoj (4-slojnoj) arhitekturi biti neophodne u predloženoj vrlo često ne će nego će postojeća pravila (implicitna i eksplicitna) biti dovoljna.

5.2. Komunikacijske prednosti

Da bi se pokazale prednosti i nedostatci u komunikaciji između metaprotokola i drugih rješenja, usporedit će se koliko treba komunikacije za prijenos podatka određene veličine kojim protokolom. Za usporedbu se za početak uzima protokol MQTT, a slična rasprava vrijedi poslije i za protokol CoAP i slične popularne protokole IoT-a. Pogledat će se koliko ima suvišnih bajtova (engl. *overhead*) u njihovoj komunikaciji. Može se gledati i broj razmijenjenih poruka, što se usput spominje.

U ovom se primjeru gleda najviše koliko je posla s vremena na vrijeme poslati jednu poruku iz jednoga čvora u drugi. Prvi čvor može biti „stvar“ a drugi „usmjernik“ ili „poslužitelj“.

Prije same kvantitativnosti treba razmotriti i to koliko je problematično uopće upogoniti (engl. *deploy*) protokole kao što je protokol MQTT, ili slične. Naime, protokol MQTT,

CoAP, XMPP i ostali – primjenski su protokoli (4. sloj modela TCP/IP) i ispod njih treba prvo ostvariti protokol TCP, eventualno protokol UDP u nekim slučajima, protokol IP, i još protokol podatkovne razine kao Ethernet. Konkretno se za protokol MQTT u standardu pretpostavlja spojno usmjeren pouzdan (engl. *connection-oriented reliable*) protokol, a to bi skoro uvijek bio protokol TCP. Naime, iako i podatkovni protokoli mogu imati uspostavljene veze, pouzdanost nije dio njihove zadaće, kao ni dio zadaće mrežnih protokola. U svakom slučaju, u ovakvu sustavu, tj. lokalnoj bežičnoj mreži, prijenosni i mrežni sloj stvaraju puno veće zahtjeve na sustav, ne spominjući i sam primjenski protokol. S druge strane, metaprotokol se može prenositi i podatkovnim protokolom i ne mora ni imati vlastiti sloj. Tad se ispuštaju sva polja poruke osim tijela jer se drugo može izvući iz podatkovnoga protokola.

Dodatno, neki se dijelovi metaprotokola ne moraju ni ostvariti u čvoru, ali usprkos i izostavljanju cijelih slojeva, ovdje se ne će razmatrati niži protokol. Tako da nema potrebe razmatrati dodatno je li npr. jednostavnije upotrijebiti čisto *sučelje domaćinova upravljača* (engl. Host Controller Interface, HCI) za slanje 1 bajta protokolom BLE ili nadsloj iz samoga standarda kao što je *prijenos općenita tipa atributa* (engl. General Attribute Type Transport, GATT), ili pravdati da je u metaprotokol uključeno dovoljno mehanizama mrežnoga i prijenosnoga sloja da bi usporedba uopće imala smisla. Pretpostavlja se da se i metaprotokol i konkretno protokol MQTT šalju istim nižim protokolom: recimo protokolom TCP, iako nije važno je li to on ili nije – trošenje komunikacije na npr. rukovanja ne će se gledati.

Za početak komunikacije protokolom MQTT klijent poslužitelju šalje poruku `CONNECT` za uspostavu veze protokola MQTT. Za odgovor na poruku `CONNECT` mora se primiti poruka `CONNACK` za potvrdu da je veza uspostavljena. Za slanje samih podataka postoji poruka `PUBLISH`, gdje se poruka poslije može proslijediti pretplaćenim čvorovima. Ne želi li se veza održavati – recimo, ako se rijetko šalju poruke – može se ona i raskinuti porukom `DISCONNECT`. Žele li se često slati podatci, s vremena na vrijeme šalju se poruke `PUBLISH`. Slijedi brojenje bajtova u porukama za slanje podataka, po standardu protokola MQTT. Uzimaju se najmanje smislene vrijednosti – tj., poruke jednostavno ne će raditi ako se ispuste neka polja pa se sva takva uzimaju u obzir.

Poruka `CONNECT` ima najmanje 31 bajt, ne računajući, kako je rečeno, više nižih slojeva. Taj 31 bajt otpada konkretno na:

- 2 bajta za polje `Fixed Header` (hrv. *fiksno zaglavlje*) koje postoji u svim porukama;
- 11 bajtova (najmanje) za polje `Variable Header` (hrv. *varijabilno zaglavlje*) koje postoji u svim porukama;
- 18 bajtova za polje `ClientID` (hrv. *klijentov ID*) gdje se za identifikaciju pošiljatelja kao identifikator šalje identifikator EUI-64 u obliku „`UTF-8 Encoded String`“ (hrv. *kodirani znakovni niz UTF-a 8*).

Poruka `CONNACK`, nadalje, ima najmanje 5 bajtova na sloju protokola MQTT. Tih 5 bajtova otpada konkretno na:

- 2 bajta za polje `Fixed Header`;
- 3 bajta najmanje za polje `Variable Header`.

Poruka `PUBLISH` ima najmanje 24 bajta da bi se dobila jednaka izražajnost kao i u metaprotokolu. Ta 24 bajta otpadaju konkretno na:

- 2 bajta za polje `Fixed Header`;
- 21 bajt za minimalno polje `Variable Header` koji uključuje polje `Topic Name` (hrv. *naziv teme*) gdje je naziv stupca „<EUI>`.d`“ kodiran u obliku „`UTF-8 Encoded String`“;
- 1 bajt (najmanje) za polje `Payload` koje nosi podatke.

Poruka `DISCONNECT` ima najmanje 4 bajta. Ta 4 bajta otpadaju na:

- 2 bajta za polje `Fixed Header`;
- 2 bajta za minimalno polje `Variable Header`.

Ukratko, za sve ove poruke, a za slanje 1 bajta, treba za protokol MQTT najmanje 31 + 5 + 23 + 4 + taj jedan korisni bajt. Samo poruka `PUBLISH` ima 23 suvišna bajta. Ostale su poruke tu u slučaju da se veza želi cijelo vrijeme održavati. Najveći problem poruke `PUBLISH` jest zapravo slanje identifikatora podatka koji se treba kodirati npr. „`abababababababab.d`“. Moglo bi se postaviti pitanje zašto se ne stavi kraći identifikator. Činjenica jest da bi se mogao, ali onda ne bi bio sigurno jedinstven, a činjenica jest i da se u metaprotokolu on implicitno šalje adresom pa to ne bi bila jednaka izražajnost u usporedbi s njim.

Što se tiče protokola XMPP, klijent prvo mora uspostaviti vezu protokola TCP s poslužiteljem. U standardu XMPP-a je ostavljena, ali ne i istražena, mogućnost, drugoga nižega protokola, a pristupnika ima. Poslije uspostave klijent prvo šalje sljedeću „stancu“ (engl. *stanza*, tako se u standardu zovu odsječci *proširenoga doradnoga jezika* (engl. eXtended Markup Language, XML)):

```
<stream:stream to='a.b.c' version='1.0' xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'>
```

pod pretpostavkom da je simbolička adresa poslužitelja `a.b.c` (dulja može biti u obliku `ime.organizacija.domena`), izvorište je polje `from` ispušteno kao i polje `xml:lang`. Ukupno to je 110 B. U stvarnom primjeru adresa bi bila dulja. Poslužitelj odgovara „stancom“:

```
       <stream:stream from='a.b.c' id='1' version='1.0'
             xml:lang='en' xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'>
```

pod pretpostavkom da je vrijednost `'1'` dovoljna za identifikaciju veze, što daje ukupno 133 B. U stvarnom svijetu identifikator bi bio dulji. Poslužitelj dalje šalje svojstva toka:

```
                <stream:features> <mechanisms
          xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
<mechanism>PLAIN</mechanism> </mechanisms> </stream:features>
```

pod pretpostavkom da postoji samo jedno svojstvo, a to je popis mehanizama autentifikacije, i to samo jedan mehanizam na tom popisu. Ukupno to je 133 B. U stvarnom svijetu upotrebljavalo bi se više složenijih mehanizama. Klijent odgovara pokušajem autentifikacije:

```
       <auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
                   mechanism="PLAIN">1</auth>
```

pod pretpostavkom da je broj `1` dovoljan za autentifikaciju. Ukupno to je 73 B. U stvarnom svijetu autentifikacija bi bila dulja i uključivala dodatne poruke. Poslužitelj odgovara uspjehom:

```
     <success xmlns='urn:ietf:params:xml:ns-xmpp-sasl'/>
```

pod pretpostavkom da je poruka uspjeha prazna. Ukupno to je 51 B. U stvarnom svijetu poruka ne bi bila prazna. Klijent otvara nov tok isto kao u prvoj stanci (još 110 B). Poslužitelj odobrava otvaranje isto kao u drugoj stanci ali vrijednost `id='2'` a ne broj `1` (još 133 B). Onda poslužitelj mora potvrditi mogućnost spajanja na sredstvo:

```
<stream:features> <bind xmlns='urn:ietf:params:xml:ns:xmpp-
                    bind'/> </features>
```

pod pretpostavkom da nema drugih mogućnosti toka. Ukupno to je 78 B. U stvarnom svijetu bilo bi još mogućnosti. Klijent se spaja na sredstvo:

```
<iq id='1' type='set'> <bind
xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
  <resource>d</resource> </bind> </iq>
```

pod pretpostavkom da je `'1'` dovoljan za identifikaciju zahtjeva. Ukupno to je 107 B. U stvarnom svijetu identifikator bi bio dulji. Poslužitelj potvrđuje:

```
<iq id='1' type='result'> <bind
xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
<jid>abababababababab@a.b.c/d</jid> </bind> </iq>
```

pod pretpostavkom da poslužitelj zna da je identifikator klijenta `abababababababab`. Ukupno to je 123 B. U stvarnom svijetu identifikator bi bio kraći. Klijent počinje slati poruke:

```
<message type='chat'> <body>korisni podatci</body> </message>
```

pod pretpostavkom da se ispuštaju i izvorište `from` i identifikator `id` i odredište `to` i jezik `xml:lang`. Ukupno to je 46 B bez korisnih podataka. Podatci moraju biti u obliku UTF-8. Kad klijent pošalje sve i želi završiti pošalje poruku `</stream:stream>` od 16 B. Tad mu i poslužitelj odgovara poruku `</stream:stream>` od 16 B.

