WIKIPEDIA
The Free Encyclopedia

Article Talk Read Edit View history Search Wikipedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

Print/export
Download as PDF
Printable version

Languages
العربية
Català
Čeština
Deutsch
فارسی
Français
한국어
日本語
中文
Edit links

Constrained Application Protocol

From Wikipedia, the free encyclopedia

Constrained Application Protocol (**CoAP**) is a specialized Internet application protocol for constrained devices, as defined in RFC 7252. It enables those constrained devices called "nodes" to communicate with the wider Internet using similar protocols. CoAP is designed for use between devices on the same constrained network (e.g., low-power, lossy networks), between devices and general nodes on the Internet, and between devices on different constrained networks both joined by an internet. CoAP is also being used via other mechanisms, such as SMS on mobile communication networks.

CoAP is a service layer protocol that is intended for use in resource-constrained internet devices, such as wireless sensor network nodes. CoAP is designed to easily translate to HTTP for simplified integration with the web, while also meeting specialized requirements such as multicast support, very low overhead, and simplicity.[1][2] Multicast, low overhead, and simplicity are important for Internet of things (IoT) and machine-to-machine (M2M) communication, which tend to be deeply embedded and have much less memory and power supply than traditional internet devices have. Therefore, efficiency is very important. CoAP can run on most devices that support UDP or a UDP analogue.

The Internet Engineering Task Force (IETF) Constrained RESTful Environments Working Group (CoRE) has done the major standardization work for this protocol. In order to make the protocol suitable to IoT and M2M applications, various new functions have been added.

Internet protocol suite

Application layer
BGP · DHCP(v6) · DNS · FTP · HTTP · HTTPS · IMAP · LDAP · MGCP · MQTT · NNTP · NTP · OSPF · POP · PTP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer
TCP · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer
IP (IPv4 · IPv6) · ICMP(v6) · ECN · IGMP · IPsec · *more...*

Link layer
ARP · NDP · Tunnels (L2TP) · PPP · MAC (Ethernet · Wi-Fi · DSL · ISDN · FDDI) *more...*

V · T · E

Contents [hide]

1 Specification
2 Message formats
2.1 CoAP Fixed Header: Version, Type, Token Length, Request/Response Code and Message ID.
2.1.1 Version (VER) (2 bits)
2.1.2 Type (2 bits)
2.1.3 Token Length (4 bits)
2.1.4 Request/Response Code (8 bits)
2.1.5 Message ID (16 bits)
2.2 Token
2.3 Option
3 Implementations
4 Proxy implementations
5 CoAP group communication
6 Security
7 Security issues
8 See also

9 References
10 External links

Specification [edit]

The core of the protocol is specified in RFC 7252. Various extensions have been proposed, particularly:

- RFC 7641 (2015) Observing Resources in the Constrained Application Protocol
- RFC 7959 (2016) Block-Wise Transfers in the Constrained Application Protocol (CoAP)
- RFC 8323 (2018) CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
- RFC 8974 (2021) Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP)

Message formats [edit]

The smallest CoAP message is 4 bytes in length, if omitting Token, Options and Payload. CoAP makes use of two message types, requests and responses, using a simple, binary, base header format. The base header may be followed by options in an optimized Type-Length-Value format. CoAP is by default bound to UDP and optionally to DTLS, providing a high level of communications security.

Any bytes after the headers in the packet are considered the message body. The length of the message body is implied by the datagram length. When bound to UDP, the entire message MUST fit within a single datagram. When used with 6LoWPAN as defined in RFC 4944, messages SHOULD fit into a single IEEE 802.15.4 frame to minimize fragmentation.

CoAP Header

Offsets	Octet	0								1								2								3							
Octet	**Bit**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
4	**32**	VER		Type		Token Length				Request/Response Code								Message ID															
8	**64**	Token (0 - 8 bytes)																															
12	**96**																																
16	**128**	Options (If Available)																															
20	**160**	1	1	1	1	1	1	1	1	Payload (If Available)																							

CoAP Fixed Header: Version, Type, Token Length, Request/Response Code and Message ID. [edit]

The first 4 bytes are mandatory in all CoAP datagrams.

These fields can be easily extracted from these 4 bytes in C via these macros:

```
#define COAP_HEADER_VERSION(data)  ( (0xC0 & data[0])>>6    )
#define COAP_HEADER_TYPE(data)     ( (0x30 & data[0])>>4    )
#define COAP_HEADER_TKL(data)      ( (0x0F & data[0])>>0    )
#define COAP_HEADER_CLASS(data)    ( ((data[1]>>5)&0x07)    )
#define COAP_HEADER_CODE(data)     ( ((data[1]>>0)&0x1F)    )
#define COAP_HEADER_MID(data)      ( (data[2]<<8)|(data[3]) )
```

Version (VER) (2 bits) [edit]

Indicates the CoAP version number.

