7. RFC 7252：The Constrained Application Protocol (CoAP)
RFC 7252：受限制的應用協定 (CoAP)

網際網路工程任務組 (IETF)
Request for Comments：7252
類別：標準跟蹤
ISSN：2070-1721

Z.Shelby
ARM
K.Hartke
C.Bormann
Universitaet Bremen TZI

2014 年 6 月

受限制的應用協定 (CoAP)

摘要

受限制的應用協定(CoAP)是一種專門的 web 傳輸協定，用於受限節點和受限制的網路(例如：低功耗、有損)。節點通常具有 8 位元的微控制器，具有少量 ROM 和 RAM，而低功耗無線的 IPv6 等則受限網路個人區域網路(6LoWPAN)常常有高資料封包錯誤率和典型的 10 kbit/s 的網路輸送量。該協定專為機器對機器(M2M)應用而設計，如智慧能源和建築自動化。

CoAP 提供應用程式端點之間的請求/回應交互模型,支援內置的服務和資源被發現，並包括 Web 的關鍵概念(如：URI 和 Internet 的媒體類型)。CoAP 設計的目的是為了方便與 HTTP 進行分界，以便與 Web 進行集成，同時滿足特定的需求，如廣播支持、非常低的開銷和簡單的受限環境。

本文的狀態

這是一個網際網路標準跟蹤文件。
本文是網際網路工程任務組 (IETF) 的產品。它代表了 IETF 團體的共識。該報告已獲公眾審閱，並已獲網際網路工程督導小組(IESG)批准出版。有關網際網路標準的更多資訊，請參閱 RFC 5741 第 2 條。

有關此文件的目前狀態、任何錯誤以及如何提供有關文件的回饋資訊，請訪問 http://www.rfc-editor.org/info/rfc7252.

版權聲明

目錄

1. 前言 7
1.1. 特點 7
1.2. 術語 9
2. 受限制的應用協定 13
2.1. 消息傳遞模型 13
2.2. 請求/回應模型 15
2.3. 仲介和緩存 17
2.4. 資源發現 17
3. 訊息格式 17
3.1. 選擇格式 19
3.2. 選項值的格式 21
4. 信息傳輸 22
4.1. 消息和端點 23
4.2. 可靠的資訊傳輸 23
4.3. 不可靠地發送消息 25
4.4. 消息之間的關聯 26
4.5. 去除重複消息 26
4.6. 消息大小 27
4.7. 擁塞控制 28
4.8. 傳輸參數 29
4.8.1. 改變參數 29
4.8.2. 傳輸參數的衍生時間 30
5. 請求/回應語義 32
5.1. 請求 33
5.2. 回應 33
5.2.1. 附帶回應 34
5.2.2. 單獨回應 35
5.2.3. 無需確認消息 36
5.3. 請求/回應的匹配 36
5.3.1. 權杖 36
5.3.2. 請求/回應匹配規則 37
5.4. 選項 38
5.4.1. 重要選項/非重要選項 39
5.4.2. 代理不安全或安全跟無快存金鑰 39
5.4.3. 長度 40

5.4.4. 默認值 40
5.4.5. 可重複選項 41
5.4.6. 選項編號 41
5.5. 有效負載和表示法 42
5.5.1. 表現 42
5.5.2. 診斷式 有效負載 42
5.5.3. 經由選擇的表現 43
5.5.4. 內容協商 43
5.6. 緩存 43
5.6.1. 新鮮度模型 44
5.6.2. 校驗模型 45
5.7. 代理 45
5.7.1. 代理操作 46
5.7.2. 正向代理 47
5.7.3. 反向代理 48
5.8. 方法定義 48
5.8.1. GET 49
5.8.2. POST 49
5.8.3. PUT 49
5.8.4. DELETE 50
5.9. 返回碼的定義 50
5.9.1. 成功2.xx 50
5.9.2. 客戶端錯誤4.xx 51
5.9.3. Server Error 5.xx 53
5.10. Option的定義 54
5.10.1. Uri-Host, Uri-Port, Uri-Path, and Uri-Query 55
5.10.2. Proxy-Uri and Proxy-Scheme 56
5.10.3. Content-Format 57
5.10.4. Accept 57
5.10.5. Max-Age 57
5.10.6. Tag 57
5.10.7. Location-Path and Location-Query 59
5.10.8. 條件請求選項 59
5.10.9. Size1選項 61
6. CoAP URIs 61
6.1. coap URI Scheme 61
6.2. coaps URI Scheme 62
6.2.1. 標準化和比較規則 63

6.3. 將URI解碼為選項 63
6.4. 將選項編碼成URI。 64
7. 發現 66
7.1. 服務發現 66
7.2. 資源發現 66
7.2.1. 'ct'特性 67
8. 多播CoAP 67
8.1. 消息層 67
8.2. 請求/回應層 68
8.2.1. Caching 69
8.2.2. 代理 69
9. 安全CoAP 70
9.1. DTLS-Secured CoAP 71
9.1.1. 消息層 72
9.1.2. 請求/回應層 73
9.1.3. 端點身份 73
10. CoAP和HTTP的跨協定代理 76
10.1. CoAP-HTTP代理 77
10.1.1. GET 78
10.1.2. PUT 79
10.1.3. DELETE 79
10.1.4. POST 79
10.2. HTTP-CoAP代理 79
10.2.1. OPTIONS and TRACE 80
10.2.2. GET 80
10.2.3. HEAD 81
10.2.4. POST 81
10.2.5. PUT 81
10.2.6. DELETE 82
10.2.7. CONNECT 82
11. 安全事項 82
11.1. 解析協定和處理URIs 82
11.2. 代理和緩存 83
11.3. 增幅的風險 83
11.4. 地址欺騙攻擊 84
11.5. 跨協定攻擊 86
11.6. 受限節點的注意事項 87
12. 網際網路地址分配注意事項 88

12.1. CoAP代碼註冊 88
12.1.1. 方法碼 89
12.1.2. 回應碼 89
12.2. CoAP 選項號碼登錄 91
12.3. CoAP內容格式登錄 93
12.4. URI方案登記 95
12.5. 安全URI規範註冊表 96
12.6. 安全服務名稱和埠號表 97
12.7. 保全服務名稱與埠號登記 97
12.8. 多播地址表 98
13. 致謝 99
14. 引用文件 99
14.1. 規範性引用文件 99
14.2. 參考目錄 102
附錄 A. 範例 106
附錄 B. URI範例 113

1. 前言

web 服務(web api)在網路上的使用已經在大多數應用程式中變得無處不在，並且依賴於 web 的基本表示狀態傳輸(Representational State Transfer, REST)。

在受限的 RESTful 環境(CoRE)上的工作目的在於，以一種適用於最受限節點(例如，具有有限 RAM 和 ROM 的 8 位元微控制器)和網路(例如，6LoWPAN，[RFC4944])的形式實現 REST 架構。受限制的網路，如 6LoWPAN 支援將 IPv6 封包分割成小鏈結層；然而，這導致資料封包輸送機率大幅降低。CoAP 的其中一個設計目標便是保持整體消息小，進而限制碎片化的需求。

CoAP 的主要目標之一是為這種受限環境的特殊需求設計通用的 web 協定，特別是考慮到能源、建築自動化和其他機器對機器(M2M)應用程式。CoAP 的目標不是盲目地壓縮 HTTP [RFC2616]，而是實現與 HTTP 通用的 REST 子集，但針對 M2M 的應用程式進行優化。儘管 CoAP 可以用於將簡單的 HTTP 介面重新設計成更緊湊的協定，但更重要的是，它還為 M2M 提供了一些特性，比如內置的開發、多播支援和非同步消息交換。

本文具體說明瞭受限的應用程式協定(CoAP)，它可以很容易地轉換為 HTTP，以便與現有 Web 集成，同時滿足特定的需求，如多播支援、非常低的開銷、簡單的受限環境和 M2M 應用程式的簡單性。

1.1. 特點

CoAP 具有以下特性:

- Web 協定在受限環境中可滿足 M2M 需求。
- UDP [RFC0768]與可選擇的信度綁定，支持單播和多播請求。
- 非同步消息交換。

- 低標頭 overhead 和解析複雜度。
- 支援 URI 與 HTTP 內容類型。
- 簡單的代理和緩存功能。
- 無狀態 HTTP 映射時，允許構建代理，以統一的方式通過 HTTP 訪問 CoAP 資源，或者通過 CoAP 實現 HTTP 的簡單介面。
- 安全綁定到資料封包傳輸層安全(DTLS) [RFC6347]。

1.2. 術語

在此文件中，當以下關鍵詞以大寫字母出現時，將按照[RFC2119]中的描述解釋，關鍵字包含以下"必須(MUST)"，"不得(MUST NOT)"，"必要(REQUIRED)"，"必須(SHALL)"，"不得(SHALL NOT)"，"應該(SHOULD)"，"不應該(SHOULD NOT)"，"建議(RECOMMENDED)"，"不建議(RECOMMENDED)"，"可以(MAY)"，"可選(OPTIONAL)"這些詞也可能用小寫字母出現在本檔中，此時並不適用於以上規範。

他具體要求讀者熟悉[RFC2616]中討論的所有術語和概念，包括"resource", "representation", "cache"和"fresh"。(在 HTTP RFCs 更新集 RFC 7230 到 RFC 7235 之前已經完成，本規範特別引用了以前的版本──RFC2616) 此外，本規範定義了以下術語：

端點
: 一個參與 CoAP 協定的實體。通俗的說，端點位於「節點」上，儘管「主機」更符合 Internet 標準的使用，並且可以通過傳輸層多工資訊進一步標識端點，這些資訊可以包括 UDP 埠號和安全關聯。（章節 4.1）

收訊
: 消息的初始端點。當關注特定發送方的標識方面時，也要關注「來源端點」。

接收方
: 訊息的目標端點。當關注特定接收方的標識方面時，也就是「目標端點」。

客戶端
: 請求的初始端點；回應的目標端點。

伺服器
: 請求的目標端點；回應的初始端點。

來源伺服器

給特定資源停留或創建的伺服器。

仲介

CoAP 端點，它同時作為伺服器和用戶端來源伺服器(可能通過進一步的仲介)進行操作。代理是仲介的一種常見形式；在本規範中討論了這類代理中的幾個類別。

代理

仲介主要負責轉發請求和轉發迴回應，可能在過程中執行緩存、命名空間轉換或協定轉換。與一般意義上的仲介相反，代理通常不實現特定的應用程式語義。根據請求轉發在整個結構中的位置，有兩種常見的代理形式：正向代理和反向代理。在某些情況下，單個端點可能充當原始伺服器、正向代理或反向代理，根據每個請求的性質切換。

正向代理

用戶端選擇的端點，通常通過本地配置規則，代表用戶端執行請求，執行所有必要的轉換。有些轉換是極小的，例如對於"coap" URIs 的代理請求，而其他請求可能需要與完全不同的應用層協定進行轉換。

反向代理

代表一個或多個其他伺服器並且這些伺服器滿足請求的端點，執行任何必要的轉換。與正向代理不同，用戶端可能不知道它正在與反向代理通信；反向代理接收請求，就好像它是目標資源的來源伺服器一般。

CoAP-to-CoAP 代理

對比 cross-proxy，CoAP-to-CoAP 代理是從 CoAP 請求映射到 CoAP 請求的代理，即是在伺服器端和用戶端都使用 CoAP 協定。

跨域代理伺服器

跨域代理伺服器，或簡稱跨域代理，是在不同協定之間轉換的代理，如 CoAP-to-HTTP 代理或 HTTP-to-CoAP 代理。雖然該規範對 coap 到 coap 代理提出了非常具體的要求，但是在跨代理中可能有更多的變化。

可確認的訊息

有些消息需要確認。這些消息稱為「可確認的」。當沒有丟失任何封包時，每個可確認的消息都會引發一個確認或重置的消息回復。

無法確認的訊息

其他一些消息不需要被確認。對於應用程式需求經常重複的消息，例如來自感測器的重複讀數尤其如此。

ACK 消息

ACK 消息用於確認某個可靠消息已經到達。ACK 消息自身並不代表這個請求處理的結果是成功還是失敗。ACK 消息有可能會同時為附帶回應(Piggybacked Response)。

重置訊息

重置消息表示收到了特定的消息(可確認的或不可確認的)，但是缺少一些前後關係來正確處理它。這種情況通常是在接收節點重新引導且忘記解釋消息所需的狀態造成的。觸發重置消息(例如，通過發送一個空的可確認消息)也可以作為檢測端點活性的一種較便宜的方法("CoAP ping")。

附帶回應

在 CoAP 確認(ACK)消息中包含一個附帶回應，該消息被發送以確認接收到此回應的請求(章節 5.2.1)。

單獨回應

當帶有請求的可確認消息以空消息被確認時(例如，因為伺服器無法立即得到答案)，將在單獨的消息交換中發送單獨的回應(章節 5.2.2)

空消息

代碼為 0.00 的消息，既不是請求也不是回應。空消息只包含 4-byte 的開頭。

關鍵選項

為正確處理消息，最終接收消息的端點需要理解的選項(章節5.4.1)。請注意，正如「選項」的名稱所暗示的，關鍵選項的實現通常是可以選擇的；不受支援的關鍵選項會導致錯誤回應或消息摘要被拒絕。

選擇性選項

一個被不理解它的端點所忽略的選項。即使不理解該選項也可以處理消息(章節 5.4.1)。

不安全的選項

為了安全的轉發消息，接收消息的代理需要理解此選項(章節5.4.2)。然而，並不是每一個關鍵的選擇都是不安全的。

Safe-to-Forward 選項

一種被不理解它的代理轉發的安全選項。即使在不理解該選項的情況下轉發消息也是可以接受的(章節 5.4.2)。

資源探索

CoAP 用戶端向伺服器查詢其託管資源列表(即，章節 7 定義的連結)。

內容格式

網路媒體類型(可能具有給定的特定參數)和內容編碼(通常是標識內容編碼)的組合，由「CoAP 內容格式」註冊表定義的數位識別碼符標識。當關注的焦點不是數位識別碼符，而是資源表示的這些特徵的組合時，這也稱為「表示格式」。

約束節點和約束節點網路的其他術語可以在[RFC7228]中找到。

在本規範中，術語「byte」作為「八隅體」的同義使用，其現在的習慣意義是「位元組」。

該協定中的所有多位元組整數都按網路位元組順序解釋。

在使用算術的地方，該規範使用程式設計語言 C 中熟悉的符號，除了運運算元「**」代表指數。

2. 受限制的應用協定

CoAP 的交互模型類似於 HTTP 的客戶端/服務器模型，然而機器對機器的交互通常會導致 CoAP 在客戶端/服務器中起作用。一個 CoAP 的請求相當於一個 HTTP 的請求，由客戶端送出，請求在服務器上的資源(由 URI 鑑定) 執行操作(使用方法代碼)。然後服務器發送帶有回應代碼的回應,這個回應可以包含了資源代表。