Što se tiče protokola CoAP, klijent prvo šalje sljedeći datagram:

`Ver=0b01` (Version = 1)

`T=0b00` (Type = Non-confirmable)

`TKL=0b0000` (Token Length = 0)

`Code=0b0000010` (kod 0.02, metoda PUT)

`Message ID=0x01`

(polje Token prazno)

`Option Delta=0b1011` (Option = Uri-Path)

`Option Length=0b1101` (Length = O.L.(extended)+13)

(polje Option Delta (extended) prazno)

`Option Length (extended)=0x05` (5 + 13 = 18)

`Option Value=`<oblik standarda Net-Unicode za "abababababababab.d"i sl.> (18 B)

`Payload Marker=0xFF`

korisni podatci

Ukupno 23 B + korisni podatci. Poslužitelj odgovara:

`Ver=0b01` (Version = 1)

`T=0b00` (Type = Non-confirmable)

`TKL=0b000` (Token Length = 0)

`Code=0b01000100` (kod 2.04, odgovor Changed)

`Message ID=0x01`

(polje Token prazno)

(polja Options prazna)

(bez polja Payload Marker)

(polje Payload prazno)

Ukupno 2 B.

Treba reći nešto i o poljima u metaprotokolu. Kako se vrlo često mogu izostaviti neka (ili sva) polja osim tijela poruke, onda cijela poruka, u tom minimalnom obliku, jednostavno glasi „`\0`<tijelo>“ ako je tijelo dulje od 1 bajta, odnosno „<tijelo>“ ako nije. To nije naravno uvijek moguće, pa se daje i usporedba s „maksimalnim“ oblikom poruke. Kad se upotrebljavaju sva polja (zaglavlje 1 B, identifikator 1 B, duljina 2 B, odredište 8 B, izvorište 8 B, zaštita 4 B – ovo zadnje rijetko), to je 20 B „viška“.

Cjelokupna je usporedba prikazana u Tablici 5.II. Vidi se da je metaprotokol najbolji za kratke poruke. Šalju li se dulje poruke, i dalje je metaprotokol optimiziran da se ne šalje „sve i svašta“ nego samo ono što treba da bi se one prenijele i razumjele pa i tu ima prednostî.

Tablica 5.II. Usporedba jednostavne komunikacije različitih protokola u bajtima

Tijelo/protokol	**1 B rijetko**	**20 B rijetko**	**100 B rijetko**	**100 B često**
MQTT	64	83	163	123
XMPP	1130	1149	1229	146
CoAP	26	45	125	125
Metaprotokol s najmanjim mogućim zaglavljem	1	21	101	101
Metaprotokol s punim zaglavljem	21	40	120	120

Treba reći i da se ovdje ne uspoređuje ni obličje zapisa podataka. Recimo, protokolom MQTT se često šalju podatci zapisa JSON, a metaprotokolom podatci jezika SQL. Svaki od njih ima svoj način zapisa, a i različite mogućnosti tipova ili organizacije podataka, tako da se nije ulazilo u to. Ipak, u metaprotokolu bez zaglavljâ ne može se ne vidjeti da se 1 bajt može poslati „samo tako“, a ne s još jednim bajtom zaglavlja. To se napominje i jer zbog toga tablica tu izgleda asimetrično.

U tablici još treba i objasniti što je to „rijetko“ i „često“ slanje, iako je to možda i očito. Dakle, misli se na to da, ako se poruke šalju rijetko, onda se mora svaki put spojiti na poslužitelj i odspojiti posebnim porukama. Šalju li se često, poruke se npr. `CONNECT`, `CONNACK` i `DISCONNECT` zanemaruju. Dodatno, kad bi se gledali konkretni niži protokoli, postojalo bi još nižih poruka ako se upotrebljava recimo protokol TCP, jer se veza protokola TCP mora i uspostaviti i održavati i raskinuti.

5.3. Implementacijske bilješke

Prototip implementacije nalazi se na mjestu za *web* GitHub [113]. Za implementaciju su se upotrebljavali programski jezici C++ (C++14) i PHP (PHP7.4). Upotrebljavao se je i skriptni jezik za ljusku `bash`. Jezik C++ služi za glavni program i „vanjske“ funkcije iz modela čvora, jezik PHP za konfiguraciju za *web*, a jezik `bash` za ispitne scenarije, kao i za skripte za instalaciju sustava, deinstalaciju i sl. Dodatno, upotrijebio se je i programski jezik Kotlin za izgradnju aplikacije za OS Android za ispitivanje funkcionalnostî protokola BLE.

Za bazu se je upotrijebila baza PostgreSQL (inačica 12) jer je to besplatna baza koja najviše poštuje standard jezika SQL [114], a za posluživanje za *web* poslužitelj Apache Web Server (inačica 2.4) jer je to najpopularniji besplatni poslužitelj [115]. Postoje i druge ovisnosti, više ili manje očite, kao što je kompilator `g++` za prevođenje datoteka izvornoga koda. Prototip je ostvaren i ispitan na OS-u Linux, distribuciji ArchLinux [116], ali bi trebao raditi i na kojem bilo drugom sustavu temeljenom na OS-u Linux. Na distribuciji ArchLinux se skriptom priloženom u repozitoriju [117] lako mogu instalirati sve ovisnosti da bi sustav mogao raditi, dokle se za druge OS-e Linux moraju naći analogni paketi. Npr., povezivanje jezika PHP i poslužitelja Apache na distribuciji ArchLinux je paket „`php-apache`“, ali na OS-u Ubuntu, odnosno OS-u Debian, i na OS-u Debian temeljenim sustavima, to je paket „`libapache2-mod-php`“. Distribucija ArchLinux nije temeljena na OS-u Debian i nema ni njegov upravitelj paketa `apt-get`, tako da već na distribuciji Ubuntu instaliranje ovisnostî zahtijeva ručnu intervenciju.

Glavni program ostvaren je višedretveno. Postoji glavna dretva za obradbu poruka te zasebna dretva za slanje i zasebna dretva za primanje, obje za svaki modul UPM. Isto tako postoji i drugi proces u kojem baza izvodi vanjske funkcije. To će se u budućnosti razdvojiti, jer njihove međuovisnosti nisu nepremostive.

Komunikacija između dijelova programa i između programa i vanjskoga svijeta ostvarena je redovima poruka. Svaka dretva za slanje ima svoj red poruka, odakle čita poruke za slanje, a dretva za primanje piše poruke u red poruka glavne dretve. Ukupno ima $2 \cdot N+1$ dretava i $N+1$ redova poruka, gdje je N broj modulâ UPM. Pri tom se zanemaruju vanjske funkcije u drugom procesu.

U glavnu petlju za primanje poruka stižu i radnje od okidačâ i ručne konfiguracije za *web*, tako da se ne dogodi neka promjena baze usred obradbe poruke. Iako se pretpostavlja da je obradba kratka, ipak se očekuju mnoge poruke. Glavna dretva koja čita iz glavnoga reda čini samo jednu stvar u jednom trenutku, obrađuje jednu poruku za slanje ili primanje, ili izvodi naredbe na okidač, i slično. U budućnosti se tu očekuje dodatan paralelizam, ali za sad se ostaje na ovom, da se ne bi trebale zaključavati tablice dokle se nešto radi. Naime, taj dio nije ni približno jednolik po bazama.

Skica prototipa čvora odgovara prikazanomu na Sl. 3.5 u potpoglavlju 3.7. Radi jednostavnosti tom slikom nisu prikazane dretve i redovi poruka. Ostvareni su moduli UPM prikazani na njoj, a strjelice na njoj samo govore o povezanosti, ne i o smjeru.

Primjerice, vanjske funkcije svoje rezultate vraćaju u glavni program, iako se pozivaju preko baze, i komunikacija je tu jednosmjerna iako tako nije prikazano tom slikom.