Type (2 bits) [edit]

This describes the datagram's message type for the two message type context of Request and Response.

- Request
  - 0 : Confirmable : This message expects a corresponding Acknowledgement message.
  - 1 : Non-confirmable : This message does not expect a confirmation message.
- Response
  - 2 : Acknowledgement : This message is a response that acknowledge a confirmable message
  - 3 : Reset : This message indicates that it had received a message but could not process it.

Token Length (4 bits) [edit]

Indicates the length of the variable-length Token field, which may be 0-8 bytes in length.

Request/Response Code (8 bits) [edit]

0	1	2	3	4	5	6	7
Class			Code				

The three most significant bits form a number known as the "class", which is analogous to the class of HTTP status codes. The five least significant bits form a code that communicates further detail about the request or response. The entire code is typically communicated in the form `class.code` .

You can find the latest CoAP request/response codes at [1], though the below list gives some examples:

- Method: 0.XX
  0. EMPTY
  1. GET
  2. POST
  3. PUT
  4. DELETE
  5. FETCH
  6. PATCH
  7. iPATCH
- Success: 2.XX
  1. Created
  2. Deleted
  3. Valid
  4. Changed
  5. Content
  31. Continue
- Client Error: 4.XX
  0. Bad Request
  1. Unauthorized
  2. Bad Option
  3. Forbidden
  4. Not Found
  5. Method Not Allowed
  6. Not Acceptable
  8. Request Entity Incomplete
  9. Conflict
  12. Precondition Failed
  13. Request Entity Too Large
  15. Unsupported Content-Format
- Server Error: 5.XX
  0. Internal Server Error
  1. Not Implemented
  2. Bad Gateway
  3. Service Unavailable
  4. Gateway Timeout
  5. Proxying Not Supported
- Signaling Codes: 7.XX
  0. Unassigned
  1. CSM
  2. Ping
  3. Pong
  4. Release
  5. Abort

Message ID (16 bits) [edit]

Used to detect message duplication and to match messages of type Acknowledgement/Reset to messages of type Confirmable/Non-confirmable.:Response messages will have the same Message ID as request.

Token [edit]

Optional field whose size is indicated by the Token Length field, whose values is generated by the client. The server must echo every token value without any modification back to the client. It is intended for use as a client-local identifier to provide extra context for certain concurrent transactions.

Option [edit]

Option Format

Bit Positions							
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Option Delta				Option Length			
Option Delta Extended (None, 8 bits, 16 bits)							
Option Length Extended (None, 8 bits, 16 bits)							
Option Value							

Option Delta:

- 0 to 12: For delta between 0 to 12: Represents the exact delta value between the last option ID and the desired option ID, with no Option Delta Extended value
- 13: For delta from 13 to 268: Option Delta Extended is an 8-bit value that represents the Option Delta value minus 13
- 14: For delta from 269 to 65,804: Option Delta Extended is a 16-bit value that represents the Option Delta value minus 269
- 15: Reserved for Payload Marker, where the Option Delta and Option Length are set together as 0xFF.

Option Length:

- 0 to 12: For Option Length between 0 to 12: Represents the exact length value, with no Option Length Extended value
- 13: For Option Length from 13 to 268: Option Length Extended is an 8-bit value that represents the Option Length value minus 13
- 14: For Option Length from 269 to 65,804: Option Length Extended is a 16-bit value that represents the Option Length value minus 269
- 15: Reserved for future use. It is an error if Option Length field is set to 0xFF.

Option Value:

- Size of Option Value field is defined by Option Length value in bytes.
- Semantic and format this field depends on the respective option.

Implementations [edit]

Name	Programming Language	Implemented CoAP version	Client/Server	Implemented CoAP features	License	Link
aiocoap	Python 3	RFC 7252	Client + Server	Blockwise Transfers, Observe (partial)	MIT	https://pypi.python.org/pypi/aiocoap