與 HTTP 不同，CoAP 透過面向資料封包的傳輸(如 UDP) 異步處理這些交換。這是一個在邏輯上使用支援可選可靠得消息層(有指數回退)完成的。CoAP 定義四種消息類型：可證實的、不可證實的、確認、重置。其中一些消息裡包含了方法代碼和回應代碼使它們攜帶請求或回應。這些四種類型消息的基本交換對於請求/回應交互有些正交；請求可以攜帶在可證實和不可證實的消息，且回應也可以被攜帶在其中，以及可以在確認消息中捎帶。

可以將 CoAP 邏輯上使用兩層方法，CoAP 的消息傳遞層用於處理 UDP 和交互的異動性質和使用方法和回應代碼的請求/回應互動(見圖 1)。然而 CoAP 是單一協定，CoAP 標題只含有消息傳遞和請求/回應特徵。

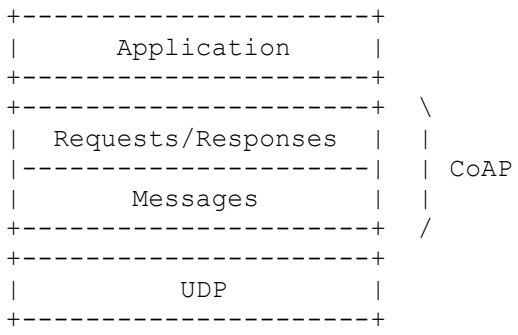


圖 1：CoAP 的抽象分層

2.1. 消息傳遞模型

CoAP 消息傳遞模型是基於 UDP 訊息上終端間的變化。

CoAP 使用短的固定-長度二進制標題(4 位元)，後面可以接緊湊的二進制選項和有效負荷。這個訊息格式是由要求和回應共用。這個 CoAP 訊息格是在第三節有詳細說明。每個訊息包含訊息 ID 用

於檢測重複和給予可選的可靠性。(訊息 ID 是緊湊的；它 16 位元大小每秒最多能允許 250 條訊息和預設的參數從一個終端到另外一個)

通過把消息標記為 CON 的，可以保障消息傳輸的可靠性。如圖 2 所示，在收到一個 CON 消息之後，接收端會發送一個帶有相同消息 ID(Message ID)（在這個例子中是 0x7d34）的 ACK。如果在默認的超時時間之後沒有收到帶有相同消息 ID 的 ACK，那麼它將會被重傳，如果仍然沒有收到 ACK，此後重傳超時時間會以指數級遞增。當接收端無法處理一個 CON 消息(也無法返回一個正常的錯誤回應)時，它將會回應一個 RST 消息，而不是 ACK。

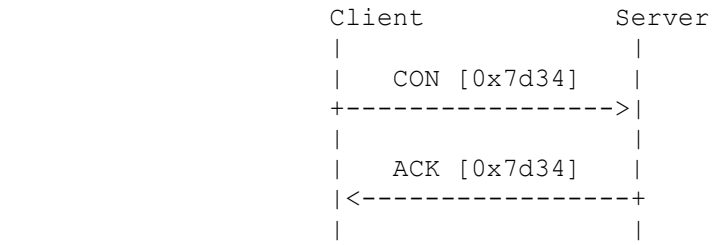


圖 2：可信賴的消息傳輸

訊息不需要可信賴訊息傳輸(例如，每個單一測量遠離傳感器數據流)，可以被當成非證實訊息發送(NON)。這些都不被承認,但仍然有訊息 ID 被發現完全一樣(在這個例子，(0x01a0)；看圖三。當接收器不能處理無法確認消息，它也許會回復重置訊息(RST)。

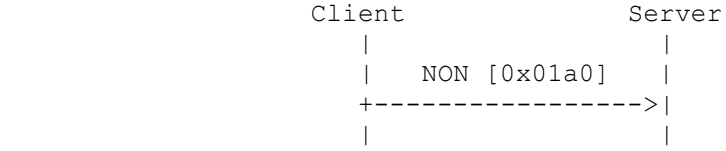


圖 3: 不可靠的消息傳輸，在章節 4 中查看 CoAP 訊息細節.

由於 CoAP 在 UDP 運行，它也支援使用多重播送 IP 目的地址，啟用多播 CoAP 請求。章節 8 討論多播位址正確使用 CoAP 訊息和避免回應擁擠的預防措施。

在章節 9 中給 CoAP 定義幾個安全模式不等從無安全到基於證書的安全性。這些文獻指定綁定 DTLS 以保護協定；CoAP 的網際網路安全協定的使用被討論在(網際網路安全協定-CoAP)

2.2. 請求/回應模型

CoAP 請求和回應語義學在 CoAP 訊息中運行，其分別包含方法代碼或回應代碼。可選的(或默認的)請求和回應資訊，像是 URI 和有效載荷媒體類型作為 CoAP 選項運行。單辭序列習慣從基礎訊息獨立回應匹配的請求(章節 5.3)。重點注意單詞序列是訊息 ID 的分開的概念。

請求運行在可證實(CON)或不可證實(NON)消息中，且如果立即啟用，對請求的回應攜帶在可證實訊息中運行產生確認(ACK)消息。這稱為搭載回應，細節在章節 5.2.1。沒有需要個別確認搭載回應，如果攜帶著搭載回應的確認訊息遺失，客戶端會再重新發送請求。兩個例子對於基本 GET 搭載回應的請求顯示在圖 4，一個成功，一個造成 404(not found)回應。

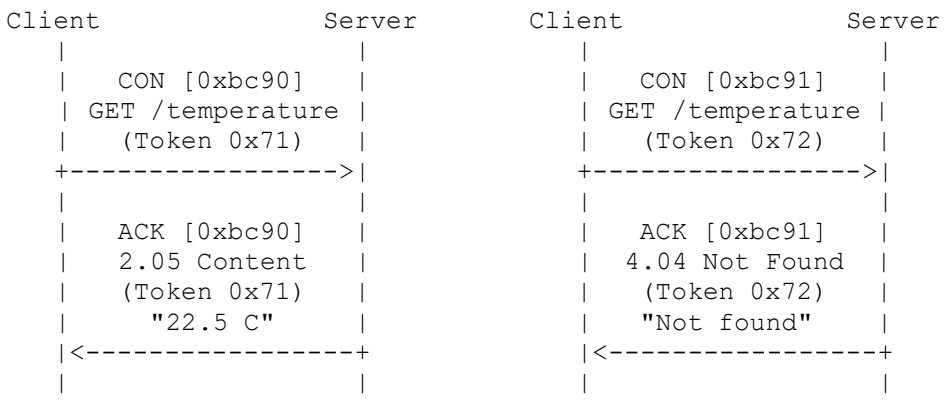


圖 4：兩個帶有捎帶回應的 GET 請求

如果伺服器無法立即回應運行在確認訊行的請求，簡單的回應空的致謝訊息，這樣客戶端可以停止再發送請求。當回應準備好，伺服器在新的可證實訊息中發送回應(然後反過來需要客戶端的確認)。這個稱為"單獨回應"，如圖 5 所示並在章節 5.2.2 中有更詳細的描述。

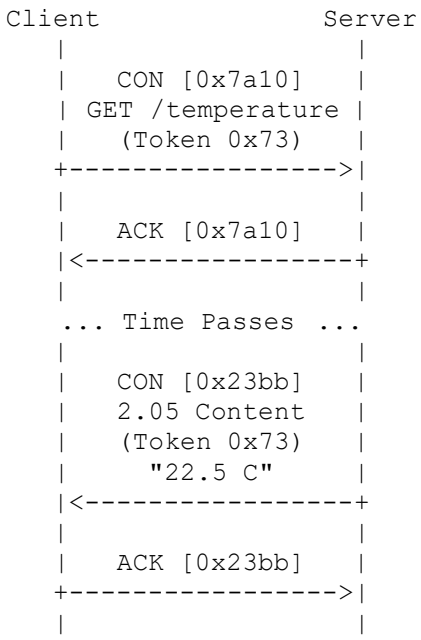


圖 5:具有單獨回應的 GET 請求

如果在無法確認訊息中送出請求，然後使用新的無法確認訊息回應。雖然伺服器可能會帶送出可確認訊息。這些類型的交換在圖 6 說明。

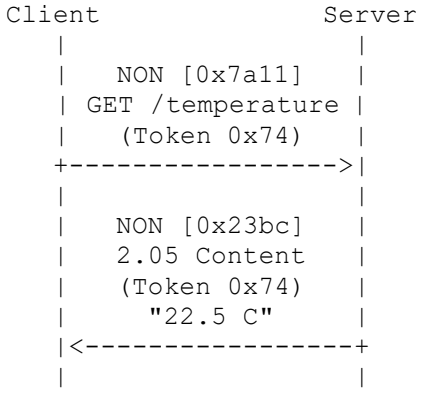


圖 6：在無法確認消息中攜帶的請求和回應

CoAP 以和 HTTP 相似方法利用 GET、PUT、POST 跟消除方法，在章節 5.8 中的語義規定(注意 CoAP 方法的詳細語義學是"幾乎，但不是完全不同"[HHGTTG]那些 HTTP 方法:直覺採取來自 HTTP 經驗通常應用良好，但那些不同足夠讓它值得去實際閱讀現行規範。

基本四的方法外可以被在個別規格中被添加在 CoAP。新方法不一定需要使用成對的請求&回應。即使現有方法，單獨請求可能產生多重反映，例如，用於多播請求（章節 8）或觀察選項 [OBSERVE]。

URI 支持在伺服器內被簡化，因為客戶端早已解析 URI 且切割為主機，埠，路徑和查詢組件，利用內定質提升效率。回應代碼和小部分 HTTP 代碼狀態有關，其中添加一些 CoAP 特定代碼，如章節 5.9 中所定義。

2.3. 仲介和緩存

該協定支持緩存回應為了有效率的履行請求。使用 CoAP 回應攜帶最新且有效訊息授權簡單的緩存。緩存可以位於終端或仲介。緩存功能的規定在章節 5.6。

基於很多原因,代理權在約束網路是有益得,包括限制網路流量，增加效能，訪問睡眠設備的資源，和出於安全性的考量。協定支持代表其他 CoAP 終端的請求代理。當使用代理，要請求的資源 URI 包含在請求裡，當目的地 IP 位元址設置代理位元址，在章節 5.7 觀看代理功能的更多資訊。

由於 COAP 根據[REST]架構設計，因此展示那些相似 http 協定功能。從 CoAP 映射到 HTTP 是相當直接的，反之亦同。這種映射可用於實現 HTTP REST 介面使用 CoAP 或在 HTTP 跟 CoAP 之間的轉換。這些轉換可以被跨協定(「交叉代理」)代理運行，其轉換方法或回應代碼、媒體類型，以及相應 HTTP 功能的選項。章節 10 提供更多關於 HTTP 映射的細節。

2.4. 資源發現

資源發現對機器之間的交互是非常重要的且如章節7支討論使用CoRE鏈接格式[RFC6690]來支持。

3. 訊息格式

CoAP 基於壓縮消息的交換,預設情況下,這些壓縮消息通過 UDP 傳輸(即，每個 CoAP 消息佔用一個 UDP 資料封包的資料部分)。CoAP 還可以用於資料封包傳輸層安全(DTLS)(參見章節 9.1)。它還可以用於其他傳輸，如 SMS、TCP 或 SCTP，但這些傳輸的規範超出了本文的範圍。(CoAP 不支持 UDP-lite [RFC3828]和 UDP zero checksum [RFC6936]。)

CoAP 消息以簡單的二進位格式編碼。訊息格式以一個固定大小的 4 位元組頭開始。然後是一個可變長度的單詞序列值，其長度可以在 0 到 8 位元組之間。

在單詞序列值後面是一個零或多個類型長度值(TLV)格式的 CoAP 選項序列，後面還可以選擇一個負載，該負載將佔用 datagram 的其餘部分。

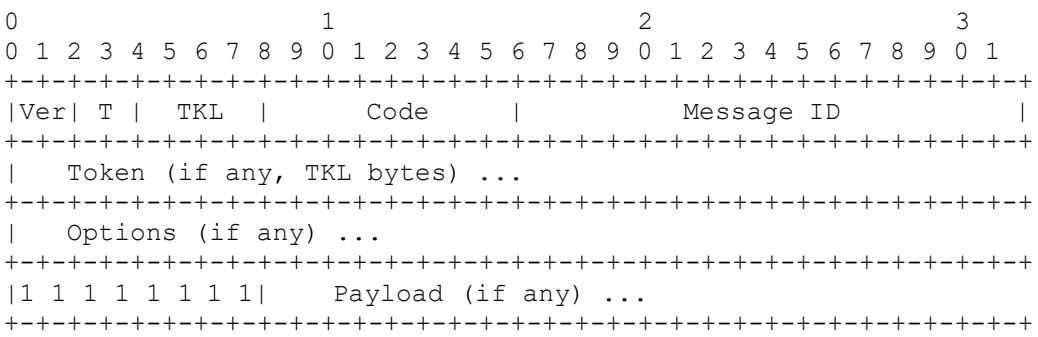


圖 7：訊息格式

header 中的欄位定義如下:

版本(Ver): 2 位元不帶正負號的整數。指示 CoAP 版本號。此規範的實現必須將此欄位設置為 1(01 二進位)。其他值保留給將來的版本。必須忽略版本號未知的消息。

類型(T): 2 位元不帶正負號的整數。指示此消息的類型是否為可驗證(0)、不可驗證(1)、確認(2)或重置(3)。這些消息類型的語義在第 4 節中定義。

單詞序列長度(TKL):4 位元不帶正負號的整數。指示可變單詞序列欄位的長度(0-8 位元組)。保留長度 9-15，不能發送，必須作為消息格式錯誤處理。

代碼: 8 位元不帶正負號的整數，分為 3 位元類別(最重要的位元)和 5 位元詳細資訊(最不重要的位元)，文件為"c.dd"。其中"c"是 3 位子欄位從 0 到 7 的一位元數字，"dd"是 5 位子欄位從 00 到 31 的兩位元數字。該類別可以指示為一個請求(0)、一個成功回應(2)、一個客戶機錯誤回應(4)或一個伺服器錯誤回應(5)。作為一種特殊情況，代碼 0.00 表示一條空訊息。在請求的情況下，Code 欄位表示請求方法；對於回應，則使用回應代碼。可能的值保存在 CoAP 代碼註冊表中(章節 12.1)。請求和回應的語義在第 5 節中定義。