Opisani su mehanizmi metaprotokola i ponašanje čvorova skoro u potpunosti ostvareni u ovoj implementaciji. Nekoliko nezavršenih stvari jesu razlike u nekim scenarijima prikazanima u ovom radu i ostvarivanje sigurnosti kroz protokol BLE i protokol 154. Opis scenarija kako su trenutačno ostvareni može se naći na repozitoriju. Problem sigurnosti jest kako ju ispitati s obzirom na to da za te podatkovne protokole još nije nađeno dobro rješenje njihove simulacije. O ostvarenim mogućnostima može se pročitati u repozitoriju. Svakako na samom početku treba pročitati i datoteku `README` [118], gdje je sve opisano. Konkretan kod jezika C++ nalazi se samo u datoteki `IoT.cpp`.

Scenariji su napisani u ljuski `bash` i simulirani uz pomoć ugrađene „*containerske*“ potpore zvane `systemd-nspawn` [119]. Alternativno se je mogao uzeti neki mrežni simulator. Ipak se na njem ne bi mogao simulirati rad glavnoga programa, zanemarujući „programerske“ simulatore kao što je to simulator OMNET++. Scenariji se nalaze u mapi `scenarios`. U mapi `http` je kod jezika PHP. U početnoj su mapi još dodatne skripte i datoteke te opis nekih drugih scenarija koji ovdje nisu prikazani. Osim tih mapa, može se vidjeti još i pomoćna mapa `function_keys` za olakšavanje rada na distribuciji ArchLinux, pomoćna mapa `system_updates` za olakšavanje ažuriranja toga OS-a, aplikacija za OS Android u datoteki `.zip`, standard jezika SQL po kojem se je radilo, u datoteki `.pdf`, početni kriptografski ključevi koje treba zamijeniti novima prije ozbiljnijega rada sustava itd.

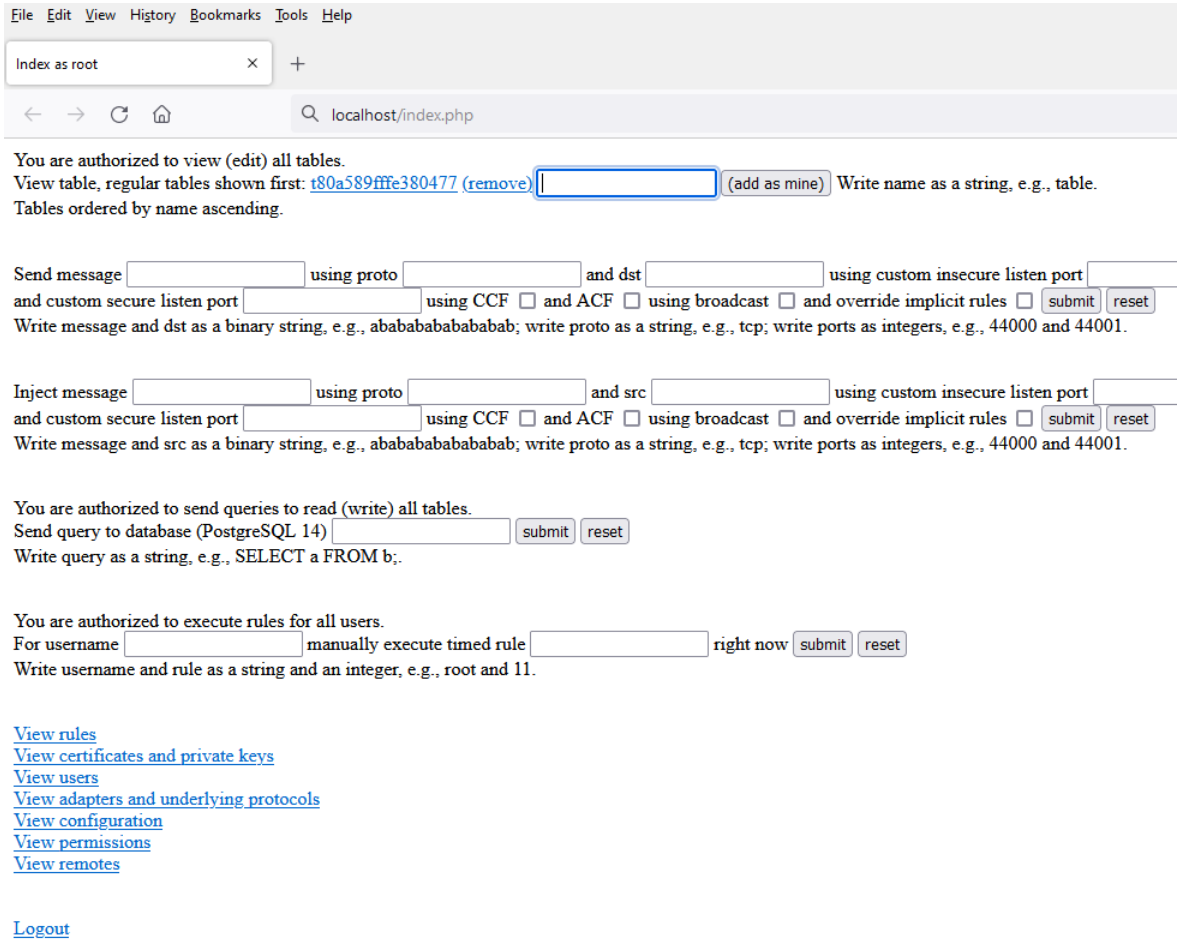
Instalacija, deinstalacija i slične operacije „zamotane“ su u te dodatne skripte tako da se sustav lako instalira, konfigurira i sl. Instaliraju se posebno i programske ovisnosti, jer bez npr. baze i poslužitelja za *web* sustav ne će raditi u punom opsegu.

Trenutačno su aktualni jezici C++20/23 i PHP8.1/8.2, ali to ne utječe na program. To je zanimljivo za korisnika ako poželi ručno dograđivati prototip, za što treba instalirati još ovisnosti. U nastavku, za ilustraciju mogućnosti, slijedi prikaz sučelja u jeziku PHP.

Na Sl. 3.6 u poglavlju 3.8 vidjela se je stranica za uređivanje pravilâ ali je ona ondje dorađena u uređivaču slika jer ne stane na tu stranicu. Na Sl. 3.7 u poglavlju 3.9 vidjela se je stranica za uređivanje konfiguracije, a na Sl. 5.1 vidi se početna stranica sučelja za *web*; opisi svega što se može raditi bili su prije za Sl. 3.6 i Sl. 3.7 i 3.8 i 3.9. Kako su u tim

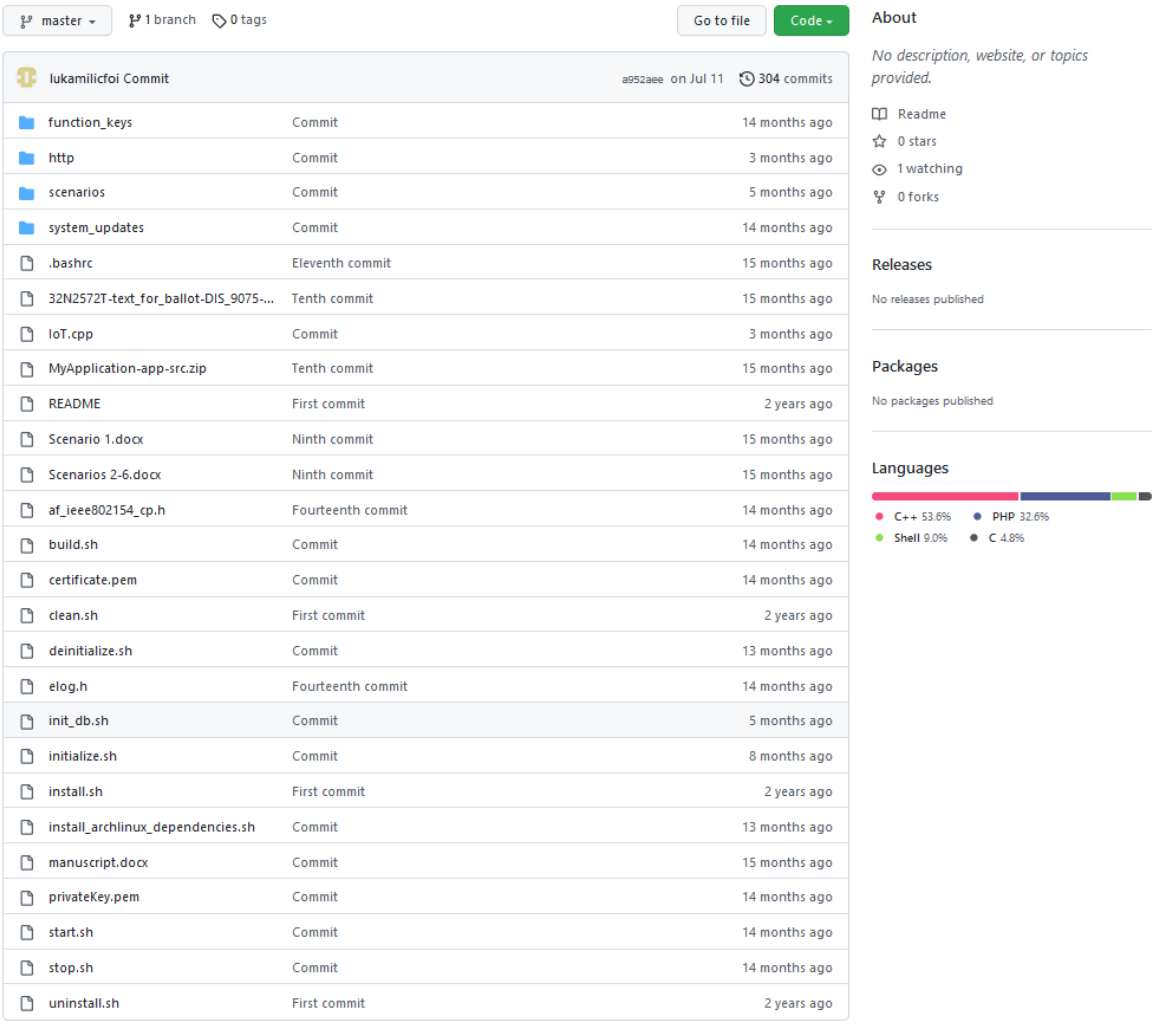
potpoglavljima potanko prikazane sve mogućnosti ovoga sučelja, ovdje se one više ne će ponavljati.