Name	Programming Language	Implemented CoAP version	Client/Server	Implemented CoAP features	License	Link
Californium	Java	RFC 7252, RFC 7641, RFC 7959	Client + Server	Observe, Blockwise Transfers, Multicast (since 2.x), DTLS (+ DTLS 1.2 Connection ID)	EPL+EDL	https://www.eclipse.org/californium https://github.com/eclipse/californium
cantcoap	C++/C	RFC 7252	Client + Server		BSD	https://github.com/staropram/cantcoap
Canopus	Go	RFC 7252	Client + Server	Core	Apache License 2.0	https://github.com/zubairhamed/canopus
Go-CoAP	Go	RFC 7252, RFC 8232, RFC 7641, RFC 7959	Client + Server	Core, Observe, Blockwise, Multicast, TCP/TLS	Apache License 2.0	https://github.com/go-ocf/go-coap
CoAP implementation for Go	Go	RFC 7252	Client + Server	Core + Draft Subscribe	MIT	https://github.com/dustin/go-coap
CoAP.NET	C#	RFC 7252, coap-13, coap-08, coap-03	Client + Server	Core, Observe, Blockwise Transfers	3-clause BSD	https://github.com/smeshlink/CoAP.NET
CoAPSharp	C#, .NET	RFC 7252	Client + Server	Core, Observe, Block, RD	LGPL	http://www.coapsharp.com
CoAPthon	Python	RFC 7252	Client + Server + Forward Proxy + Reverse Proxy	Observe, Multicast server discovery, CoRE Link Format parsing, Block-wise	MIT	https://github.com/Tanganelli/CoAPthon
CoAP Shell	Java	RFC 7252	Client	Observe, Blockwise Transfers, DTLS	Apache License 2.0	https://github.com/tzolov/coap-shell
Copper	JavaScript (Browser Plugin)	RFC 7252	Client	Observe, Blockwise Transfers	3-clause BSD	https://github.com/mkovatsc/Copper https://addons.mozilla.org/firefox/addon/copper-270430/[*permanent dead link*]
eCoAP	C	RFC 7252	Client + Server	Core	MIT	https://gitlab.com/jobol/ecoap

Name	Programming Language	Implemented CoAP version	Client/Server	Implemented CoAP features	License	Link
Erbium for Contiki	C	RFC 7252	Client + Server	Observe, Blockwise Transfers	3-clause BSD	http://www.contiki-os.org/ (er-rest-example)
FreeCoAP	C	RFC 7252	Client + Server + HTTP/CoAP Proxy	Core, DTLS, Blockwise Transfers	BSD	https://github.com/keith-cullen/FreeCoAP
iCoAP	Objective-C	RFC 7252	Client	Core, Observe, Blockwise Transfers	MIT	https://github.com/stuffrabbit/iCoAP
java-coap	Java	RFC 7252, RFC 7641, RFC 7959, RFC 8323	Client + Server		Apache License 2.0	https://github.com/PelionIoT/java-coap
jCoAP	Java	RFC 7252	Client + Server	Observe, Blockwise Transfers	Apache License 2.0	https://code.google.com/p/jcoap/
libcoap	C	RFC 7252	Client + Server	Observe, Blockwise Transfers, DTLS	BSD/GPL	https://github.com/obgm/libcoap
LibNyoci	C	RFC 7252	Client + Server	Core, Observe, Block, DTLS	MIT	https://github.com/darconeous/libnyoci
lobaro-coap	C	RFC 7252	Client + Server	Observe, Blockwise Transfers	MIT	http://www.lobaro.com/lobaro-coap
microcoap	C	RFC 7252	Client + Server		MIT	https://github.com/1248/microcoap
microCoAPy	MicroPython	RFC 7252	Client + Server	Core	Apache License 2.0	https://github.com/insighio/microCoAPy
nanocoap	C	RFC 7252	Client + Server	Core, Blockwise Transfers	LGPL	https://api.riot-os.org/group__net__nanocoap.html

Name	Programming Language	Implemented CoAP version	Client/Server	Implemented CoAP features	License	Link
nCoap	Java	RFC 7252	Client + Server	Observe, Blockwise Transfers, CoRE Link Format, Endpoint-ID-Draft	BSD	https://github.com/okleine/nCoAP
node-coap	Javascript	RFC 7252, RFC 7641, RFC 7959	Client + Server	Core, Observe, Block	MIT	https://github.com/mcollina/node-coap
Ruby coap	Ruby	RFC 7252	Client + Server (david)	Core, Observe, Block, RD	MIT, GPL	https://github.com/nning/coap https://github.com/nning/david
Sensinode C Device Library	C	RFC 7252	Client + Server	Core, Observe, Block, RD	Commercial	https://silver.arm.com/browse/SEN00
Sensinode Java Device Library	Java SE	RFC 7252	Client + Server	Core, Observe, Block, RD	Commercial	https://silver.arm.com/browse/SEN00
Sensinode NanoService Platform	Java SE	RFC 7252	Cloud Server	Core, Observe, Block, RD	Commercial	https://silver.arm.com/browse/SEN00
SwiftCoAP	Swift	RFC 7252	Client + Server	Core, Observe, Blockwise Transfers	MIT	https://github.com/stuffrabbit/SwiftCoAP
TinyOS CoapBlip	nesC/C	coap-13	Client + Server	Observe, Blockwise Transfers	BSD	https://web.archive.org/web/20130312140509/http://docs.tinyos.net/tinywiki/index.php/CoAP
txThings	Python (Twisted)	RFC 7252	Client + Server	Blockwise Transfers, Observe (partial)	MIT	https://github.com/mwasilak/txThings/
coap-rs	Rust	RFC 7252	Client + Server	Core, Multicast, Observe option, *Too Many Requests* Response Code	MIT	https://github.com/Covertness/coap-rs https://docs.rs/coap/