訊息 ID: 按網路位元順序排列的 16 位元不帶正負號的整數。用於檢測消息重複，並將確認/重置類型的消息與可確認/不可確認類型的消息匹配。第 4 節中定義了生成訊息 ID 和匹配消息的規則。

頭後面是單詞序列值，它可以是 0 到 8 位元組，由單詞序列長度欄位給予。單詞序列值用於關聯請求和回應。第 5.3.1 節定義了生成單詞序列值和關聯請求和回應的規則。

頭和單詞序列後面跟著 0 個或多個選項(第 3.1 節)。一個選項之後可以是消息的末尾、另一個選項或有效負載標記和有效負載。

在頭、單詞序列和選項(如果有的話)後面是可選擇的有效負載。如果存在且長度非零，則用一個固定的單字節有效負載標記(0xFF)作為首碼，該標記指示選項的結束和有效負載的開始。有效載荷資料從標記符後擴充到 UDP 資料封包的末尾，即，有效載荷長度由資料封包大小計算。沒有有效載荷標記表示零長度的有效載荷。必須將後跟零長度有效負載的標記的存在作為消息格式錯誤處理。
執行計劃:位元組值 0xFF 也可能出現在一個選項長度或值中，因此簡單的位元組掃描 0xFF 不是查找有效負載標記的可行技術。位元組 0xFF 只有在另一個選項開始時才具有有效負載標記的含義。

3.1. 選擇格式

CoAP 定義了一些可以包含在消息中的選項。消息中的每個選項實例都指定定義的 CoAP 選項的選項號碼、選項值的長度和選項值本身。

不必直接指定選項號碼，實際上必須按它們的選項號的順序出現，並且在它們之間使用增量編碼:每個實例的選項號計算為其增量和消息中前一個實例的選項號的和。對於消息中的第一個實例，假定前面的選項實例的選項號碼為零。可以使用 0 作為增量來包含相同選項的多個實例。

選項號保存在「CoAP 選項號」註冊表中(第 12.2 節)。有關本文中定義的選項的語義，請參見第 5.4 節。

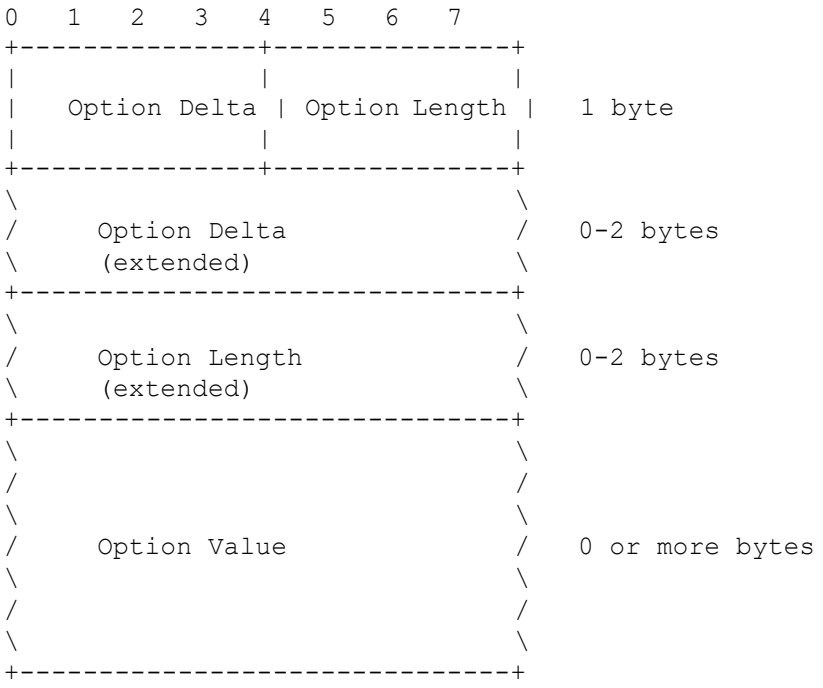


圖 8: 選擇格式

選項中的欄位定義如下:

選項 Delta: 4 位不帶正負號的整數。0 到 12 之間的值表示選項 Delta。為特殊構造保留三個值:

13:一個 8 位元不帶正負號的整數跟在初始位元組後面，表示選項 Delta - 13。

14:網路位元組順序中的 16 位元不帶正負號的整數跟在初始位元組後面，表示選項 Delta - 269。

15:預留給有效載荷標記。如果欄位設置為該值，但整個位元組不是有效負載標記，則必須將其作為消息格式錯誤處理。

產生的選項增量用作此選項的選項數與前一個選項的選項數之間的差值(或第一個選項為零)。換句話說，通過簡單地將這個選項和之前所有選項的選項增量值相加，就可以計算出選項數。

選項長度:4 位不帶正負號的整數。0 到 12 之間的值表示選項值的長度，單位為位元組。為特殊構造而保留三個值:

13:一個 8 位元不帶正負號的整數在選項值之前，表示選項長度減去 13。

14:網路位元組順序中的一個 16 位元不帶正負號的整數位於選項值之前，表示選項長度減去 269。

15:預留作為將來使用。如果欄位設置為該值，則必須將其作為消息格式錯誤處理。

數值:精確的選項長度位元組序列。選項值的長度和格式取決於各自的選項，這些選項可以定義可變長度值。本檔使用的格式見第 3.2 節；在其他檔中定義的選項可以使用其他選項值格式。

3.2. 選項值的格式

本文中定義的選項使用以下選項值格式。

empty:一個零長度的位元組序列。

opaque: 不透明的位元組序列。

uint: 一個非負整數，使用選項長度欄位提供的位元組數以網路位元組順序表示。

選項定義可以指定允許的位元組數範圍；如果有選擇，發送方應該用盡可能最少的位元組表示整數，而不是前置字元為零位元組。例如，數位 0 用一個空的選項值(一個零長度的位元組序列)表示，數位 1 用一個數值為 1 的位元組表示(最重要的位元優先符號中的位元組合 00000001)。收件人必須準備處理以零位元組開頭的值。

執行計劃: 允許發送方的異常行為是為使用範本中固定大小選項的高度受限的範本實現(例如，硬體執行)設計的。

字串:使用 UTF-8 [RFC3629]以 Net-Unicode 形式編碼的 Unicode 字串[RFC5198]。

注意，這裡以及 CoAP 協定中使用 UTF-8 編碼的所有其他地方，其目的是通過 CoAP 協定實現可以直接使用編碼後的字串，並將其作為不透明的位元組字串進行比較。
在 CoAP 實現中不存在任何期望，也不需要執行標準化(除非從 CoAP 協定之外的源導入未知的未標準化 Unicode 字串)。還要注意，ASCII 字串(不使用特殊控制字元)始終是有效的 UTF-8 Net-Unicode 字串。

4. 信息傳輸

CoAP 消息在 CoAP 端點之間非同步交換。它們用於傳輸 CoAP 請求和回應，其語義在第 5 節中定義。

由於 CoAP 綁定到 UDP 等不可靠的傳輸，CoAP 消息可能會出現順序錯誤、出現重複或丟失，而不需要通知。由於這個原因，CoAP 實現了羽量級的可靠性機制，而不需要重新創建 TCP 等傳輸的完整特性。它具有以下特點:

- 對於可確認的消息，具有指數級回退的簡單停等重傳可靠性。

- 確認和非確認消息的重複檢測。

4.1. 消息和端點

CoAP 端點是 CoAP 消息的源或目的地。端點的特定定義取決於用於 CoAP 的傳輸。對於本規範中定義的傳輸，端點的標識取決於使用的安全模式(參見第 9 節):沒有安全性，端點僅由 IP 位址和 UDP 埠號標識。對於其他安全模式，端點則由安全模式定義。

有不同類型的消息。消息的類型由 CoAP 領頭的類型欄位指定。

與消息類型分離後，消息可以攜帶請求、回應或為空。這由 CoAP 頭中的請求/回應代碼欄位發出信號，並且與請求/回應模型相關。欄位的可能值保存在 CoAP 代碼註冊表中(章節 12.1)。

空消息的代碼欄位元元設置為 0.00。單詞序列長度欄位必須設置為 0，並且在消息 ID 欄位元元之後不能出現資料位元組。如果有位元組，則必須將其作為消息格式錯誤處理。

4.2. 可靠的資訊傳輸

通過在 CoAP 領頭中將消息標記為可確認的，可以啟動訊息的可靠傳輸。可驗證訊息總是攜帶一個請求或回應，除非它僅用於引發重置消息，在這種情況下，它是空的。收件人必須(A)承認可證實的消息和一個確認消息或(b)拒絕消息如果接受者缺乏上下文正確地處理消息，包括消息為空的情況下，使用一個代碼保留類(1、6 或 7)，或一個消息格式錯誤。拒絕確認消息的方法是發送匹配的重置消息，否則忽略該消息。確認消息必須回應可確認消息的消息 ID，並且必須帶有回應或為空(參見 5.2.1、5.2.2 節)。重置消息必須回應可確認消息的消息 ID，且必須為空。拒絕確認或重置消息(包括確認包含一個請求或帶有保留類的代碼，或者重置消息不是空的情況)是通過靜默地忽略它來實現的。更一般地說，確認和重置消息的接收方不能使用確認或重置消息進行回應。

發送方以指數增長的間隔重新傳輸可確認消息，直到接收到確認消息(或重置消息)或嘗試次數耗盡為止。

重傳由超時時間和重傳計數控制。對於每一個 CON 類型的消息，發送端必須一直維護超時時間和重傳計數,直到收到對應的 ACK 或者 RST。對於一個新的 CON 消息，初始的超時時間被設置為介於 ACK_TIMEOUT 和 ACK_TIMEOUT*ACK_RANDOM_FACTOR)之間（參見 4.8 節）的隨機值（通常不是整數秒），重傳計數被設置為 0。當超時發生，且重傳計數的值小於 MAX_RETRANSMIT，消息被重傳，重傳計數增加，超時時間變為原來的兩倍。如果在超時發生的時候重傳計數達到了 MAX_RETRANSMIT，或者收到了一個 RST 消息，那麼就會放棄消息的傳輸，由應用程式來處理這個傳輸失敗；如果在超時之前收到了 ACK，那麼傳輸就被認為成功了。

此規範對於用在實現上述二進位指數退避演算法的時鐘的精度沒有嚴格要求。特別的是，由於某個端點的睡眠時間安排，它可能會延遲某個特定的重傳，並且可能會趕上下一個端點。然而，在另一個重傳之前的最小間隔是 ACK_TIMEOUT，並且整個(重傳)序列必須保持在 MAX_TRANSMIT_SPAN 的信封中(參見章節 4.8.2)，即使這意味著發送方可能會錯過傳輸的機會。

發送可確認消息的 CoAP 端點可能會在到達 max_retransport 計數器值之前放棄嘗試獲取 ACK。例如，應用程式已經取消了請求，因為它不再需要回應,或者有其他跡象表明 CON 消息已經到達。特別是 CoAP 請求消息可能引發了單獨的回應，在這種情況下，請求者很清楚，只有 ACK 丟失了，重新傳輸請求沒有任何作用。然而，回應者不能反過來依賴於請求者的這種跨層行為。如果需要的話，即使請求者已經確認了一個可確認的回應，接收端也應該回復請求 ACK 的狀態。

另一個放棄重傳的原因可能是收到了 ICMP 錯誤。另一個放棄重傳的原因可能是收到了 ICMP 錯誤。如果希望處理 ICMP 錯誤以減輕潛在的 ICMP 欺騙攻擊的影響，那麼在實現上，應該仔細檢查產生 ICMP 消息的原始資料，包括埠號和 CoAP 頭部資訊，如 Message type，code，Message ID 和 Token；如果由於 UDP 提供

的介面 API 的限制而無法檢查，那麼就應該忽略 ICMP 錯誤。如果遵循了第 4.6 節中的"實現注意"，那麼正常情況下不應該發生資料封包過大錯誤(IPv4 的"fragmentation neede and DF set" [RFC0792])RFC4443]，因此應該被忽略。如果沒有遵循，那麼就應該進入一個路徑 MTU 計算演算法[RFC4821]。Source Quench 和 Time Exceeded 類型的 ICMP 錯誤應該被忽略。主機，網路，埠或協定不可達錯誤和參數錯誤有可能經過適當的審查後用於通知應用程式，消息發送失敗。

4.3. 不可靠地發送消息

有些消息不需要確認。尤其是對於應用程式需求經常重複的消息，例如對感測器的資料的重複讀取，並不需要每次都成功。

在沒有可靠性保障的情況下傳輸,可以把消息標記為 NON 類型，作為更加羽量級的選擇。非確認消息總是帶有請求或回應，且不能為空。NON 消息不能被收件人確認。如果消息缺少上下文來正確處理消息，包括消息為空、使用帶有保留類(1、6 或 7)的代碼或消息格式錯誤，則接收方必須拒絕該消息。拒絕不可確認的消息可能涉及發送匹配的重置消息，除重置消息外，必須默默地忽略被拒絕的消息。

在 CoAP 層，發送者沒有任何辦法得知 NON 類型的消息是否被接收端收到。發送者可能在 MAX_TRANSMIT_SPAN(在第 4.7 節的條款所限制，如果沒有收到回應，則使用 PROBING_RATE)之內發送多個副本，網路傳輸也有可能導致消息出現重複。為了使接收端能夠只處理一次，NON 消息也帶有 Message ID(和 CON 消息(可驗證消息)的 Message ID 共用數位池)。

總結第 4.2 和 4.3 節，四種類型的消息的使用如表 1 所示。"*"代表的是正常情況下不會這麼使用這個組合，除非為了引起一個 RST 消息(也就是"CoAPping")。

	CON	NON	ACK	RST
Request	X	X		
Response	X	X	X	X
Empty	*	-	X	X

表 1: 消息類型的使用

4.4. 消息之間的關聯

確認或重置消息通過消息 ID 和相應端點的附加位址資訊與 CON 消息(可確認消息)或 NON 消息(不可確認消息)關聯。MessageID 是一個 16 位元不帶正負號的整數，由可驗證或不可驗證消息的發送方生成，並包含在 CoAP 頭中。接收方必須在確認或重置消息中回顯消息 ID。