Prigodom ostvarivanja stranica pazilo se je i na moguću injekciju *hipertekstnoga doradnoga jezika* (engl. HyperText Markup Language, HTML), injekciju *jednolikoga sredstvenoga lokatora* (engl. Uniform Resource Locator, URL) te injekciju jezika SQL.



Slika 5.1. Početna stranica sučelja za *web*

Na Sl. 5.2 vidi se repozitorij datoteka na mjestu za *web* GitHub i do sad navođene datoteke. Repozitorij se može odanle skinuti naredbom *git clone* (hrv. kloniraj git) ili ručno i potom mijenjati po *GNU-ovoj općoj javnoj licenciji, inačica 2* (engl. GNU General Public License, Version 2, GPLv2) [120]. Za budući se rad ostavlja i dotjerati kod, jer je u jeziku C++ za sad samo jedna preduga datoteka.



Slika 5.2. Repozitorij datoteka na mjestu GitHub

Za ručno dotjerivanje koda se preporučuje program Eclipse [121], s kojim se je radilo, jer se u njem mogu razmjerno jednostavno uklanjati pogrješke u vanjskim knjižnicama, kojih ovdje ima nekoliko kao što su knjižnica `libpq`, OpenSSL [122], `libc++`. Za bazu se preporučuje baza PostgreSQL, s kojim za sad prototip jedino može raditi. Baza PostgreSQL ima nekoliko neostvarenih sitnica iz standarda jezika SQL pa je bolje upotrijebiti što noviju inačicu. Naime, taj se broj sitnica stalno smanjuje u novijim inačicama.

5.4. Sigurnosna analiza

Sigurnost se u predloženoj arhitekturi može ostvariti na više načina, odnosno: nižim protokolima, samim metaprotokolom, nekim drugim načinom ili kombinacijom njih više. Tri su najvažnija slučaja koja će se analizirati:

1. sigurnost nije ostvarena nikakvim protokolom,
2. sigurnost je ostvarena nekim nižim protokolom, te
3. sigurnost je ostvarena metaprotokolom.

U prvom slučaju najvjerojatnije se slučaj odnosi na sustav s ograničenim sredstvima. U takvu je sustavu moguć kakav bilo napad, npr. ubacivanjem zlonamjerna čvora u mrežu. Nije li sigurnost ostvarena fizičkom izoliranošću, sve moguće zlorabe sigurnosti mogu se ostvariti. Tako čvor-napadač može bez preprjeka i prisluškivati poruke i mijenjati ih i lažno se predstaviti i odašiljati stare poruke itd.

Kad se upotrebljavaju samo sigurnosni mehanizmi nižih protokola, oni osiguravaju komunikaciju između dvaju čvorova koji upotrebljavaju taj protokol, ali oni ne moraju biti početni (izvorište) i završni (odredište) čvor. Jesu li početno izvorište (polje `SRC`) i završno odredište (polje `DST`), onda sigurnost ovisi samo o tom protokolu. Primjerice, ako se protokol TLS ispravno upotrebljava između dvaju čvorova na Internetu, ta je komunikacija potpuno sigurna. Ako u komunikaciji postoje međučvorovi, odnosno postoji više skokova između početnoga i završnoga čvora, onda između svakih dvaju čvorova koji implementiraju metaprotokol, ili, međučvorova IoT-a, postoje odvojeni sigurnosni mehanizmi koji se upotrebljavaju da bi se osigurala poveznica među njima. Npr., između stvari 1 i stvari 2, stvari 2 i usmjernika, usmjernika i poslužitelja može se cijelo vrijeme upotrebljavati protokol TLS ali mogu biti i različiti mehanizmi. Ugrozi li se samo jedan čvor ili poveznica, pada cijeli lanac sigurnosti jer takav kompromitirani čvor, odnosno zlonamjerni čvor na takvoj poveznici, ima pristup neosiguranim podatcima.

U trećem slučaju razmatra se osiguravanje samo metaprotokolom, koje bi se trebalo rijetko upotrebljavati. Naime, više-manje svi niži protokoli imaju neke sigurnosne mogućnosti optimizirane za sustave u kojima se upotrebljavaju i obično pokrivaju sve sigurnosne potrebe. Čak i u slučaju višeskokovnih (engl. *multi-hop*) sustava postoje protokoli za sigurnost „s kraja na kraj“ koji se mogu u njima upotrebljavati, jer uključivanje metaprotokolske sigurnosti za koju još nije optimiziran ni jedan sustav moglo bi zahtijevati više sredstava (procesorske snage, memorije). Sigurnost koju nudi metaprotokol usporediva je jer uključuje standardne mehanizme digitalne omotnice i digitalnoga potpisa u svrhu osiguravanja povjerljivosti i autentičnosti. Svaki čvor ima svoj privatni ključ asimetrične kriptografije i javne ključeve ostalih svojih sugovornika. Ti se ključevi obično raspodijele ručno u konfiguracijske postavke čvorova prije same komunikacije. Digitalna

omotnica osigurava da nitko ne može prisluškivati komunikaciju jer napadač nema privatni ključ primatelja. Ali ako napadač ima javni ključ primatelja, može mijenjati poruke (zamijeniti izvornu omotnicu svojom), tj. lažno se predstavljati. Upotrebljava li se samo digitalni potpis, može se prisluškivati ali se ne može ni mijenjati ni izmisliti poruka. Tek ako se oba mehanizma upotrijebe sprječava se i prisluškivanje i mijenjanje, odnosno, osigurava se i neporecivost i tajnost/povjerljivost i besprijekornost i autentičnost. Ni jedno od tih zahtjeva ne može se povrijediti jer napadač nema ni privatni ključ pošiljatelja ni privatni ključ primatelja. Jedini slučaj kad može biti slomljena neporecivost jest u slučaju posredničke sigurnosti ako se u skladu s opisanim u potpoglavlju 3.10 privatni ključ nekoga čvora daje nekomu drugomu čvoru, i u tom slučaju pošiljatelj bi mogao poreći da je on poslao poruku.

Što se tiče napada ponovnim odašiljanjem starih poruka, na prvi se pogled on čini vrlo opasnim jer ni omotnica ni potpis ne služe osiguravanju od njega. Postoji više načina kako se napad ponavljanjem može spriječiti. Najvažniji od tih načina jest uporaba polja `ID` primljene poruke. On se stalno uvećava u modulu 256 za određeno izvorište i određeno odredište, a zajedno s poljem `SRC` i poljem `DST` jednoznačno identificira poruku u zadanom razdoblju. Pokuša li se više puta poslati ista poruka, u tom se razdoblju mogu lako ustanoviti duplikati i odbaciti. Problem nastaje ako napadač pričeka dulje vrijeme i onda ponovno pošalje poruku, ali upravo zbog stalnoga slijednoga povećavanja može se lako vidjeti da je neka poruka došla izvan redoslijeda. Dodatna pravila koja to provjeravaju u tom slučaju odbacuju takvu poruku. Osim toga, postoje i drugi mehanizmi koji se mogu iskoristiti za stvaranje jedinstvene poruke, kao što je uključivanje vremenskoga biljega u poruke `DATA` kao dodatnoga stupca.

Zadnji napad koji će se obraditi jest napad prekidanjem, odnosno uskraćivanjem usluge. Napadi uskraćivanjem usluge nemaju smisla u samom metaprotokolu jer on ne čuva stanje nego samo odgovara na zahtjeve, za razliku od npr. protokola TCP. Ti zahtjevi se mogu gomilati, i zlonamjerni čvorovi mogu pokušati zakrčiti pretplate ili baze ako se ne upotrebljava nikakva sigurnost. Zato se u tom slučaju mogu upotrijebiti spominjani mehanizmi za sprječavanje ubacivanja zlonamjernika uz pomoć povjerljivosti i autentičnosti. To ne će spriječiti napade uskraćivanjem usluge na niže protokole, npr. protokol TCP. U tom se slučaju mogu zamijeniti niži protokoli nečim otpornijim, ili uključiti kakav vatrozid (engl. *firewall*) koji sprječava zakrčivanje nižega protokola.