Name	Programming Language	Implemented CoAP version	Client/Server	Implemented CoAP features	License	Link
YaCoAP	C				MIT	https://github.com/RIOT-Makers/YaCoAP

Proxy implementations [edit]

- Squid 3.1.9 with transparent HTTP-CoAP mapping module
- jcoap Proxy
- Californium cf-proxy2
- CoAPthon
- FreeCoAP

CoAP group communication [edit]

In many CoAP application domains it is essential to have the ability to address several CoAP resources as a group, instead of addressing each resource individually (e.g. to turn on all the CoAP-enabled lights in a room with a single CoAP request triggered by toggling the light switch). To address this need, the IETF has developed an optional extension for CoAP in the form of an experimental RFC: Group Communication for CoAP - RFC 7390[3] This extension relies on IP multicast to deliver the CoAP request to all group members. The use of multicast has certain benefits such as reducing the number of packets needed to deliver the request to the members. However, multicast also has its limitations such as poor reliability and being cache-unfriendly. An alternative method for CoAP group communication that uses unicasts instead of multicasts relies on having an intermediary where the groups are created. Clients send their group requests to the intermediary, which in turn sends individual unicast requests to the group members, collects the replies from them, and sends back an aggregated reply to the client.[4]

Security [edit]

CoAP defines four security modes[5]

- NoSec, where DTLS is disabled
- PreSharedKey, where DTLS is enabled, there is a list of pre-shared keys, and each key includes a list of which nodes it can be used to communicate with. Devices must support the AES cipher suite.
- RawPublicKey, where DTLS is enabled and the device uses an asymmetric key pair without a certificate, which is validated out of band. Devices must support the AES cipher suite and Elliptic Curve algorithms for key exchange.
- Certificate, where DTLS is enabled and the device uses X.509 certificates for validation.

Research has been conducted on optimizing DTLS by implementing security associates as CoAP resources rather than using DTLS as a security wrapper for CoAP traffic. This research has indicated that improvements of up to 6.5 times none optimized implementations. [6]

In addition to DTLS, RFC8613[7] defines the Object Security for Constrained RESTful Environments (OSCORE) protocol which provides security for CoAP at the application layer.

Security issues [edit]

Although the protocol standard includes provisions for mitigating the threat of DDoS amplification attacks,[8] these provisions are not implemented in practice,[9] resulting in the presence of over 580,000 targets primarily located in China and attacks up to 320Gbps.[10]

See also [edit]

- Internet of Things
- OMA Lightweight M2M
- Web of Things
- Static Context Header Compression (SCHC)

References [edit]

1. ^ RFC 7252, Constrained Application Protocol (CoAP)
2. ^ "Integrating Wireless Sensor Networks with the Web" , Walter, Colitti 2011
3. ^ RFC 7390, Group Communication for CoAP
4. ^ "Flexible Unicast-Based Group Communication for CoAP-Enabled Devices" , Ishaq, I.; Hoebeke, J.; Van den Abeele, F.; Rossey, J.; Moerman, I.; Demeester, P. Sensors 2014
5. ^ RFC 7252, Constrained Application Protocol (CoAP)
6. ^ Capossele, Angelo; Cervo, Valerio; De Cicco, Gianluca; Petrioli, Chiara (June 2015). "Security as a CoAP resource: An optimized DTLS implementation for the IoT". *IEEE*: 529–554. doi:10.1109/ICC.2015.7248379.
7. ^ Palombini, Francesca; Seitz, Ludwig; Selander, Goeran; Mattsson, John. "Object Security for Constrained RESTful Environments (OSCORE)". *tools.ietf.org*. Retrieved 2021-05-07.
8. ^ "TLS 1.3 is going to save us all, and other reasons why IoT is still insecure", Dani Grant, 2017-12-24
9. ^ "When Machines Can't Talk: Security and Privacy Issues of Machine-to-Machine Data Protocols", Federico Maggi and Rainer Vosseler, 2018-12-06
10. ^ "The CoAP protocol is the next big thing for DDoS attacks", Catalin Cimpanu, 2018-12-05

External links [edit]

- RFC 7252 "The Constrained Application Protocol (CoAP)"
- coap.me – CoAP test server run by University of Bremen

Wikimedia Commons has media related to ***SSL and TLS***.

Categories: Hypertext Transfer Protocol | Application layer protocols | Internet of things