在 EXCHANGE_LIFETIME 時間之內，相同的 Message ID 必須不能重複使用(即與同一個端通信)(章節 4.8.2)。

實現注意：有許多策略可以用來生成 Message ID。最簡單的情況是，通過一個變數來生成 Message ID，每當發送了一個 CON 或 NON 消息，不論目標位址或埠是什麼，都改變這個變數。如果一個端需要處理大量的連接，那麼它也可以使用多個變數，例如為每個首碼或者目標位址使用一個變數。(注意，有些接收端可能無法區分發收到的資料封包是單播還是多播的，所以生成 Message ID 的端必須要確保這種情況下不會重複)。強烈建議這個變數的初始值是隨機的，這樣可以降低 off-path 攻擊成功的可能性。

為了使 ACK 或 RST 消息與 CON 或 NON 消息匹配，ACK(或 RST)消息的 Message ID 和源位元元址必須和 CON(或 NON)消息的 Message ID 和目的地址一致。

4.5. 去除重複消息

在 EXCHANGE_LIFETIME(章節 4.8.2)時間之內，當 ACK 消息丟失或者在第一個超時時間之前沒能到達原始服務端，接收端可能

收到多次重複的 CON 消息(由 Message ID 和源端位址標識)。接收端應該對每一次收到的重複消息都回以相同的 ACK 或 RST，但應該只處理一次。當 CON 消息傳輸的請求是冪等的時候見章節 5.1)，或者可以以冪等的方式來處理時，這一規則可以放寬。消息去除重複規則被放寬的例子如下：

- 服務端對於冪等的請求的每一次重傳都回以相同的回應（章節 4.2），這樣一來它就無需維護 Message ID 的狀態，在此情況下可以放寬規則。例如，如果重複處理的過程的開銷小於保留上一個回應的開銷，實現中可能把 GET,PUT,或 DELETE 請求的重傳當作獨立的請求來處理。

- 對於一些非冪等的請求，只要在應用層語義上這個取捨是值得的，一些資源受限的伺服器也可能會放寬規則。例如，如果一個 POST 請求對服務端的資料狀態的影響是很短暫的，那麼可能重複處理請求的開銷會小於保留上一次傳輸的相同請求處理狀態的開銷。

接收端可能在 NON_LIFETIME（章節 4.8.2）時間內收到重複的 NON 消息（由 Message ID 和源端位址標識）。接收端應該忽略掉重複的 NON 消息，只處理一次。這一規則根據應用程式的語義，有可能被放寬。

4.6. 消息大小

為了提高實現的品質，應該儘量使 CoAP 消息小到可以在一個鏈路層資料封包中傳輸（見第 1 章）。CoAP 文件本身只限制了消息大小的上限。大於 IP 資料封包大小的 CoAP 消息會造成分片。一個 CoAP 消息應該儘量包含在一個 IP 資料封包之內(即避免 IP 分片）並且在 UDP 包的有效負載之中。如果目的地址的 MTU 大小是未知的，那麼應該假定 IP 包的 MTU 大小為 1280 位元組。如果無法從頭部獲知消息大小，那麼應該設置消息最大為 1152 位元元元組，有效負載最大為 1024 位元組。

實現注意：CoAP 消息大小的選擇適用於 IPv6 和目前的大部分 IPv4 地址。（然而，對於 IPv4，很難保障絕對不發生 IP 分片。如果需要支援運行在受限網路上的 IPv4，那麼協定的實

現應該使用更為保守的IPv4資料封包大小,例如576位元組。按照[RFC0791]中所述，IPv4網路的MTU可以小到68位元組，減去用於安全開銷的位元組數，可用於UDP有效負載的就只剩下40位元組。如果要解決這個問題，也許應該設置IPv4的DF標誌位元，並且執行一些路徑MTU探測演算法[RFC4821]。然而在使用CoAP的一般場景中，沒有必要採用這些策略）在許多受限網路中，一個重要的資料分片發生在適配層（例如6LoWPAN L2資料封包最大只有127位元組，還包括了各種開銷在內）。這使得協定的實現應該盡可能減少資料封包大小,當消息大小達到3位元元元數的時候,應該使用塊傳輸(BLOCK -wise transfer)。

消息大小對於約束節點上的實現也非常重要。許多實現都需要為接收消息創建緩衝區。如果一個實現由於資源過於受限而無法分配足夠的緩衝區，那麼對於不使用DTLS的消息，它可以使用以下策略：如果接收到一個資料封包，但緩衝區太小不足以存儲整個資料封包，接收端通常能夠判斷出資料封包的尾部是否被丟棄，並且能獲得資料封包的開頭。一般來說,CoAP的頭部和option部分很可能在緩衝區中。因此服務端可以正確理解這個請求，如果有效負載部分被截斷了，可以返回一個4.13(請求資料過長，見第5.9.2.9節)的回應。當某個端發送一個冪等的請求，但接收到的回應大於它緩衝區大小，那麼它可以為Block Option設置一個恰當的值，重複發送這個請求，並為Block選項[BLOCK]提供一個合適的值。

4.7. 擁塞控制

CoAP的基本擁塞控制由指數回退機制提供，見章節4.2。

為了避免擁塞，用戶端（包括代理）應該嚴格限制他們同時與指定的伺服器（包括代理）維持的未完成交互的數量（即NSTART值)。一個未完成的交互可以是一個仍在等待ACK的CON消息；也可以是一個在等待回應訊息或者ACK的請求（這兩種情況可以同時出現，做為同一個未完成的交互）。在本協定裡NSTART的預設值為1。

另外，將來可能會有更多的擁塞控制優化和注意事項，例如，可能會自動初始化章節 4.8 中定義的 CoAP 傳輸參數，因此可能允許 NSTART 的值大於 1。

當 EXCHANGE_LIFETIME 超時後，如果該 CON 請求還沒有收到回應，用戶端就會停止等待。用戶端停止等待一個已經收到 ACK 的 CON 請求(單獨回應情況)或者對一個 NON 請求的回應的策略還未被定義。除非有額外的擁塞控制優化，否則它向其他未回應端的平均發送速率必須不超過 PROBING_RATE。

注意：CoAP 協定中，擁塞控制主要由用戶端實現。然而，用戶端可能會出現故障(或者用戶端實際上就是攻擊者)，例如，在章節 11.3 中提到的放大攻擊。為了將損失（網路頻寬及能耗）降到最低，對合理的應用請求，伺服器應該對回應限速。對於行為異常的端來說限速是有且最有效的辦法了。

4.8. 傳輸參數

資訊的傳輸由以下參數控制：

name	default value
ACK_TIMEOUT	2 seconds
ACK_RANDOM_FACTOR	1.5
MAX_RETRANSMIT	4
NSTART	1
DEFAULT_LEISURE	5 seconds
PROBING_RATE	1 byte/second

表 2: CoAP 協定參數

4.8.1. 改變參數

ACK_TIMEOUT、ACK_RANDOM_FACTOR、max_retransmission、NSTART、DEFAULT_LEISURE(章節 8.2)和 PROBING_RATE 的值可以轉換為特定於應用程式環境的值(包括動態調整的值) 但是配置方法超出本協定的討論範圍。本規範建議在應用環境中使用統一的參數。同樣，對於配置不一致的參數值所產生的影響也超出本協定討論範圍。

選擇傳輸參數是為了在網路出現擁塞時實現安全的行為。如果配置希望使用不同的值，則有責任確保不會違反這些擁塞控制屬性。特別是，ACK_TIMEOUT 降低到 1 秒以下將違反 [RFC5405][RTO-CONSIDER]也提供了一些額外的說明)。 CoAP 的設計可以保證那些不能維持 RTT 測量的應用運行。然而，要減小 ACK_TIMEOUT 或者增加 NSTART，只有在能保持那個測試結果的時候安全進行。如果不使用確保擁塞控制安全的機制(在配置中或在未來的標準文件中定義)，配置不能減少 ACK_TIMEOUT 或增加 NSTART。

ACK_RANDOM_FACTOR（應答隨機因數）絕對不能小於 1.0，且最好有一個和 1.0 不同的值以保護同步。

MAX_RETRANSMIT（最大重傳時間）可以自由的調整，但是如果太小的話將會減小收到 CON 包的概率，然而如果比這裡給定的值要大的話，那些對時間有要求的參數要做更進一步的調整（參見章節 4.8.2）。

如果傳輸參數的選擇導致衍生時間的增加（參見章節 4.8.2），必須確保調整後的值對所有通信端都有效。

4.8.2. 傳輸參數的衍生時間

ACK_TIMEOUT（應答超時時間）、ACK_RANDOM_FACTOR（應答隨機因數）和 MAX_RETRANSMIT（最大重傳時間）這三個參數共同影響著重傳時間，重傳時間反過來也影響著這些參數需要保持的時間長度。為了能夠對這些 derived 時間值有一個明確的參考，這裡給出以下名稱：

- 最大傳輸跨度（MAX_TRANSMIT_SPAN）：指的是從 CON 消息第一次發送到它的上一次重發之間的最大時間間隔。對默認的傳輸參數，它的值是(2+4+8+16)*1.5 = 45s，或者一般表示為：

```
ACK_TIMEOUT * ((2 ** MAX_RETRANSMIT) - 1) * ACK_RANDOM_FACTOR
```

- 最大傳輸等待時間（MAX_TRANSMIT_WAIT）：是指從第一次傳輸 CON 消息到發送方放棄接收 ACK 者 RST 回應之間的時間間隔。默認的傳輸參數的值是 (2+4+8+16+32)*1.5 = 93 秒，或者表示為：

```
ACK_TIMEOUT * ((2 ** (MAX_RETRANSMIT + 1)) -
1) * ACK_RANDOM_FACTOR
```

此外，我們還需要在一些網路和節點的特徵上做些假設。

- 最大延遲（MAX_LATENCY）指的是資料封包從開始發送到完全接收之間的最大時間。該常量與[RFC0793] 協定中的 MSL（最大段週期）相關，一般被設定為 2 分鐘([RFC0793] glossary，第 81 頁)。注意，這並不一定比最大傳輸等待時間小，最大延遲並不是想要描述當協定工作良好的情形，而是確保在最壞的情況下有保障。我們也可以很隨意的定義最大延遲為 100 秒。除了大量的配置與以前的 TCP 接近，這個值也允許 Message ID 存活時間計時器由 8-bit 數值表示（單位為秒）。在這些計算中，沒有考慮傳輸方向的影響（假設網路是對稱的）。如果不是這種情況，接下來計算將變得稍微複雜一些。

- 處理延時（PROCESSING_DELAY）指的是 CON 消息得到 ACK 回應的時間。我們假設節點恰好在發送端超時之前發送 ACK，那麼這個時間就等於 ACK_TIMEOUT。

- MAX_RTT：往返時間的最大值，或者是：

(2 * MAX_LATENCY) + PROCESSING_DELAY

從這些值中，我們可以得到與協定操作相關的下列值：

- 交換週期(EXCHANGE_LIFETIME)：它是指從開始發送 CON 消息到不再接收 ACK 之間的時間，即該資訊在消息層交換時可以被清除。交換週期包括 MAX_TRANSMIT_SPAN、發送過程的 MAX_LATENCY、PROCESSING_DELAY 和接收過

程的 MAX_LATENCY。注意，如果最後等待週期 (ACK_TIMEOUT * (2 ** MAX_RETRANSMIT)或 MAX_TRANSMIT 和 MAX_TRANSMIT_WAIT 的差)小於 MAX_LATENCY，這裡就無需考慮 MAX_TRANSMIT_WAIT(這種情況一般不可能出現)。在這種情況下，EXCHANGE_LIFETIME 簡化為：

```
MAX_TRANSMIT_SPAN + (2 * MAX_LATENCY) + PROCESSING_DELAY
```

一般缺省值為 247 秒。

- 不需確認消息週期(NON_LIFETIME)：它指的是從發送 NON 消息到該 Message ID 可以被覆用之間的時間。如 NON 消息沒有多次發送，那麼它的值是 MAX_LATENCY 或者是 100 秒。然而，尤其是在多播應用中，一個 CoAP 發送端可能發送 NON 消息很多次。Message ID 重用超出本文範圍內。接收端希望在 MAX_TRANSMIT_SPAN 的時間內判斷是否是重傳包。基於這樣的目的，使用缺省值 145 秒或使用下面的值將會更安全：

```
MAX_TRANSMIT_SPAN + MAX_LATENCY
```

對於僅想通過一個超時時間來判斷 Message ID 是否能夠重用的應用來說，使用較大的 EXCHANGE_LIFETIME 更安全。

表 3 列舉了上述一些參數的缺省值

name	default value
MAX_TRANSMIT_SPAN	45s
MAX_TRANSMIT_WAIT	93s
MAX_LATENCY	100s
PROCESSING_DELAY	2s
MAX_RTT	202s
EXCHANGE_LIFETIME	247s
NON_LIFETIME	145s

表 3: 衍生的協定參數

5. 請求/回應語義

CoAP 在 HTTP 相似的請求/回應模型下運行；CoAP 終端在規則中為客戶端，向伺服器發送一個或更多 CoAP 請求，伺服器藉由發送 CoAP 回應請求服務。不像 HTTP，請求和回應不是在先前的既定連結上發送，而是通過 CoAP 訊息上異步交換。

5.1. 請求

CoAP 請求構成要應用於資源的方法，資源的鑑別碼，人事跟網路媒體類型(如果有的話)，跟隨意關於請求的後設資料。

CoAP 協定支援基本的 GET，POST，PUT 和 DELETE 方法，這些方法可以簡單的映射到 HTTP 中。它們和 HTTP（見[RFC2616] 章節 9.1）有相同的安全屬性（僅檢索）和冪等（多次調用有同樣的效果）特性。獲得的方法是安全的，因此，只能對這個資源進行檢索，不能進行其他任何操作。獲得，PUT 和 DELETE 方法必須以冪等的方式執行.POST 不支持冪等，因為它的效果是由源服務端和目標資源決定的，POST 的結果是建立一個新資源或者是更新目標資源。

請求通過設置 CoAP 協定頭部的欄位來初始化一些資訊，比如該消息是是 CON 還是非的。

請求使用的方法在章節 5.8 中有詳細的描述。

5.2. 回應

在接收到並解析了一個請求之後,服務端會回應一個 CoAP 回應，這個回應是通過客戶端生成的權杖來匹配。注意，這和 CON 消息要與 ACK 通過消息 ID 匹配的機制不同。

一個回應是通過 CoAP 協定標題的欄位來定義為一個狀態碼，與 HTTP 狀態碼類似,CoAP 協定狀態碼顯示嘗試理解並滿足請求的結果，這些代碼在章節 5.9 有完整的定義。狀態碼的編號是在 CoAP 協定標題的欄位中設定的，並保存在 CoAP 協定狀態碼表中（章節 12.1.2）