Treba spomenuti i da se u čvoru mogu konfigurirati složena pravila slijednim okidanjem, i dio se sigurnosti može ostvariti i preko njih. Pravila mogu otkrivati moguće napade i odgovoriti na njih sprječavanjem dalje komunikacije sa zlonamjernim čvorovima ili čak javljanjem korisniku da nešto nije u redu.

6. ZAKLJUČAK

U ovom se radu predlaže nova arhitektura za primjenu u domeni IoT-a. Glavni su motivi stvaranja ovakve arhitekture lakše uključivanje jednostavnih čvorova u sustav i fleksibilnost za ostvarivanje po volji povezanih čvorova, bez nametanja hijerarhijske organizacije. Radi ostvarivanja navedenoga predložen je poseban protokol (metaprotokol) i model čvora koji ga upotrebljava.

Jednostavni se čvorovi ne trebaju zamarati mrežnom konfiguracijom sustava ni upotrebljavati složenije komunikacijske protokole nego se mogu osloniti na složenije čvorove koji će ih neizravno povezati s potrebnim odredištima (čvorovima).

Čvorovi koji pomažu uključivanje novih čvorova, ako upotrebljavaju predloženi model, ostvaruju to bilo jednostavnim implicitnim pravilima, bilo naprednima eksplicitnima. Ta pravila mogu preoblikovati poruke, proslijediti ih, pohraniti ih, odbaciti i sl.

Komunikacija se u predloženom sustavu temelji na posebnom protokolu koji se ovdje naziva „metaprotokolom“, zbog dosta značajkâ koje ga odvajaju od uobičajenih protokola. Kao najvažnija svrha metaprotokola ističe se omogućivanje komunikacije preko više postojećih nižih protokola, čije mogućnosti metaprotokol iskorištava, a dodaje samo ono što je još potrebno. Primjerice, ako niži protokol sadržava adresu konačnoga odredišta poruke, onda se takva adresa ne mora dodatno stavljati u samu poruku i sl.; ako je sigurnost ostvarena nižim protokolom ne mora ju ostvarivati metaprotokol.

Sadržaj je poruke modeliran na relacijskom modelu jezika SQL, s ciljem istodobna omogućivanja i jednostavna i napredna prijenosa, i razmjene, podataka i naredaba. Dva su osnovna tipa poruka: podatkovne poruke, kojima se šalju podatci odredišnomu čvoru, i zahtjevi za podatcima – upiti. Po primitku podatkovne poruke, tzv. tipa `DATA`, odredišni će čvor učiniti operaciju unosa podatka u bazu (naredba `INSERT`), a po primitku poruke s upitom obavit će upit nad bazom (naredba `SELECT`) i dobivenim podatcima odgovoriti pošiljatelju podatkovnom porukom. Uz ove osnovne mogućnosti postoje i druge, kao što su upravljanje porukama i upitima, od jednostavnih do složenijih.

Naprednije poruke jezika SQL mogu imati naprednije upite koji preoblikuju podatke, agregiraju ih, i sl., a rezultat zahtijevaju natrag k pošiljatelju. Na jezik SQL se ovdje nadovezuju i poruke koje služe pretplaćivanju na podatke. U svakom slučaju, zajedno s ostalim tipovima poruka i ostalim svrhama ovdje kratko spomenutih tipova – omogućuje se

stvaranje različitih sustava. To mogu biti standardni slojeviti sustavi IoT-a, tj. stvar-usmjernik-poslužitelj-klijent, ali i sustavi bez pridijeljenih kategorija čvorova s posebno krojenim ulogama za čvorove.

Na nekolicini primjera, jednostavnijih i složenijih, pokazala se je uporaba predložene arhitekture. Raščlamba tih primjera i usporedba sa sličnim rješenjima, istraživačkima i komercijalnima, pokazuje određene prednosti predložene arhitekture u pogledu jednostavnosti uporabe i fleksibilnosti u ostvarivanju sustava.

Osim opisa metaprotokola i modela čvora u radu je ostvaren i konkretan prototip programske potpore temeljen na njima. On se može upotrijebiti za ispitivanje svih značajkâ različitim načinima, bilo uporabom u čvorovima koji skladište podatke i obrađuju ih pa upotrebljavaju skoro sve dijelove modela, bilo jednostavnim čvorovima koji upotrebljavaju samo neke dijelove, a možda i samo jedan njegov dio.

Programska potpora uključuje tzv. *middleware* za čvorove IoT-a napisan u jeziku C++, sučelje za *web* za konfiguraciju pametnoga čvora napisano u jeziku PHP, ispitne scenarije napisane u jeziku `bash`, ispitnu aplikaciju napisanu u jeziku Kotlin i još nekoliko sitnih stvari a nalazi se na mjestu za *web* GitHub.

Predložena arhitektura ne isključuje uporabu drugih rješenja, pa se može upotrijebiti samo u jednom dijelu nekoga sustava IoT-a gdje su važne njezine značajke u odnosu na ta druga rješenja. Kao što se i metaprotokol svagdje zasniva na jednakom fleksibilnom modelu podataka, a dodaje u komunikaciju samo stvari koje nedostaju, na isti se način može promatrati i cijeli model arhitekture. To jest, model podataka omogućuje interoperabilnost s drugim rješenjima, a arhitektura u cjelini omogućuje primjenu u mnogim sustavima IoT-a ili podsustavima različite složenosti.

U disertaciji je ostvaren dvojak znanstveni doprinos: 1. metaprotokol za prijenos podataka u Internetu stvari temeljen na operacijama svojstvenim bazama podataka (opisano u potpoglavljima 3.3, 3.5-6, 3.10-11) i 2. arhitektura sustava Interneta stvari temeljena na izgrađenom metaprotokolu i sustavu pravila s naglaskom na prilagodljivost i jednostavnost (opisano u ostatku poglavlja 3).

Budući rad uključuje završetak ostvarivanja programske potpore koja će pokrivati sve operacije predložene u ovom radu. Onda se mogu izvesti različite raščlambe, od sklopovskih zahtjeva do evaluacije performansâ za različite čvorove i operacije. Dalje istraživanje može isto tako uključivati upogonjivanje predloženoga metaprotokola,

programske potpore i prilagođenih sredstvima ograničenih stvari u stvarnim okružjima za rješavanje stvarnih problema. Onda bi se mogle dalje evaluirati jednostavnost i fleksibilnost predložene arhitekture, prikazane u ovom radu na sintetičkim primjerima.

LITERATURA

1. Nižetić, S., Šolić, P., López-de-Ipiña González-de-Artaza, D., "Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future", Journal of Cleaner Production, Vol. 274, No. 1, 2020, str. 122877.
2. IBM Report, "Smart Planet", 2008.
3. ITU Internet Reports 2005, "The Internet of Things", 2005.
4. Atzori, L, Iera, A., Morabito, G., "The Internet of Things: A survey", Computer Networks, Vol. 54, No. 15, 2010, str. 2787-2805.
5. Lin, J., Yu, W., Zhang, N., "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications", IEEE Internet of Things Journal, Vol. 4, No. 5, 2017, str. 1125-1142.
6. IEEExplore, "IEEExplore Search Results", dostupno na https://ieeexplore.ieee.org/search/searchresult.jsp?queryText=internet+of+things (1. travnja 2022.)
7. IBM, "Internet of Things | IBM", dostupno na http://www.ibm.com/cloud/internet-of-things (1. travnja 2022.)
8. Microsoft, "Azure IoT - Internet of Things Platform | Microsoft Azure", dostupno na https://azure.microsoft.com/en-us/overview/iot/ (1. travnja 2022.)
9. Amazon, "AWS IoT - Amazon Web Services", dostupno na http://aws.amazon.com/iot/ (1. travnja 2022.)
10. Rana, D., "Top 11 Cloud Platforms for Internet of Things (IoT) – DZone IoT", dostupno na https://dzone.com/articles/10-cloud-platforms-for-internet-of-things-iot (1. travnja 2022.)
11. Galloway, B., Hancke, G. P., "Introduction to Industrial Control Networks", IEEE Communications Surveys & Tutorials, Vol. 15, No. 2, 2012, str. 860-880.
12. Ashton, K., "That 'Internet of things' Thing", RFID Journal, Vol. 22, No. 7, 2009, str. 97-114.
13. Nord, J. H., Koohang, A., Paliszkiewicz, J., "The Internet of Things: Review and theoretical framework", Expert Systems with Applications, Vol. 133, No. 1, 2019, str. 97-108.
14. ISO/IEC 9075-2:2016, "Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation) Ed 5", 2016.