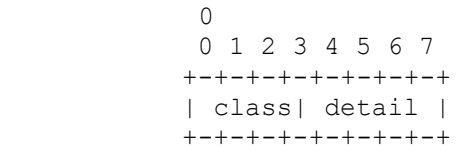


圖 9：回應代碼結構

上面 8-bit 的回應碼中的高 3-bits 定義回應的種類。低 5-bits 沒有任何分類作用，他們僅提供一些額外的細節（圖 9）

作為規範和協定診斷的人類可讀符號，CoAP 狀態碼格式是"c.dd"，這裡"c"是一個十進位數字，"dd"是兩位十進位數字。舉個例子，"Forbidden"被定義為 4.03——這是一個 8-bit 的值，16 進制為 0x83（40c20+3）或者十進位 131（432+3）。

有 3 種狀態碼：

2- Success:代表成功收到請求，理解並接收。

4 - client Error：用戶端錯誤，代表請求包含了錯誤的語法或者服務端不能滿足請求。

5 - server error：服務端錯誤代表伺服器未能回應請求。

狀態碼被設計成可擴充的：在客戶端或者服務端錯誤中，如果某個端不能識別具體狀態碼，這個端會將之視為一般狀態碼(即 4.00 或 5.00）然而，對於成功類，如果某個端不能識別具體狀態碼，那麼這個端只能判斷這個請求成功，而不能有更多的操作。

在章節 5.9 對狀態碼的做了詳細的描述。

回應能被通過多種方式發送，在接下來的小節中有定義。

5.2.1. 附帶回應

在大多數案例中，回應直接攜帶在 ACK 中（如果這個請求是 CON）。這稱為「附帶回應」。

無論回應顯示是成功還是失敗，回應通過 ACK 消息回傳。實際上，回應是附帶在 ACK 中，不需要單獨發送一個回應消息。

實現注意：規範將是否在 ACK 消息附帶回應的決定權(例，發送一個單獨回應)留給服務端，客戶端必須同時做好接收這兩種的準備。從實現層面考慮，強烈建議服務端只要有可能就採用附帶回應 - 這種方式能節省客戶端和服務端的網路資源。

5.2.2. 單獨回應

不是所有的消息都能將回應附帶在 ACK 中回傳。舉個例子，一個服務端可能需要一段時間（比 ACK 超時時間更長）來獲取資源表現，因此不需要冒險讓客戶端反復的重傳請求資訊（參見章節 4.8.2 中 PROCESSING_DELAY 的討論）。當請求是攜帶在 NON 中時，回應總是和 ACK 分離的（因為 NON 不需要 ACK）。

服務端產生單獨回應的時機是，服務端在獲得資源表現過程中，ACK 定時器超時。如果服務端提前知道不會有附帶回應，服務端也可能馬上發一個 ACK。在這兩個情況中， ACK 都表示請求將馬上執行。

當服務器最終獲得了資源表現後，它發送這個回應。服務器希望消息不丟失，它會選擇一個新的消息 ID，發送 CON 到客戶端，並需要客戶端回復一個 ACK（可能也會發送 NON，參見章節 5.2.3）。

當服務端選擇使用單獨回應，它會發送空的 ACK 給 CON 請求。只要服務端回復了空的 ACK，即使客戶端重傳另一個同樣的請求，它也不能再回復附帶回應。如果收到一個重傳的請求（可能是因為原有的 ACK 延遲了），服務端發送另一個空的 ACK，所有的回應都必須發送單獨回應。

如果這個服務端接著發了一個 CON 的單獨回應，客戶端回復這個回應的 ACK 也必須是空的資訊（不攜帶請求和回應的消息）。收到 ACK 後，服務端在所有匹配的 ACK（忽略任何狀態碼和有效載荷）和 RST 消息中停止重傳該回應。

實現注意：由於底層資料封包傳輸可能是無序的，因此，單獨回應的 CON 消息可能在請求的空 ACK 前到達客戶端為了避免重傳，這個 CON 消息也作為 ACK 來處理也要注意，雖然 CoAP 協定協定本身不做任何特殊的要求，但應用可能期望回應會在一個時間期限內到達。因為底層傳輸沒有保活機制，請求者可能要建立一個超時時間（與 CoAP 協定重傳時間無關）以防服務端出問題或者不能回應。

5.2.3. 無需確認消息

NON 如果是 NON 請求，回應也應該通過 NON 回復。然而，端必須在發送 CON 請求時準備好接收一個 NON 回應（領先或者落後於一個空的 ACK），在發送 NON 請求時準備好接收一個 CON 回應。

5.3. 請求/回應的匹配

不管回應是怎麼被發送的，回應和請求依靠包含在客戶端的請求中的權杖（權杖），以及額外的相關端位址資訊來匹配。

5.3.1. 權杖

權杖是用於匹配回應與請求的.token 值有 0~8 字節（注意，每個資訊都攜帶權杖，即使其長度為零）。每個請求都攜帶由客戶端生成的權杖，服務端在回應時必須複製（不能修改）這個權杖。

token 用作 client-local 標示，用於區分並發請求（參見章節 5.3），也稱為"請求 ID"。

客戶端生成權杖時需要注意，當前使用的權杖對給定的源端和目的端應該都是獨一無二的。（注意客戶端在生成權杖時，如果要向不同的端（比如源埠號不同）中發送請求，可以使用同樣的權杖）。當只向目的端產生一個權杖，或者向每個目的端發送的請求都是順序的，且都是附帶回應，權杖為空也是可行的。有多種策略實現。

如果客戶端不使用傳輸層安全（TLS，見章節 9）發送請求，就需要使用複雜的，隨機的權杖來防止欺詐回應（見章節 11.4），起到保護功能，這也是權杖允許使用最多 8 個字節的原因.token 中隨機組件的實際長度取決於客戶端的安全需求和欺詐回應造成的威脅程度。接入到網際網路的客戶端至少應該使用 32 位隨機碼，記住，沒有直接連接網際網路也不一定能有效防範欺詐。注意，消息 ID 幾乎沒有添加保護，因為它通常是順序分配的，因此可能被猜測到，並通過欺詐回應繞過。客戶端想要優化權杖長度，可能會向進一步檢測正在進行的攻擊等級（例如計算接收的權杖不匹配的消息個數）。[RFC4086]討論對安全的隨機性要求。

端接收一個不是它生成的權杖，必須把這個令當成不透明的，不能假設它的內容和結構。

5.3.2. 請求/回應匹配規則

確切的匹配回應與請求的規則如下：

1. 回應的源端必須和原始請求的目的端一致。

2. 在附帶回應中，CON 請求和 ACK 的消息 ID 必須匹配，回應和原始請求的“權杖”必須匹配。在單獨回應中，只需回應和原始請求的權杖匹配。

萬一資訊攜帶異常的回應（不是認定的端，端地址，權杖和客戶端的期望不匹配），這個回應必須被拒絕（見章節 4.2 和 4.3）。

實現注意：客戶端接收到 CON 回應之後，可能想在回復完 ACK 馬上清除這個消息的狀態。如果這個 ACK 丟失，且服務端重傳這個 CON 消息，客戶端可能不會再有任何與該回應相關聯的狀態，會導致這個重傳成為異常消息；客戶端可能會發送 RST 資訊，這樣它就不會再收到更多的重傳消息這個行為是正常的，並不是一個錯誤（沒有積極優化內存使用狀態的客戶端會將第二個 CON 認定為重發。客戶端事實上期望從服務器[觀察]得到更多消息，就必須在任何情況下保持狀態）

5.4. 選項

請求和回應可能包含一個或多個選項的列表。舉個例子，請求消息裡的 URI 是都在多個選項中傳輸，在 HTTP 中元數據可能會攜帶在 HTTP 頭欄位，也是作為選項來提供的。

CoAP 協定定義了一組用於請求和回應的選項:

- 內容格式
- ETag
- 位置路徑
- 位置查詢
- 最大年齡
- 代理-URI
- 代理計劃
- URI 的主機
- URI 路徑
- URI 的埠
- URI 查詢
- 接受
- 如果-匹配
- 如果-無-匹配
- 尺寸 1

這些選項的語義和他們相應的屬性都在章節 5.10 有詳細定義。

並不是所有的選項都被定義可以使用所有方法和回應代碼。方法和狀態碼的可能的選項都各自被定義在 5.8 和 5.9 節。如果一個選項沒有定義方法或回應代碼，那它就禁止包含在發送內容裡，並且必須被接收端當作未識別的選項。

5.4.1. 重要選項/非重要選項

選項分兩個種類：重要"critical"或者非重要"elective"。這兩者的不同之處是端如何處理一個不能識別的選項。

- 根據接收情況，不能識別的非重要選項必須忽略。
- 不能識別的非重要選項出現在一個 CON 請求中，必須返回 4.02（Bad Option）的回應。這個回應應該包含一個診斷的有效負載來描述這個不能識別的選項（見章節 5.5.2）。
- 不能識別的重要選項出現在一個 CON 回應中或者附帶回應的 ACK 中，必須拒絕這個回應（章節 4.2）。
- 不能識別的"關鍵"選項出現在一個非消息中，必須拒絕這個消息。

注意，無論重要還是非重要，選項永遠不會強制（總是可選）：這些規則是為了實現停止處理它們沒有理解的或沒有執行的選項。

重要/非重要規則不適用於代理。代理處理選項基於不安全/安全轉發（定義於章節 5.7）

5.4.2. 代理不安全或安全跟無快存金鑰

一個選項除了被標註為重要或者不重要之外，選項同樣會基於代理如何處理不能識別的選項來分類。為此，選項可以被視為不安

全轉發（unsafe 標識被設置），或者安全到 - 序（不安全的標識被清除）。

此外，在請求中對於被標記為 safe-to-forward 的選項，選項編號表明它是否成為緩存密鑰（見章節 5.6）的一部分。只要有一位 NoCacheKey bits 是 0，它就是 cache-key 的一部分；如果所有的 NoCacheKey 位是 1，它才不是（見章節 5.4.6）。

注意：高速緩存 - 關鍵字只和依賴於不安全/安全到正向指示，而不是將給定的選項執行為請求選項的代理相關舉個例子，ETag 的，使用請求選項作為緩存鍵的一部分實際上是非常低效的，但如果 ETag 的沒有被代理執行，這就是你所能做的最好的事，因為回應將根據請求選擇而變化。一個更有用的代理，不用執行 ETag 的請求選項，就是不使用 ETC 作為 Cache-Key 的一部分.

NoCacheKey 以 3 位表示，所以八個狀態碼只有一個是 NoCacheKey，剩下七個狀態碼代表其他情況。

與這些分類相關的代理行為在章節 5.7 有定義。與這些分類相關的代理在 5.7 節中有定義。

5.4.3. 長度

選項值定義有一個特定長度，通常的形式是一個上界和下界。如果在請求中的選項的長度值超過定義的範圍，這個選項必須當做一個不能識別的選項處理（參見章節 5.4.1）

5.4.4. 默認值

選項可能有被定義一個默認的值。如果一個選項的值是默認值，這個選項就不應該包含在消息中，如果這個選項不存在，必須假定為默認值。

當一個重要選項有一個默認值時，就會以這種方式被選擇，消息裡的選項空缺可以被兩種執行合理地處理，一種是沒有意識到重要選項的執行，另一種是將空缺解釋為存在選項默認值的執行。

5.4.5. 可重複選項

一些選項的定義指定這些選項是可重複的。一個消息中可能包含一個或者幾個可重複的選項。一個不可重複的選項，在一個消息中絕對不能出現超過一次。

如果消息裡的選項出現的次數比定義的選項多，隨後出現的多餘選項必須當做無法識別的選項（參見章節 5.4.1）。

5.4.6. 選項編號

一個選項由一個選項編號所定義，這個編號也能提供一些額外的語義資訊，比如，奇數編號代表一個重要的選擇，偶數編號代表非重要選項注意，這不只是一個約定，也是協定的一個功能：選項是否重要，取決於這個選項編號是奇數還是偶數。

更通俗點說，一個選項編號由一位編碼來確定這個選項是否重要，是不安全的還是非常安全，還有，如果非常安全，還提供一個緩存鍵，如下圖所示。在下文中，這位編碼被表示成為一個字節，當 LSB 格式時，選項編號表示成無符號整數。當位 7（最低有效位）是 1，選項是重要（同樣，如果是 0 的話為非重要）。當 bit6 是 1，選項是不安全的（同樣，如果是 0 的話為 safe-to-forward）。當 bit6 是 0 的時候，也就是這個選項不是不安全。當且僅當位 3-5 全都設置為 1 的時候它不是一個緩存鍵（NoCacheKey），所有其他的位的組合，代表它確實是一個高速緩存 - 關鍵字。這些選項的分類在下面章節中有闡述。

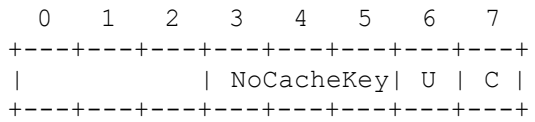


圖 10：選項號掩碼（最低有效字節）

一個端可以使用一段等價的 Ç 代碼（如圖 11），來得到這個選項的編號"onum"的特性。

```
Critical = (onum & 1)； UnSafe = (onum & 2)；
NoCacheKey = ((onum & 0x1e) == 0x1c)；
```

圖 11：選項編號的確定特性

選項的編號都定義在 CoAP 選項編號文件中（參見章節 12.2）

5.5. 有效負載和表示法

請求和回應依賴於各自的方法碼或者返回碼，因而可以包含有效載荷。如果一個方法碼或者返回碼沒有定義為包含有效載荷，那麼發送端一定不能包含有效載荷，而且接收端必須要忽略它。

5.5.1. 表現

請求或者回應的有效負載標明瞭成功是資源的表現(「資源表示」)或者請求動作的結果（"action result"）。它的格式由網際網路媒體類型指定,內容編碼由內容格式選項決定。當缺乏這個選項時，沒有假定的默認值，那麼格式將由應用來決定（比如從應用的內容上）。如果沒有指定的內容類型，才會嘗試使用探測的有效載荷。

實現注意：在執行質量層面，強烈建議由資源表現來提供內容，格式的顯示因為這不是協定的要求，所以並不是一個強制要求，同時也很難概述出究竟在什麼情況下會違反這種。建議。

對於標示客戶端或者服務端錯誤的回應，當且僅當內容格式（內容格式)選項給出時,有效載荷被認為是請求動作的結果的表示。當沒有該選項時，有效載荷是診斷式的（章節 5.5.2）。

5.5.2. 診斷式 有效負載

如果沒有內容格式（內容格式）的選項，回應中的有效載荷使用簡潔可讀的診斷資訊來指示一個客戶端錯誤或者服務端錯誤，並解釋錯誤的情況。該診斷資訊一定要使用 UTF-8 格式編碼[RFC3629]，更確切的說是使用淨 Unicode 的格式[RFC5198]。

這種消息和 HTTP 的狀態行中的原因描述(Reason-Phrase)很類似。它不是為終端用戶設計的，而是為軟件工程師設計的，因為調試過程中需要在當前的上下文中用符合英語語言規範來解釋它；因此，不需要提供語言標記的機制。與 HTTP 不同的是，如果在返回碼(Response Code)外沒有額外的信息，負載必須是空的。

5.5.3. 經由選擇的表現

不是所有的攜帶有效負載的回應都提供由請求尋址的資源的表現。然而，能夠參考與回應相關的表示有時也是有用的，不依賴於它實際上是否封閉獨立。

我們使用術語"selected representation"來引用被一個成功的回應選中的目標源的當前表示，如果這個相關的請求已經使用了 GET 方法，並且不包括任何有條件的請求選項。(章節 5.10.8)

已確定的回應選項提供關於選擇表示的元數據，它可能不同於回應一些包含狀態改變的方法的消息的表現。在該篇規範中定義的回應選項，只有 ETag 回應選項（章節 5.10.6）定義為關於選擇表示的元數據。

5.5.4. 內容協商

服務器可能會以多種表示格式來提供資源表示。如若沒有來自客戶端的更多資訊，服務端會提供它偏愛的格式。

通過使用請求中的接收選項(章節 5.10.4)，客戶端能夠標示它偏愛接收的內容格式。

5.6. 緩存

CoAP 端點為了減少回應時間和網路頻寬消耗，它可以緩存回應。

CoAP 中緩存的目標是通過重利用先前的回應消息來應答當前的請求。在某些情況下：甚至無需網路請求，就能夠重利用已經存儲的回應，從而減少延時和網路回傳；一種名為"freshness"的機

制用於這個目標(參照章節 5.6.1)。甚至當有一個新請求時，通常可以重利用先前回應的有效負載來應答該請求，從而減少了網路頻寬的消耗；一種名為"validation"的機制用於這個目標(參照章節 5.6.2)。

與 HTTP 不同，CoAP 回應的緩存能力並不依賴於請求方法，而是依賴於返回碼(Response Code)。在章節 5.9 中返回碼定義中列出了每種返回碼代表的緩存能力。端點標示成功和無法識別的狀態碼必須不能被緩存。

對於已提出的請求，CoAP 端點一定不能使用已存儲的回應，除非：

- 已提出的請求方法和用於獲取存儲回應的方法相匹配

- 已提出的請求和那些用於獲取存儲回復的請求(包含請求 URI)的所有選項相匹配，除了不需要匹配標記為 NoCacheKey 的任何請求選項(章節 5.4)或者能被緩存識別並且相對於緩存行為能夠完全解釋的選項匹配(比如在 5.10.6 中描述的 ETag 請求選項，也可以參考章節 5.4.2)

- 已儲存的回復是按照下面將要定義的更新或者成功驗證。

用於匹配緩存入口的請求選項族可以全體稱為"Cache-Key"。比起 coap 和 coaps，在 URI 格式中，構成請求 URI 的選項匹配可以在 URI 格式下的特定規則中執行。

5.6.1. 新鮮度模型

當緩存中的回復是「新鮮」的，不用聯繫原始服務端就能用於應答請求，因此提高了效率。

對起點服務器的測定新鮮度的機制通過使用 Max-Age 選項(章節 5.10.5)在未來提供一個明確的到期時間。Max-Age 選項的意思是當回應的時間超過了指定的秒數後就被認定為"不新鮮"的。

Max-Age 選項的默認值是 60。所以，如果回應不在一個可緩存的回應中，那麼在 60 秒後該回應就被認定是不新鮮的。如果最初的服務器希望禁用緩存，它就必須將 Max-Age 選項的值指定為 0 秒。

如果客戶端有一個新鮮的已存儲的回應，並且為已存儲的回應生產了一個匹配的新請求，新的回應就會使得舊的回應失效。

5.6.2. 校驗模型

當端點對一個 GET 請求有一個或多個存儲的回應，但又不能使用其中任意一個時(例如它們都不是新鮮的)，它能夠使用 GET 請求中的 ETag 選項（章節 5.10.6）給原始服務端一個選擇存儲的回應並且更新它的新鮮度的機會。這個過程稱為驗證或者重驗證已經存儲的回應。

當發送一個這樣的請求，端點應當為每個適當的存儲回應添加一個 ETag 選項來指定它們的 entity-tag。

按照章節 5.9.1.3 中的描述，攜帶 2.03(Valid)狀態碼的回應中，ETag 選項中的 entity-tag 所標識的已存儲回應，可以在完成更新後重新使用。

其它任何狀態碼都表明請求中的已存儲回應都不適用。相反，回應應當用於應答請求並替代已存儲的回應。

5.7. 代理

代理是能夠代表 CoAP 客戶端執行請求的 CoAP 端。當客戶端不能生成請求，或者需要減少回應時間、降低網路頻寬或者功耗，因此需要 cache 回應時，代理相當有用。

在受限的 RESTful 環境中的整體架構中，代理可以實現完全不同的目的。客戶端可以明確地選擇代理，我們稱為正向代理。代理也可以被插入來代替原始服務器，我們稱為“反向代理”。從 CoAP 請求映射到 CoAP 請求(CoAP-to-CoAP)的代理或者轉換不同的協

定(跨協定)的代理，可以和正向代理、反向代理互相組合。在章節 1.2 中有這些術語的完整定義。

注意：這篇規範中的術語與網際網路應用環境中的術語是相容的，在各項細節中無需匹配它(甚至都與受限的 RESTful 環境無關)。沒有太多的語義應該歸屬於術語的成分(例如正向，反向或者跨協定)。

HTTP 代理，除了作為 HTTP 代理，通過提供傳輸層協定的代理功能來保證端對端通信的傳輸層安全。這篇規範中並沒有在 CoAP-to-CoAP 的代理中定義這樣的功能，因為在受限的 RESTful 環境中轉發 UDP 包看起來沒太多價值。可以參考章節 10.2.7 中的跨協定代理例子。

當客戶端使用代理提出請求，會使用一個安全的 URI 方案(例如 "coaps"或"https")，只要在客戶端和代理之間沒有使用等效的底層安全機制，那麼發往代理的請求必須使用 DTLS。

5.7.1. 代理操作

根據從客戶端接收到的請求，代理通常需要一種為到目的端的請求分配可能的請求參數的方法。該方法完全由正向代理指定，但是也可以依賴於反向代理的特定配置。特別是，反向代理的客戶端通常沒有標示目的端的定位器，因此有必要在反向代理中有命名空間轉換的格式。然而，代理操作的一些方面對於其各種形式是常見的。

如果代理沒有使用緩存，那麼它僅僅簡單的往指定的目的端轉發請求。否則，如果代理使用緩存但是沒有與轉化的請求相匹配的，且已存儲的新鮮的回應，那麼根據章節 5.6 它需要更新緩存。代理識別請求的 option，它應當知道該 option 是否能夠當作在緩存值中查詢的鍵值的一部分。舉個例子，由於對於不同 Uri 路徑值的請求指向不同的資源，Uri 路徑值通常當作 Cache-Key 的一部分，而 token 值從來都不能當作 Cache-Key 的而一部分。對於代理沒能識別的但是在選項碼中標記為 Safe-to-Forward 的選項，選項也標示了它是否包含在 Cache-Key 中(NoCacheKey 沒有完全設

定或者完全設定)。（無法識別並且標記為 Unsafe 的選項就是 4.02Bad Option）。

如果發往目的端的請求超時了，那麼必須返回一個 5.04(GateWay Timeout）回應。如果發往目的端的請求返回一個無法被代理處理的回應(比如無法識別的關鍵選項或者消息格式錯誤)，那麼必須返回一個 5.02(Bad Gateway)回應。否則代理向客戶端返迴回應。如果回應在緩存中生成，生成的（或者隱含的）Max-Age 選項一定不能超過最初由服務端設定的 max-age，max-age 表示在緩存中資源存活的時間。舉個例子，對於每條回應可以按照下面的公式由代理計算出 Max-Age 選項：

```
proxy-max-age = original-max-age - cache-age
```

舉個例子，如果代理資源在 20 秒之前更新，並且最初的 Max-Age 是 60 秒，那麼代理中該資源的 max-age 就是 40 秒。考慮到初始服務器的潛在網路延時，代理生成的回應的 max-age 值最好比該值偏小。

出現在代理請求裡的所有選項都必須被處理。請求中無法被代理識別的 Unsafe 選項一定會導致代理返回一個 4.02（Bad Option）回應。CoAP-to-CoAP 代理必須往原始服務端轉發不能識別的所有 Safe-to-Forward 選項。類似的，在回應中無法被 CoAP-to-CoAP 代理服務端識別的 Unsafe 選項會導致一個 5.02(Bad Gateway)的回應。此外，不被識別的 Safe-to-Forward 選項必須被轉發。

在第 10 章中詳細討論 CoAP 和 HTTP 的跨協定代理。

5.7.2. 正向代理

CoAP 區分原始服務端的請求和正向代理的請求。CoAP 向正向代理的發出請求是普通的 CON 和 NON 請求，但是它們用不同的方式來指定請求 URI：代理請求中的請求 URI 指定為 Proxy-Uri 選項中的字串（參考章節 5.10.2），而原始服務端中的請求分為 Uri-Host，Uri-Port，Uri-Path 和 Uri-Query 選項(參考章節 5.10.1)。作為另一種選擇，代理請求中的 URI 也可以由 Proxy-Scheme 選項和剛才提到的分開的選項組合而成。

當端點收到一個代理請求，而端點不想或者不能當作代理來處理這個請求URI，那麼它必須返回一個5.05(Proxying Not Supported）的回應。如果授權(host and port)當作代理端點本身(章節 5.10.2)，那麼請求必須當作一個本地的（non-proxied）請求。

一般來說，代理必須按照下面來解釋請求：請求 URI 的設計定義輸出的協定和它的細節(例如 coap 設計編碼 CoAP 是在 UDP 之上的，而 coaps 設計編碼是在 DTLS 之上的)。對於一個 CoAP-to-CoAP 的代理，初始服務端的 IP 位址和埠是由請求 URI 的授權分量決定的，請求 URI 可以被解碼分為 Uri-Host，Uri-Port，Uri-Path 和 Uri-Query 選項。如果代理被配置為將代理請求轉發到另一個代理，上面的解釋方法會使其佔用 Proxy-Uri 或 Proxy-Scheme 選項，導致其無法轉發到原始服務端。

5.7.3. 反向代理

反向代理不會利用 Proxy-Uri 或者 Proxy-Scheme 選項，但是需要從請求資訊和配置資訊中確定請求的目的端（下一跳）。例如，反向代理在通過資源偵測獲知資源的存在後，反向代理能提供各種資源，好像它們就是自己的資源一樣。反向代理可以自由的為識別這些資源的 URIs 建立命名空間。反向代理也可以構建一個命名空間，可以讓客戶端更好的控制請求路徑，比如，將主機標識符和埠編號嵌入到資源的 URI 路徑。

在回應處理中，反向代理必須謹慎處理：不同資源的 ETag 選項值不能和客戶端提供的資源混合起來。在很多情況下，ETag 能夠在不修改的條件下被轉發。如果從反向代理提供的資源到各種初始服務器提供的資源的映射不是唯一的，反向代理需要生成一個新的 ETag，來保證該選項的語義是正確的。

5.8. 方法定義

在本小節中，每個方法及其行為都有定義。帶有無法識別或者不支援的方法碼的請求必須生成一個 4.05(Method Not Allowed)的附帶回應。

5.8.1. GET

GET 方法根據請求 URI 定位資源，從相符合的資訊中獲取對應的表現。如果請求包含Accept選項，就表明了首選的回應內容格式。如果請求包含 ETag 選項，GET 方法要求驗證 ETag 並且只有當驗證失敗的時候才會傳輸表現。當驗證成功，回應中應當包含 2.05（Content）或者 2.03（Valid）的狀態碼。

GET 方法是安全而且冪等的。

5.8.2. POST

POST 方法要求處理包含在請求中的表現。POST 方法執行的實際功能由原始服務端決定，並依賴於目標資源。它通常的結果是創建新資源或者更新目標資源。

如果服務端回應請求，創建了資源，那麼返回的狀態碼為 2.01（Created），並應當在一系列 Location-Path 和/或 Location-Query 選項（章節 5.10.7）中包含新資源的 URI。如果 POST 方法成功但不是由服務端創建新資源引起的，那麼回應中應當含有 2.04（已修改）的狀態碼。如果 POST 方法成功並且是由刪除目標資源引起的，那麼回應中應當包含 2.02（已刪除）的狀態碼。

5.8.3. PUT

PUT 方法要求更新或創建由請求 URI 定位的資源。表現的格式由媒體類型和 Content-Format 選項中的內容編碼製定。

如果請求 URI 中的資源已存在，那麼封閉表現應當看做是該資源修改後的版本，並且應當返回一個 2.04（Changed）的狀態碼。如果沒有對應的資源，那麼服務端可能根據 URI 創建一個新的資源，返回一個 2.01（Created）的狀態碼。如果既不能創建資源，又不能修改資源，那麼應當發送一個恰當的錯誤狀態碼。

在 If-Match（章節 5.10.8.1）或 If-None-Match（章節 5.10.8.2）中包含更多對於 PUT 方法的限制規定。

PUT 方法不安全但是冪等的。

5.8.4. DELETE

DELETE 方法要求刪除由請求 URI 定位的資源。如果成功或者該資源不存在，必須返回 2.02（已刪除）的狀態碼。

DELETE 方法不安全但是冪等的。

5.9. 返回碼的定義

在下面將要說明的返回碼中，包括了所有請求可能導致的回應。同時，某些地方會說明該返回碼對應的 HTTP [RFC2616]中的返回碼，但這並不意味著會修改章節 10 中 CoAP 和 HTTP 的映射。

5.9.1. 成功 2.xx

這類返回碼表明客戶端的請求被成功的接收、理解並接受。

5.9.1.1. 2.01 Created

類似 HTTP 的返回碼 201(“Created”)，但該返回碼僅用於對 POST 和 PUT 請求的回應。回應包中如果同時帶有有效負載，則表明了服務端處理的結果。