15. Google Scholar, "internet of things - Google Scholar", dostupno na https://scholar.google.hr/scholar?q=internet+of+things (1. travnja 2022.)
16. Risteska Stojkoska, B. L., Trivodaliev, K. V., "A review of Internet of Things for smart home: Challenges and solutions", Journal of Cleaner Production, Vol. 140, No. 3, 2017, str. 1454-1464.
17. Mugauri, P. C., Aravind, K., Desmukh, A., "A Survey on Applications of Internet of Things in Healthcare Domain", Research Journal of Pharmacy and Technology, Vol. 11, No. 1, 2018, str. 93-96.
18. Castro, M., Guillen, A., Fuster, J. L., "Oxygen Cylinders Management Architecture Based on Internet of Things", Proceedings of the 2011 International Conference on Computational Science and Its Applications, Santander, Spain, 2011., str. 271-274.
19. Hu, W., Wang, X., Kan, A., "The architecture and desing for early warning system of Tibet tourism crisis based on the Internet of Things", Proceedings of the 2011 International Conference on Electrical and Control Engineering", Yichang, China, 2011, str. 3695-3698.
20. Zorzi, M., Gluhak, A., Lange, S., "From today's INTRAnet of things to a future INTERnet of things; a wireless- and mobility-related view", IEEE Wireless Communications, Vol. 17, No. 6, 2010, str. 44-51.
21. Tan, L., Wang, N., "Future internet: The Internet of Things", Proceedings of the 2010 3rd International Conference on Advanced Computer Theory and Engineering, Chengdu, China, 2010., str. 5.376-5.380.
22. Yang, Y., Wu, L., Yin, G., "A Survey on Security and Privacy Issues in Internet-of-Things", IEEE Internet of Things Journal, Vol. 4, No. 5, 2017, str. 1250-1258.
23. Chakravarty, S., "Did you know Google is tracking you 24×7? Here is how you can stop it", dostupno na https://geospatialworld.net/blogs/did-you-know-google-is-tracking-you-24x7-here-is-how-you-can-stop-it/ (1. travnja 2022.)
24. Qiu, T., Chen, N., Li, K., "How Can Heterogeneous Internet of Things Build Our Future: A Survey", IEEE Communication Surveys & Tutorials, Vol. 20, No. 3, 2018, str. 2011-2027.
25. Ciortea, A., Boissier, O., Zimmermann, A., "Responsive Decentralized Composition of Service Mashups for the Internet of Things", Proceedings of the 6th International Conference on the Internet of Things, Stuttgart, Germany, 2016., str. 53-61.

26. Dias, J. P., Faria, J. P., Ferreira, H. S., "A Reactive and Model-Based Approach for Developing Internet-of-Things Systems", Proceedings of the 2018 11th International Conference on Quality of Information and Communications Technology, Coimbra, Portugal, 2018., str. 276-281.
27. European Union INFSO-2009-00136-00-00-EN-REV-00, "Internet of Things – Action Plan for Europe", 2009.
28. Hosek, J., Masek, P., Andreev, S., "A SyMPHOnY of Integrated IoT Businesses: Closing the Gap between Availability and Adoption", IEEE Communications Magazine, Vol. 55, No. 12, 2017, str. 156-164.
29. Christiansen, T., Foy, B. D., Wall, L., "Programming Perl, 4th Edition", O'Reilly Media, Sebastopol, 2012.
30. Milić, L., Jelenković, L., "A novel versatile architecture for Internet of Things", Proceedings of the 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 2015., str. 1026-1031.
31. Bluetooth Specification, "Core Specification 5.3", 2021.
32. Raza, S., Misra, P., He, Z., "Bluetooth smart: An enabling technology for the Internet of Things", Proceedings of the 2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications", Abu Dhabi, UAE, 2015., str. 155-162.
33. IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016), "IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks--Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", 2021.
34. ABIresearch, "Emerging 802.11ah Low-Power Wi-Fi Standard to Face Difficult Path to Adoption in Crowded IoT Connectivity Space", dostupno na https://www.abiresearch.com/press/emerging-80211ah-low-power-wi-fi-standard-face-dif/ (1. travnja 2022.)
35. ISO/IEC/IEEE 8802-15-6:2017(E), "ISO/IEC/IEEE International Standard - Information Technology -- Telecommunications and information exchange between systems -- Local and metropolitan area networks -- Specific Requirements -- Part 15-6: Wireless body area network", 2018.

36. IEEE Std 802.16-2017 (Revision of IEEE Std 802.16-2012), "IEEE Standard for Air Interface for Broadband Wireless Access Systems", 2018.
37. Oliveira, L., Rodrigues, J. J. P. C., Kozlow, S. A., "MAC Layer protocols for Internet of things: A Survey", Future Internet, Vol. 11, No. 1, 2019, str. 16-57.
38. LoRa Alliance®, "LoRaWAN® 1.0.4 Specification Package", 2020.
39. SigFox Build, "SigFox_Radio_Specifications_v1.5", 2020.
40. Texas Instruments, "Wireless Connectivity Products | Overview | TI.com", dostupno na https://www.ti.com/wireless-connectivity/overview.html (1. travnja 2022.)
41. IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015), "IEEE Standard for Low-Rate Wireless Networks", 2020.
42. ZigBee Document 05-3474-21, "ZigBee Specification", 2015.
43. Thread Group, "Thread 1.1 Specification", 2016.
44. IETF RFC 4944, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", 2007.
45. Z-Wave Alliance 2020C, "Z-Wave Specification", 2020.
46. IETF RFC 8200, "Internet Protocol, Version 6 (IPv6) Specification", 2017.
47. Google, "IPv6 - Google", dostupno na https://www.google.com/intl/en/ipv6/statistics.html (1. travnja 2022.)
48. Toumi, K., Ayari, M., Azouz Saidane, L., "HAT: HIP Address Translation Protocol for Hybrid RFID/IP Internet of Things communication", Proceedings of the 2010 International Conference on Wireless and Ubiquitous Systems, Sousse, Tunisia, 2010, str. 1-7.
49. Seskar, I., Nagaraja, K., Nelson, S., "MobilityFirst future internet architecture project", Proceedings of the 7th Asian Internet Engineering Conference, Bangkok, Thailand, 2011., str. 1-3.
50. Asim, M., "A Survey on Application Layer Protocols for Internet of Things (IoT)", International Journal of Advanced Research in Computer Science, Vol. 8, No. 3, 2017, str. 996-1000.
51. IETF RFC 7252, "The Constrained Application Protocol (CoAP)", 2014.
52. IETF RFC 6120, "Extensible Messaging and Presence Protocol (XMPP): Core", 2011.
53. OASIS Group, "MQTT Version 5.0", 2019.
54. OASIS Group, "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0", 2012.
55. Richardson, L., Ruby, S., "RESTful Web Services", O'Reilly Media, Sebastopol, 2018.

56. W3C Recommendation, "SPARQL 1.1 Protocol", 2013.
57. W3C, "Semantic Web Standards", dostupno na https://www.w3.org/2001/sw/wiki/Main_Page (1. travnja 2022.)
58. W3C Recommendation, "RDF 1.1 Concepts and Abstract Syntax", 2014.
59. Hafidh, B., Al Osman, H., Arteaga-Falconi, J. S., "SITE: The Simple Internet of Things Enabler for Smart Homes", IEEE Access, Vol. 5, No. 1, 2017, str. 2034-2049.
60. PostgreSQL, "Cyclic Tag System - PostgreSQL wiki", dostupno na https://wiki.postgresql.org/Cyclic_Tag_System (1. travnja 2022.)
61. Park, N. S., Lee, H. K., Jang, J., "Rule-based modeling tool for web of things applications", Proceedings of the 2015 IEEE 5th International Conference on Consumer Electronics - Berlin, Berlin, Germany, 2015., str. 515-518.
62. Kaed, C. E., Khan, I., Van Der Berg, A., "SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways", IEEE Transactions on Industrial Informatics, Vol. 14, No. 2, 2018, str. 715-724.
63. Mazzei, D., Fantoni, G., Montelisciani, G., "Internet of Things for designing smart objects", Proceedings of the 2014 IEEE World Forum on Internet of Things, Seoul, South Korea, 2014., str. 293-297.
64. Yao, L., Sheng Q. Z., Dustdar, S., "Web-Based Management of the Internet of Things", IEEE Internet Computing, Vol. 19, No. 4, 2015, str. 60-67.
65. Yangqun, L., "A Light-Weight Rule-Based Monitoring Systems for Web of Things", Proceedings of the 2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Beijing, China, 2013., str. 251-254.
66. Tuomisto, T., Kymäläinen, T., Plomp, J., "Simple Rule Editor for the Internet of Things", Proceedings of the 2014 International Conference on Intelligent Environments, Shanghai, China, 2014., str. 384-387.
67. Monge Roffarrello, A., "End User Development in the IoT: A Semantic Approach", Proceedings of the 2018 14th International Conference on Intelligent Environments, Rome, Italy, 2018., str. 107-110.
68. Hwang, I., Kim, M., Ahn, H. J., "Data Pipeline for Generation and Recommendation of the IoT Rules Based on Open Text Data", Proceedings of the 2016 30th International Conference on Advanced Information Networking and Applications Workshops, Crans-Montana, Switzerland, 2016., str. 238-242.