如果回應包中包含一個或多個 Location-Path 與/或 Location-Query 的選項，這些選項的值說明這個被創建資源的路徑。否則，這個資源就在請求的 URI 下創建。一個緩存端如果收到這個回應必須將所有這個被創建資源的回應緩存標記為未刷新。

這種回應不能被緩存。

5.9.1.2. 2.02 Deleted

這個狀態碼和 HTTP 的狀態碼 204（“No Content”）類似，但僅僅用於回應那些導致該資源無效的請求，如 DELETE，或者某些情況下的 POST。回應包中如果同時帶有有效負載，表明了服務端處理的結果。

這個回應不能被緩存，一個緩存端必須將所有為這個被刪除資源的回應緩存標記為未刷新。

5.9.1.3. 2.03 Valid

這個狀態碼和 HTTP 的狀態碼 304（"Not Modified"）相關，但僅用於當攜帶了 ETag 選項時，說明（請求）中的 entity tag 是合法的。相應的，回應也必須攜帶 ETag 選項且必須不能包含有效負載。

如果一個緩存端能夠認出並且處理攜帶 ETag option 的回應報文，當它收到 2.03 回應時，它必須將其緩存的回應中的 Max-Age option 的值修改為和該回應的值一致（可能為明確的值或者是一個默認值，參見章節 5.6.2）。對於回應中每個轉發安全選項，所有緩存的回應中的對應選項中的值應該更新為該回應的值。非安全選項可能會觸發類似的選項自定義的特有的操作。

5.9.1.4. 2.04 Changed

這個狀態碼類似 HTTP 的狀態碼 204（"No Content"），但僅用於 POST 和 PUT 請求的回應。如果有的話，有效負載和回應一起返回，攜帶該操作的結果。
這個回應是不能被緩存的。因此，一個緩存端必須將針對這個被改變資源的緩存回應標記為未刷新。

5.9.1.5. 2.05 Content

這個狀態碼類似 HTTP 的狀態碼 200（"OK"），但僅用於 GET 請求的回應。

有效負載和回應一起返回，攜帶目標資源的內容。

這個回應是可以被緩存的，緩存端能夠使用 Max-Age 選項來決定刷新時間（參見章節 5.6.1 節），如果有 ETag 選項，也可以將其用於數據確認（是否有更新）的檢查（參見章節 5.6.2 節）。

5.9.2. 客戶端錯誤 4.xx

這個狀態碼集合裡包含了客戶端一些可能出錯的情況。這些狀態碼適用於所有請求方法。

服務端在如章節 5.5.2 的情況下，應該返回一個包含診斷資訊（diagnostic）的有效負載。

這個集合裡的回應是可以被緩存的，緩存端能夠使用 Max-Age 選項來決定刷新時間（參見章節 5.6.1），但不能用於確認檢查。

5.9.2.1. 4.00 Bad Request

這個狀態碼類似 HTTP 的狀態碼 400（"Bad Request"）

5.9.2.2. 4.01 Unauthorized

客戶端並未得到執行該操作的授權。客戶端不應該重複發送這個請求，除非改變了自己的授權狀態。當然，有一些特殊的機制在這種情況下能被使用（超出了本文的範圍），參見章節 9。

5.9.2.3. 4.02 Bad Option

這個請求裡面的一個或者更多的選項不能被服務端理解，或者有錯誤。客戶端不應該重複發送這個請求，直到修改那些選項。

5.9.2.4. 4.03 Forbidden

這個狀態碼類似 HTTP 的狀態碼 403（"Forbidden"）

5.9.2.5. 4.04 Not Found

這個狀態碼類似 HTTP 的狀態碼 404（"Not Found"）

5.9.2.6. 4.05 Method Not Allowed

這個狀態碼類似 HTTP 的狀態碼 405（"Method Not Allowed"），但並沒有類似的 Allow 頭部。

5.9.2.7. 4.06 Not Acceptable

這個狀態碼類似 HTTP 的狀態碼 406（"Not Acceptable"），但沒有回應實體。

5.9.2.8. 4.12 Precondition Failed

這個狀態碼類似 HTTP 的狀態碼 412（"Precondition Failed"）

5.9.2.9. 4.13 Request Entity Too Large

這個狀態碼類似 HTTP 的狀態碼 413"Request Entity Too Large"

除非服務端無法提供其最大接收的數據大小，否則應該提供一個攜帶 Size1 選項（參見章節 5.10.9）的回應。

5.9.2.10. 4.15 Unsupported Content-Format

這個狀態碼類似 HTTP 的狀態碼 415(“Unsupported Media Type”）

5.9.3. Server Error 5.xx

這個狀態碼集合裡包含的異常情況為：服務端出錯或者無法處理這個請求。這些狀態碼適用於所有請求方法。

服務端在如章節 5.5.2 的情況下，應該返回一個包含診斷資訊（diagnostic）的有效負載。

這個集合裡的回應是可以被緩存的，緩存端能夠使用 Max-Age 選項來決定刷新時間（參見章節 5.6.1），但不能用於確認檢查。

5.9.3.1. 5.00 Internal Server Error

這個狀態碼類似 HTTP 的狀態碼 500（"Internal Server Error"）

5.9.3.2. 5.01 Not Implemented

這個狀態碼類似 HTTP 的狀態碼 501（"Not Implemented"）

5.9.3.3. 5.02 Bad Gateway

這個狀態碼類似 HTTP 的狀態碼 502（"Bad Gateway"）。

5.9.3.4. 5.03 Service Unavailable

這個狀態碼類似 HTTP 的狀態碼 503（"Service Unavailable"），但是用 Max-Age 選項取代了 HTTP 的"Retry-After"頭部，用來表明下次重試需要等待多少秒。

5.9.3.5. 5.04 Gateway Timeout

這個狀態碼類似 HTTP 的狀態碼 504（"Gateway Timeout"）。

5.9.3.6. 5.05 Proxying Not Supported

請求中包含 Proxy-Uri 選項或者使用 Proxy-Scheme（參見章節 5.10.2），但是服務端無法（或者不願意）為其指定的 URI 做正向代理。

5.10. Option 的定義

CoAP 的那些選項在表 5-1 中總結，並且在後面的子章節中描述。

在表中，C，U 和 N 這幾列分別代表臨界（Critical），不安全（UnSafe）和不緩存（NoCacheKey）。由於 NoCacheKey 僅僅用於安全轉發（Safe-to-Forward）選項（沒有標記為 Unsafe），這一列被填滿了橫槓。

No.	C	U	N	R	Name	Format	Length	Default
1	x			x	If-Match	opaque	0-8	(none)
3	x	x	-		Uri-Host	string	1-255	(see below)
4				x	ETag	opaque	1-8	(none)
5	x				If-None-Match	empty	0	(none)
7	x	x	-		Uri-Port	uint	0-2	(see below)
8				x	Location-Path	string	0-255	(none)
11	x	x	-	x	Uri-Path	string	0-255	(none)
12					Content-Format	uint	0-2	(none)
14		x	-		Max-Age	uint	0-4	60
15	x	x	-	x	Uri-Query	string	0-255	(none)
17	x				Accept	uint	0-2	(none)
20				x	Location-Query	string	0-255	(none)
35	x	x	-		Proxy-Uri	string	1-1034	(none)
39	x	x	-		Proxy-Scheme	string	1-255	(none)
60			x		Size1	uint	0-4	(none)

C=Critical,U=Unsafe,N=NoCacheKey,R=Repeatable

表 4：選項

5.10.1. Uri-Host, Uri-Port, Uri-Path, and Uri-Query

Uri-Host，Uri-Port，Uri-Path，Uri-Query 選項都用來定位一個向原始服務端請求的目標資源。選項將請求 URI 的不同組件進行編碼，在這些選項的 value 值中，看不到百分號編碼（也稱 URL 編碼），且完整的 URI 能夠在任何有關的端被重建。CoAP URI 的語義在章節 6 定義。

將 URI 解析成選項的步驟定義在章節 6.4。這些步驟是由請求中的 0 個或者多個 Uri-Host，Uri-Port，Uri-Path，Uri-Query 選項決定，每個選項包含以下資訊：

- Uri-Host 選項定義了被請求資源的網路主機
- Uri-Port 選項定義了資源在傳輸層的埠號
- 每個 Uri-Query 選項定義了一個資源的參數。
- 每個 Uri-Query 選項定義了一個資源的參數。

注意：Fragments（[RFC3986]，見章節 3.5）並不是 URI 請求的一部分，因此在 CoAP 請求中不會被傳輸。

Uri-Host 選項的默認值是請求消息的目的 IP 位元址。同樣地，Uri-Host 選項的默認值即 UDP 的埠號。Uri-Host 和 Uri-Port 選項的默認值對於大部分服務端來說已經足夠了。當一個端有多個虛擬服務端時才會使用明確的 Uri-Host 和 Uri-Port 選項。

Uri-Path 和 Uri-Query 選項能夠被任何字元順序編碼。不會採用百分號編碼。Uri-Path 選項的值必須不可以是“.”或者“..”（因此 URI 請求必須在將填充選項前將它們解析替換掉）。

章節 6.5 定義了從選項構造請求 URI 的步驟注意，實現不一定要構造 URI；它可以簡單地通過檢查各個選項來查找目標資源。

例子可以在附錄 B 找到。

5.10.2. Proxy-Uri and Proxy-Scheme

Proxy-Uri 選項被用於生成一個向正向代理（章節 5.7）發送的請求。用來請求正向代理將該請求轉發給服務端或者從合法的緩存中返回一個回應。

該選項的值是一個絕對 URI（[RFC3986]，章節 4.3）。

注意，正向代理可能將該請求轉發給另一個代理，或者直接給服務端，由 absolute-URI 決定。為了避免請求循環，代理必須能夠識別所有它面對的服務端名字，包括任何別名，局部變化，以及 IP 地址。

接收具有 Proxy-Uri 選項的請求的端點必須導致返回 5.05（Proxying Not Supported）回應，該埠不能或不願充當請求的前向代理。

一個 Proxy-Uri 選項優先於任何 Uri-Host、Uri-Port、Uri-Path 或者 Uri-Query 選項（它們每一個都必須不能出現在一個包含 Proxy-Uri 選項的請求中）。

作為一個簡化一些代理客戶端的特例，絕對 URI 可以從 Uri-*選項中構建。當 Proxy-Scheme 選項存在，絕對 URI 能夠被構造如下：一個 CoAP 的 URI 按照章節 6.5 的定義被構建。在所得的 URI 中，初始的 scheme 的冒號之後，會被 Proxy-Scheme 選項的內容替換。注意，這只適用於在所需的 URI 組件中，除了 scheme 組件，其他要素實際上可以使用 Uri-*選項表示的情況。例如，通過一個 userinfo 組件來認證 URI 的話，只能使用 Proxy-Uri 組件。

5.10.3. Content-Format

Content-Format 選項即消息的有效負載段。Content-Format 的不同格式通過在章節 12.3 中的 CoAP Content-Format 表中定義，通過數字來索引。該選項不存在默認值，即任何無該選項的有效負載都是未定義的（見章節 5.5）

5.10.4. Accept

CoAP 的 Accept 選項用於表明哪些 Content-Format 能夠被客戶端接受。它的表示格式同樣也是在章節 12.3 中的 CoAP Content-Format 表中定義。如果沒有 Accept 選項，表明客戶端可以接收所有格式（也即該選項沒有默認值）。客戶端接受服務端返回它指定的格式。如果服務端無法提供客戶端指定的格式，服務端必須返回一個 4.06“Not Acceptable”，除非另外的錯誤碼比這個返回碼優先。

5.10.5. Max-Age

Max-Age 選項定義了一個回應在它被標記為未刷新前，最多能緩存的時間（見章節 5.6.1）。

該選項的值是一個整型的秒數，從 0 到 2**32-1（大約 136.1 年）。如回應中沒有定義這個選項，它的默認值是 60 秒。

這個值是傳輸時開始計時的，那些對 Max-Age 時間要求嚴格的服務器必須在每次重傳前更新這個值（見章節 5.7.1）。

5.10.6. Tag

實體標識作為本地資源標識符，用來區分該資源是否已經隨著時間推移而變化了。它由提供資源的服務端生成，可能通過版本、checksum、雜湊或者時間等方式來生成。一個收到實體標識的端必須將該標識視為不透明的且不能假定它的內容或結構。生成實體標識的端被提倡使用盡量壓縮的表達，因為客戶端和中間人可能保存多個 ETag 值。

5.10.6.1. 作為一個回應選項的 ETag

回應中的 ETag 選項提供了它的當前值（例如，當一個請求被處理後），作為代表標籤（“tagged representation”）。如果沒有 Location-* 選項存在，代表標籤就是目標資源的選定表現(章節 5.5.3)。如果一個或者多個 Location-*選項存在，且因此一個 URI 位址被指明，代表標籤表示客戶端需要通過給定的 URI 位址重新獲取資源。

一個 ETag 選項能夠包含在任何帶有代表標籤的回應中（當然，如果在 4.04 或者 4.00 回復中，沒什麼意義）。ETag 選項必須不能在一次回應中包含多次。

ETag 選項沒有默認值，如果它不存在回應中，服務端就沒有代表標籤的實體標識。

5.10.6.2. 作為請求選項的 ETag

在 GET 請求中，一個端如果已經從資源中獲取一個或者多個表現，且從回應獲得了它們的 ETag，就能夠為一個或者多個這些已保存的回應指定 ETag 選項。

服務器可以發出 2.03 有效回應(章節 5.9.1.3)代替 2.05 內容回應，如果給定的 etags 中的一個當前表現為實體標記，即是有效的；然後 2.03 有效回應在回應選項裡回應這個特定的 ETag。

事實上，客戶端能夠確定目前所存儲的表現是不是最新的（參見章節 5.6.2），而不需要重新將他們傳輸。

ETag 選項可能在一個請求中發生 0 次，1 次或者多次。

5.10.7. Location-Path and Location-Query

Location-Path 和 Location-Query 選項定義由一個絕對路徑、一個請求字串，或者二者一起組成的相對 URI。這些選項的組合包含在 2.01（Created）回應中，來定位這個被創建資源的位置，而這個回應是由 POST 請求（章節 5.8.2）引起的。這個地址是由請求 URI 分解得出。

如果回應攜帶一個或者多個 Location-Path 和/或 Location-Query 選項，該回應通過了一個解釋這些選項的緩存，且其中的某些回應緩存隱含了這個 URI，這些回應緩存必須被標記為未刷新。