69. Hossayni, H., Khan, I., Kaed, C. E., "Embedded Semantic Engine for Numerical Time Series Data", Proceedings of the 2018 Global Internet of Things Summit, Bilbao, Spain, 2018., str. 1-6.
70. Baricelli, B. R., Valtolina, S., "A visual language and interactive system for end-user development of internet of things ecosystems", Journal of Visual Languages & Computing, Vol. 40, No. 1, 2017, str. 1-19.
71. Red Hat, "Drools - Drools - Business Rules Management System (Java™, Open Source)", dostupno na https://drools.org (1. travnja 2022.)
72. IEEE RFC 7228, "Terminology for Constrained-Node Networks", 2014.
73. Malina, L., Srivastava, G., Dzurenda, P., "A Secure Publish/Subscribe Protocol for Internet of Things", Proceedings of the 14th International Conference on Availability, Reliability and Security, Canterbury, UK, 2019., str. 1-10.
74. Microsoft, "Tutorial - Store data with SQL module using Azure IoT Edge | Microsoft Docs", dostupno na https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-store-data-sql-server (1. travnja 2022.)
75. IEEE, "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", 2017.
76. IEEE, "Welcome to The Public Listing For IEEE Standards Registration Authority", dostupno na https://regauth.standards.ieee.org/standards-ra-web/pub/view.html (1. travnja 2022.)
77. IETF RFC 4291, "IP Version 6 Addressing Architecture", 2006.
78. Hwang, G., Lee, J., Park, J., "Developing performance measurement system for Internet of Things and smart factory environment", International Journal of Production Research, Vol. 55, No. 9, 2017., str. 2590-2602.
79. Signal, "Signal >> Documentation", dostupno na https://signal.org/docs (1. travnja 2022.)
80. Castagnoli, G., Brauer, S., Herrmann, M., "Optimization of cyclic redundancy-check codes with 24 and 2 parity bits", IEEE Transactions on Communications, Vol. 41, No. 6, 1993, str. 883-892.
81. Brayer, K., Hammond, J. L. Jr., "Evaluation of error detection polynomial performance on the AUTOVON channel", Proceedings of the 1975 IEEE National Communications Conference, New Orleans, USA, 1975., str. 8.21-8.25.

82. Koopman, P., "32-Bit cyclic redundancy codes for Internet applications", Proceedings of the 2002 International Conference on Dependable Systems and Networks, Bethesda, USA, 2002., str. 459-468.
83. Cloutier, F., "CRC32 – Accumulate CRC32 Value", dostupno na https://felixcloutier.com/x86/crc32 (1. travnja 2022.)
84. Koopman, P., Chakravarty, T., "Cyclic redundancy code (CRC) polynomial selection for embedded networks", Proceedings of the 2004 International Conference on Dependable Systems and Networks, Florence, Italy, 2004., str. 145-154.
85. Carnegie Mellon University, "8 Bit CRC Zoo", dostupno na https://users.ece.cmu.edu/~koopman/crc/crc8.html (1. travnja 2022.)
86. Milić, L., Jelenković, L., Magdalenić, I., "A Metaprotocol-Based Internet of Things Architecture", Automatika, Vol. 63, No. 4, 2022, str. 676-694.
87. Texas Instruments, "Temperature Sensors | Products | Sensors | TI.com", dostupno na https://www.ti.com/sensors/temperature-sensors/products.html" (1. travnja 2022.)
88. ISO/IEC JTC1/SC32, "N 2572", dostupno na http://jtc1sc32.org/doc/N2301-2350/32N2311T-text_for_ballot-CD_9075-2.pdf (1. travnja 2022.)
89. Winand, M., "What's New in SQL:2016", dostupno na https://modern-sql.com/blog/2017-06/whats-new-in-sql-2016 (1. travnja 2022.)
90. ISO/IEC 8859-1:1998, "Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1", 1998.
91. IETF RFC 3629, "UTF-8, a transformation format of ISO 10646", 2003.
92. IETF RFC 8446, "The Transport Layer Security (TLS) Protocol Version 1.3", 2018.
93. IETF RFC 9147, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", 2022.
94. IANA, "Service Name and Transport Protocol Port Number Registry", 2021.
95. Chakrabaty S., John, M., Engels, D. W., "Black routing and node obscuring in IoT", Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things, Reston, USA, 2016., str. 323-328.
96. OpenSSL, "/docs/man3.0.0/man1/ciphers.html", dostupno na https://www.openssl.org/docs/man3.0.0/man1/ciphers.html (1. travnja 2022.)
97. NIST Federal Information Processing Standards Publication 197, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)", 2001.
98. IETF RFC 8017, "PKCS #1: RSA Cryptography Specifications Version 2.2", 2016.

99. IETF RFC 6234, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", 2011.
100. IETF RFC 5652, "Cryptographic Message Syntax (CMS)", 2009.
101. NSA CNSS Policy No. 15, Fact Sheet No. 1, "National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information", 2003.
102. NIST, "Post-Quantum Cryptography Standardization", 2016.
103. IETF RFC 8996, "Deprecating TLS 1.0 and TLS 1.1", 2021.
104. PostgreSQL, "PostgreSQL: Documentation: 14: 4.1. Lexical Structure", dostupno na https://www.postgresql.org/docs/current/sql-syntax-lexical.html (1. travnja 2022.)
105. Cloutier, F., "x86 and amd64 instruction reference", dostupno na https://felixcloutier.com/x86/index.html (1. travnja 2022.)
106. IANA, "IANA IPv4 Special-Purpose Address Registry", 2021.
107. IETF RFC 791, "Internet Protocol", 1981.
108. Texas Instruments, "CC2652R7 SimpleLink™ Multiprotocol 2.4 GHz Wireless MCU datasheet - cc2652r7.pdf", dostupno na https://www.ti.com/lit/ds/symlink/cc2652r7.pdf" (1. travnja 2022.)
109. Rustagi, D., "Introduction to Linear Bounded Automata (LBA)", dostupno na https://www.geeksforgeeks.org/introduction-to-linear-bounded-automata-lba/ (1. travnja 2022.)
110. Muhammed, A. S., Ucuz, D., "Comparison of the IoT Platform Vendors, Microsoft Azure, Amazon Web Services, and Google Cloud, from Users' Perspectives", Proceedings of the 2020 8th International Symposium on Digital Forensics and Security, Beirut, Lebanon, 2020., str. 1-4.
111. IBM, "Node-RED", dostupno na https://nodered.org (1. travnja 2022.)
112. Zahoor, S., Mir., R. N., "Resource management in pervasive Internet of Things: A survey", Journal of King Saud University – Computing and Information Sciences, Vol. 33, No. 8, 2021, str. 921-935.
113. Milić, L., "GitHub - lukamilicfoi/IoT", dostupno na: https://github.com/lukamilicfoi/IoT (1. travnja 2022.)
114. PostgreSQL, "PostgreSQL: Documentation: 14: Appendix D. SQL Conformance", dostupno na https://www.postgresql.org/docs/current/features.html (1. travnja 2022.)

115. W^{3}Techs, "Usage Statistics and Market Share of Web Servers, February 2022", dostupno na https://w3techs.com/technologies/overview/web_server (1. travnja 2022.)
116. ArchLinux, "ArchLinux", dostupno na https://archlinux.org (1. travnja 2022.)
117. Milić, L., "IoT/install_archlinux_dependencies.sh at master · lukamiliciot · GitHub", dostupno na https://github.com/lukamilicfoi/IoT/blob/master/install_archlinux_dependencies.sh (1. travnja 2022.)
118. Luka Milić, "IoT/README at master · lukamilicfoi/IoT · GitHub", dostupno na https://github.com/lukamilicfoi/IoT/blob/master/README (1. travnja 2022.)
119. ArchWiki, "systemd-nspawn", dostupno na https://wiki.archlinux.org/title/systemd-nspawn (1. travnja 2022.)
120. Free Software Foundation, "GNU General Public License v2.0 - GNU Project - Free Software Foundation", dostupno na https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html (1. travnja 2022.)
121. Eclipse Foundation, "Eclipse IDE 2022-03 | The Eclipse Foundation", dostupno na https://www.eclipse.org/eclipseide/ (1. travnja 2022.)
122. OpenSSL, "/index.html", dostupno na https://www.openssl.org (1. travnja 2022.)

PRILOG A: ZAMJENA KLJUČNIH RIJEČI JEZIKA SQL

Znak	Riječ	Komentar uz zamjenu
0x80	<=	
0x81	=>	
0x82	<>	
0x83	||	
0x84	AND	za konjunkciju i BETWEEN
0x85	ALL	za sintagme SELECT ALL („šum“), <skupovni operator> ALL i <relacijski operator> ALL (<podupit>)
0x86	ANY	za sintagme <relacijski operator> ANY (<podupit>), ANY (<podupit>) i agregatnu funkciju
0x87	AS	za preimenovanje stupaca, tablica i obojega zajedno
0x88	ASC	„šumna riječ“
0x89	ASYMMETRIC	„šumna riječ“
0x8A	AT	
0x8B	AVG	za agregatnu funkciju
0x8C	BETWEEN	
0x8D	BERNOULLI	
0x8E	BREADTH	
0x8F	BY	za GROUP BY, ORDER BY, CORRESPONDING BY, BY BREADTH i BY DEPTH
0x90	CASE	za dva oblika izraza CASE
0x91	COALESCE	
0x92	COLLATE	
0x93	CORRESPONDING	
0x94	COUNT	za agregatnu funkciju
0x95	CROSS	
0x96	CUBE	
0x97	CURRENT_DATE	

0x98	CURRENT_TIME	
0x99	CURRENT_TIMESTAMP	
0x9A	CYCLE	
0x9B	DATE	
0x9C	DAY	
0x9D	DEFAULT	
0x9E	DEPTH	
0x9F	DESC	
0xA0	DISTINCT	
0xA1	ELSE	za dva oblika izraza CASE
0xA2	ESCAPE	za LIKE i SIMILAR
0xA3	END	za dva oblika izraza CASE
0xA4	EVERY	za agregatnu funkciju
0xA5	EXCEPT	
0xA6	EXISTS	
0xA7	FALSE	za literale i IS FALSE
0xA8	FETCH	
0xA9	FILTER	
0xAA	FIRST	za FETCH (istoznačno s riječju NEXT), NULLS FIRST, BREADTH FIRST i DEPTH FIRST
0xAB	FLAG	
0xAC	FROM	za SELECT i IS DISTINCT FROM
0xAD	FULL	za FULL OUTER JOIN i MATCH FULL
0xAE	GROUP	
0xAF	GROUPING	za GROUPING i GROUPING SETS
0xB0	HAVING	
0xB1	HOUR	
0xB2	IN	
0xB3	INNER	„šumna riječ“

0xB4	INTERSECT	
0xB5	INTERVAL	
0xB6	IS	
0xB7	JOIN	
0xB8	LAST	
0xB9	LATERAL	
0xBA	LEFT	
0xBB	LIKE	za usporedbu znakova i usporedbu bajtova
0xBC	LIKE_REGEX	
0xBD	LOCAL	
0xBE	LOCALTIME	
0xBF	LOCALTIMESTAMP	
0xC0	MATCH	
0xC1	MAX	za agregatnu funkciju
0xC2	MIN	za agregatnu funkciju
0xC3	MINUTE	
0xC4	MONTH	
0xC5	NATURAL	
0xC6	NEXT	
0xC7	NORMALIZED	
0xC8	NOT	za nijekanje, IS NOT, NOT IN, NOT LIKE, NOT BETWEEN, NOT LIKE_REGEX i NOT SIMILAR
0xC9	NULL	za literale i IS NULL
0xCA	NULLIF	
0xCB	NULLS	
0xCC	OF	
0xCD	OFFSET	
0xCE	ON	
0xCF	ONLY	

0xD0	OR	
0xD1	ORDER	
0xD2	OUTER	„šumna riječ“
0xD3	OVERLAPS	
0xD4	PARTIAL	
0xD5	PERCENT	
0xD6	RECURSIVE	
0xD7	REPEATABLE	
0xD8	RIGHT	
0xD9	ROLLUP	
0xDA	ROW	za OFFSET <broj> ROW i FETCH <broj> ROW
0xDB	ROWS	istoznačno s riječju ROW
0xDC	SEARCH	
0xDD	SECOND	
0xDE	SELECT	za korijenske odabire, WITH, podupite, jednočlanske podupite u popisu izraza SELECT i tablične podupite u popisu izraza FROM
0xDF	SET	za SEARCH i CYCLE
0xE0	SETS	
0xE1	SIMILAR	
0xE2	SIMPLE	„šumna riječ“
0xE3	SOME	istoznačno s riječju ANY
0xE4	STDDEV_POP	
0xE5	STDEV_SAMP	
0xE6	SUBSCRIBE	izmišljena ključna riječ
0xE7	SUM	
0xE8	SYMMETRIC	
0xE9	SYSTEM	
0xEA	TABLE	skoro istoznačno s oblikom SELECT * FROM
0xEB	TABLESAMPLE	

0xEC	THEN	za dva oblika izraza CASE
0xED	TIES	
0xEE	TIME	za TIME i TIME ZONE
0xEF	TIMESTAMP	
0xF0	TO	za SIMILAR TO, SEARCH i intervale
0xF1	TRUE	za literale i IS TRUE
0xF2	UESCAPE	
0xF3	UNION	UNION JOIN više ne postoji
0xF4	UNIQUE	za UNIQUE i MATCH UNIQUE
0xF5	UNKNOWN	za literale i IS UNKNOWN
0xF6	USING	za JOIN <tablica> USING i CYCLE
0xF7	UNSUBSCRIBE	izmišljena ključna riječ
0xF8	VALUES	
0xF9	VAR_POP	
0xFA	VAR_SAMP	
0xFB	WHEN	za dva oblika izraza CASE
0xFC	WHERE	
0xFD	WITH	za WITH TIES i predupit izraza WITH
0xFE	YEAR	
0xFF	ZONE	

Životopis

Luka Milić rođen je 1990. u Zadru, gdje je 2008. završio srednju školu. Godine 2011. stekao je akademski naslov sveučilišnoga prvostupnika inženjera računarstva (modul: Računarska znanost) na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, a 2013. naslov magistra inženjera računarstva (smjer: Računarska znanost) na istom Fakultetu; za vrijeme studiranja dobio je tri nagrade za iznimno uspješan studij. Od 2013. zaposlen je na Fakultetu organizacije i informatike Sveučilišta u Zagrebu kao asistent u tehničkom području (polje računarstvo) na Katedri za informatičke tehnologije i računarstvo, za predmete iz područja operacijskih sustava, arhitekture računala i mreža računala. Od 2014. doktorski je student (polje računarstvo) na Fakultetu elektrotehnike i računarstva. Od 2015. član je Laboratorija za dizajn programskih sučelja, internetske servise i videoigre na Fakultetu organizacije i informatike. Bio je mentor na deset završnih radova, održao je jedno pozvano predavanje. Bio je član programskoga odbora jedne znanstvene konferencije i dviju stručnih konferencija; bio je recenzent na jednoj znanstvenoj konferenciji. Sudjelovao je na dvama znanstvenim projektima i jednom stručnom projektu. Objavio je dva rada u časopisu i šest radova na konferencijama.

Popis radova

1. Milić, L., Jelenković, L., Magdalenić, I., "A Metaprotocol-Based Internet of Things Architecture", Automatika, Vol. 63, No. 4, 2022, str. 676-694.
2. Ivković, N., Magdalenić, I., Milić, L., "An Ad-Hoc Smartphone-to-Smartphone Live Multimedia Streaming Application with Real-Time Constraints", Journal of Advances in Computer Networks, Vol. 4, No. 1, 2016, str. 6-12.

1. Ivković, N., Milić, L., Konecki, M., "A Timed Automata Model for Systems with Gateway-Connected Controller Area Networks", Proceedings of the 2018 IEEE 3rd International Conference on Communication and Information Systems, Singapore, Singapore, 2018., str. 97-101.
2. Konecki, M., Okreša Đurić, B., Milić, L., "Using computer games as an aiding means in programming education", Proceedings of the 2015 5th Multidisciplinary Academic Conference, Prague, Czechia, 2015., str. 1-8.

3. Konecki, M., Tomičić, I., Milić, L. "Natural language processing and artificial intelligence in everyday life", Proceedings of the 2015 International Academic Conference on Social Sciences and Humanities in Prague, Prague, Czechia, 2015., str. 239-243.
4. Konecki, M., Pihir, I., Milić, L., "GUIDL: Developing a programming language for the visually impaired", Proceedings of the 2015 International Academic Conference on Social Sciences and Humanities in Prague, Prague, Czechia, 2015., str. 233-238.
5. Milić, L., Jelenković, L., "A novel versatile architecture for Internet of Things", Proceedings of the 2015 IEEE 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 2015.
6. Milić, L., Jelenković, L., "Improving thread scheduling by thread grouping in heavily loaded many-core processor systems", Proceedings of the 2014 IEEE 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, 2014., str. 1243-1246.

Biography

Luka Milić was born in 1990 in Zadar, where in 2008 he graduated from high school. In 2011 he acquired the academic degree of University Bachelor of Engineering in Computing (module: Computer Science) on the Faculty of Electrical Engineering and Computing of the University of Zagreb, and in 2013 the degree of Master of Engineering in Computing (course: Computer Science) on the same Faculty; during his studies he received three awards for exceptionally successful studying. Since 2013 he is employed on the Faculty of Organization and Informatics of the University of Zagreb as a teaching assistant in the technical area (field of Computing) on the Department of Computing and Technology, for subjects from the area of operating systems, computer architecture, and computer networks. Since 2014 he is a doctoral student (field of Computing) on the Faculty of Electrical Engineering and Computing. Since 2015 he is a member of the Laboratory for Design of Program Interfaces, Internet Services and Videogames on the Faculty of Organization and Informatics. He was a supervisor on ten Bachelor Theses, he gave one invited lecture. He was a program committee member of one scientific conference and two expert conferences; he was a reviewer at one scientific conference. He took part in two scientific projects and one expert project. He published two journal articles and six conference articles.