每個 Location-Path 選項對應該資源絕對路徑的一個段，每個 Location-Query 選項對應該資源的一個參數。Location-Path 和 Location-Query 選項能夠包含任何字元順序。不執行百分號編碼。Location-Path 選項的值必須不能是“.”或".."。

從選項中構造一個 URI 位址可以參考章節 6.5，跳過前 5 步，且該結果是一個在請求 URI 基礎上的相對 URI 引用。注意一個通過這種方式構造的相對 URI 引用經常會包含一個絕對路徑，例如，不攜帶 Location-Path 但是提供 Location-Query 意味著在這個 URI 中從“/”開始。

用來計算相對 URI 引用的選項統一被稱為 Location-*。不僅是 Location-Path 和 Location-Query，更多的 Location-*選項可能在未來被定義，且已經在 option 的數字標記中替它們保留了 128,132,136 和 140。如果這些這些保留的選項編號出現在 Location-Path 和/或 Location-Query 之外且不被支持，必須返回 4.02（Bad Option）。

5.10.8. 條件請求選項

條件請求選項允許客戶端通知服務端，當這些在選項中包含的條件被滿足時才執行請求。

對於每個條件請求選項，如果給定的條件沒有滿足，服務端必須不能執行它所請求的方法，而必須返回 4.12（Precondition Failed）。

如果條件被滿足，服務端執行它請求的方法，和條件請求選項不存在時相同。

如果一個請求可能沒有條件請求選項，收到除了 2.xx 或者 4.xx 的狀態碼，那麼任何條件請求選項可能被忽略。

5.10.8.1. If-Match

If-Match 選項可能用來為當前實體或者為一個或多個目標資源的表現的 ETag 值生成一個條件請求。If-Match 在資源更新的請求上很有用，比如 PUT 請求，用於保護多個客戶端在同一資源下進行類似操作時意外覆蓋（比如"lost update"問題）。

If-Match 選項的值是一個 ETag 或者是一個空白字串。一個帶有 ETa 的 If-Match 選項需要和對應表現的 ETag 完全相同。值為空的 If-Match 選項和所有存在的表現匹配（如，把目標資源任何當前表現的實體作為前置）。

If-Match 選項可能多次出現。如果任意一個條件匹配，那麼這個條件成立。

如果存在一個或多個 If-Match 選項，但沒有一個選項匹配，那麼這個條件不成立。

5.10.8.2. If-None-Match

If-None-Match 選項可能被用於當目標資源不存在時，生成一個條件請求。If-None-Match 對於創建請求，比如 PUT 請求來說很有用，能夠保護多個客戶端在同一資源下進行類似操作時意外覆蓋。If-None-Match 選項沒有 value 值。

如果目標資源存在，這個條件不成立。

(注意，將 If-Match 和 If-None-Match 選項放在一個請求中不太好，因為這會導致條件永不成立。)

5.10.9. Size1 選項

Size1 選項提供在一個請求中資源表現的長度信息。選項的值是一個整型，表示字節數。它主要用於塊傳輸[BLOCK]。在目前標準中，它用於狀態碼碼 4.13（參見章節 5.9.2.9），定義服務端能夠處理的請求實體的最大長度。

6. CoAP URIs

CoAP 中使用"coap"和"coap"的 URI scheme 來標識 CoAP 資源和提供資源定位。CoAP 資源由潛在的在某個 UDP 埠監聽 CoAP 請求("coap")或 DTLS 加密的 CoAP 請求("coap")的 CoAP 原始伺服器組織和分層。CoAP 伺服器是通過通用語法中的 authority 元件，即 host 元件（即 IP 位址或功能變數名稱）和一個可選的 UDP 埠號來區分的。URI 中剩餘部分則標識了一個能夠被 CoAP 協定中定義的方法操作的資源。"coap"和"coaps"的 URI 可以分別和"http"以及"https"的 URI 做類比。

"coap"和"coaps"URI scheme 的語法在本章被定義，採用 ABNF 格式（Augmented-Bacus-Naur Form）[RFC5234]。關於"host", "port", "path-abempty", "query", "segment", "IP-literal", "IPv4address"和"reg-name"的定義請參考[RFC3986]。

實現注意：不幸的是，到目前為止 URI 格式已經非常複雜。建議開發者們仔細查看[RFC3986]。例如，IPv6 地址上的 ABNF 就比預期的更複雜。同樣的，開發者們需要小心處理 URI 的百分比解碼或百分比編碼的處理，在從一個 URI 和它的解碼元件之間只執行一次。百分號編碼對資料透明相當重要，但是處理不好可能會導致未定義的結果，例如在 path 元件中的斜線。

6.1. coap URI Scheme

coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]

如果 host 元件提供的是一個 ip 如 ipv4 位址（例如 192.168.1.1），那麼 CoAP 伺服器可以通過該 ip 位址訪問。如果 host 是一個功能變數名稱，由於功能變數名稱是一個間接的標識，因此端需要使用功能變數名稱解析服務如 DNS，來發現 host 的真正地址。host 必須不能是空；如果收到一個 authority 或者 host 為空的 URI，那麼必須認為這是一個非法的 URI。port 元件則代表 CoAP 服務端可以在 UDP 的哪個埠被訪問。如果為空或者未提供，則使用默認的 5683 埠。

path 則在一個 host 和 port 範圍內定義了資源。它由一系列被"/"（U+002F）分隔開的路徑段組成。

請求用來對資源進行進一步優化。它們由"&"（U+0026）分隔。一個參數經常由"key=value"格式組成。

"coap" URI 還支援路徑首碼"/.well-known/"，它在[RFC5785]中定義為 host 命名空間裡的“well-known locations”。它允許對一個 host 的 policy 或其他資訊（"site-wide metadata"）的發現，例如 hosted 資源（章節 7）。

建議應用開發者們採用儘量簡短而清晰的 URI。由於 CoAP 經常使受到頻寬和功耗的限制，所以應該優先考慮簡短，但又不忽略清晰性。

6.2. coaps URI Scheme

在上一節所列舉的"coap"的要求也適用於"coaps"，不同的在於，"coaps"默認埠為 5684，且 UDP 資料封包必須通過 DTLS（章節 9.1）加密。

"coaps"請求的緩存回復在章節 11.2 討論。

在"coaps"下獲取的有效資源和"coap"下不是相同的，即使"coap"和"coaps"可能採用相同的 host 和埠，它們命名空間相互獨立，且可視為獨立的原始伺服器。

6.2.1. 標準化和比較規則

由於"coap"和"coaps"符合 URI 通用語法，它們的 URI 的標準化和比較規則採用上述預設描述，符合在[RFC3986]中的演算法，參見章節 6。

如果埠和默認的埠一致，通常可以省略埠子元件。類似的，path 元件為空等同於絕對路徑"/"，因此通常做法是將 path 的值替換為"/"。scheme 和 host 對大小寫不敏感（即 EXaMpLE 等價於 example），但通常採用小寫；ip 欄位則採用[RFC5952]的方式；其他所有組件則是大小寫敏感的。除了保留的字元以外，都等價於其對應的百分號編碼（參見[RFC3986]，章節 2.1）；通常做法是不對它們編碼。

舉個例子，下面的幾個 CoAP 消息的 URI 是等價的：

coap://example.com:5683/~sensors/temp.xml
coap://EXAMPLE.com/%7Esensors/temp.xml
coap://EXAMPLE.com:/%7esensors/temp.xml

6.3. 將 URI 解碼為選項

將請求的 URI 解析成 option 的步驟如下所述。經過這些步驟後，要麼生成包含 0 個或多個 Uri-Host、Uri-Port、Uri-Path 和 Uri-Query 選項的請求，要麼會失敗。

1. 如果 URI 不是一個絕對 URI([RFC3986])，那麼解析失敗。

2. 採用[RFC3986]中的解決方案來處理 URL。在這一步，即使在第 5、8、9 步後會被解碼成 UTF-8[RFC3629]，URL 是 ASCII 編碼[RFC0020]。

注意：不用關心這個 URL 是和誰相關的，因為我們知道此時它是一個絕對 URL。

3. 如果被轉化成為小寫 ASCII 字元後的 URL 不含有值為"coap"或者"coaps"的 scheme 組件，那麼解析失敗。

4. 如果 URL 有一個 fragment 元件，那麼解析失敗。

5. 如果 URL 中的 host 元件不是以 ip 字串格式組成的請求端目的 ip 位元元址，那麼要包含一個 Uri-Host 選項，並且讓選項的值和 host 元件的值相同，轉化為 ASCII 小寫字母，然後將所有百分號編碼轉化為為對應的字元。

注意：通常來說請求的目的 ip 位元元址是從 host 獲取，它保證了 Uri-Host 選項僅用於 host 元件，格式為 reg-name。

6. 如果 URL 有 port 元件，那麼將 port 的值解析成十進位整型，否則，port 採用預設值。

7. 如果 port 的值和請求的目的 UDP 埠不一致，增加 Uri-Port 選項並將其值置為 port 的值。

8. 如果 URL 中的 path 元件為空或者只有一個“/”（U+002F），則進行下一步；

   否則，針對 path 元件中的每個段，都需要包含一個 Uri-Path 選項，且將選項的值（將百分號編碼轉換為對應的字元後）置為這個段的值（不含分隔符號號號）。

9. 如果 URL 有 query 元件，那麼，針對每個 query 元件中的參數，都需要包含一個 Uri-Query 選項，並且讓選項的值（將百分號編碼轉換為對應的字元後）為該參數的值（不包含“？”和“&”）。

注意，這些規則可以完全解析任何百分比編碼。

6.4. 將選項編碼成 URI。

將請求選項轉化為 URI 的步驟如下。經過這些步驟後，要麼會生成一個 URI，要麼失敗。在這些步驟中，百分號編碼一個字元意味著將每個 UTF8 編碼字元轉化成由一個"%"開頭的、兩位元組的十六進位數,其中 A-F 是大寫的(如章節 2.1 中採用的[RFC3986]定義，為了減要麼少複雜度，CoAP URI 中，百分號編碼中的十六進位數必須為大寫）。unreserved 和 sub-delims 的定義請參考[RFC3986]。

1. 如果請求用 DTLS 加密，則 URL 開頭用"coaps://"，否則用"coap://"。

2. 如果請求包含了 Uri-Host 選項,則將 URL 中的 host 的值置為該選項的值，所有非 ASCII 的字元轉化成百分號編碼。如果 host 不是一個合法的功能變數名稱格式或者 ip 格式，將導致失敗。如果請求沒有 Uri-Host 選項，將 host 的值置為請求的目的 ip 位元元址（V4 或 V6 格式）。

3. 添加 host 到 URL。

4. 如果請求中有 Uri-Port 選項，則將 port 元件的值置為該選項的值。否則採用請求的目的端 UDP 埠。

5. 如果 port 不是該 scheme 默認的埠號，則採用"："+port 值的方式載入 URL 後面。

6. 將 resource name 清空。對請求中的每個 Uri-Path 選項，採用“/”+選項的值跟隨在 resource name 後面。在此之前，需要先將不在 unreserved 集、sub-delims 集、不為"："、"@"的字元轉化為百分號編碼。

7. 如果 resource name 為空，將它設置為"/"。

8. 對請求中的每個 Uri-Query 選項，採用"？"（第一個選項）或"&"（後續選項）+選項值的形式編碼到 resource name 中（同樣的,在此之前,需要先將不在 unreserved 集、sub-delims 集、不為"："、"@"、 "/"、 "？"的字元轉化為百分號編碼）。

9. 將 resource name 添加到 URL 中。

10. 返回 URL。

注意：這些步驟旨在以標準形式構建一個 URI（參見章節 6.3）。

7. 發現

7.1. 服務發現

作為 CoAP 服務端提供的發現服務的一部分，用戶端需要對服務端有所瞭解。

用戶端發現服務端，是用戶端通過從 URI 中獲取或學習服務端命名空間中的資源來做到的。另外，用戶端能夠使用多播 CoAP（章節 8）和"All CoAP Nodes"多播位址來查找 CoAP 服務端。

除非"coap"或者"coaps"URI 中指定了服務端的 UDP 埠，否則服務端預設是能夠通過默認埠連接的。

提供資源發現（章節 7.2）的服務端必須支援，提供其他資源的服務端應該支援 CoAP 預設埠號 5683。對於 DTLS 加密類型的 CoAP，提供資源發現和提供其他資源的服務端可能支援預設 5684 埠。此外，其他端可能採用另外的埠，例如，埠動態分配情況下。

實現注意：當一個 CoAP 服務端是由 6LowPAN 節點提供，當埠使用[RFC4944]和[RFC6282]定義的 UDP 埠壓縮方式，採用 61616-61632 之間的埠號時，頭部的壓縮會更好。注意，如果某個服務端的 UDP 埠和默認埠不同，可以將它們（採用非默認埠的和採用默認埠的）視為兩個不同的端。

7.2. 資源發現

CoAP 端提供的資源發現在 M2M（machine-to-machine）應用中相當有用，因為在其中沒有人的幹預，而固定的介面又會導致連接的難複用性和較差的魯棒性。為了最大化 CoRE 環境的互用性，

一個CoAP端應該支持[RFC6690]描述的可發現資源的CoRE連接格式，除非要求完全手動配置。由服務端來決定哪些資源能夠被發現（如果有的話）。

7.2.1. 'ct'特性

這節定義了一種採用[RFC6690]的新的 web 連接（Web Linking [RFC5988]）特性。Content-Format 碼"ct"特性提供了該資源會返回哪種 Content-Format 的提示。注意，由於這僅僅是一個暗示，並不會取代針對某個資源表現的請求的回復中的 Content-Format 選項。"ct"特性的值是採用 CoAP 編碼格式的十進位整型 ASCII 碼，而且必須在 0-65535 範圍內。舉個例子，"application/xml"被定義成"ct=41"。如果 Content-Format 特性不存在，那麼該類型無意義。Content-Format 碼特性可能包含一個用空格分隔開的 Content-Format 碼序列，表明多個 content-format 可用。這個特性的值的語法在下面被總結，其中"cardinal", "SP", 和 "DQUOTE"在[RFC6690]中定義。

```
ct-value = cardinal
         /  DQUOTE cardinal *( 1*SP cardinal )DQUOTE
```

表 12

8. 多播 CoAP

CoAP 支持在 IP 多播組中發送請求，這相當於連續的單播 CoAP。更多關於多播組內 CoAP 的交互討論在[GROUPCOMM]。

那些希望其他端能夠發現的 CoAP 端，採用多播服務發現，加入一個或多個適當的 all-CoAP-node 的多播位址（12.8 節）並且監聽默認 CoAP 埠（注意，這些端可能會收到其他多播位址的多播請求，包含 all-nodes 的 IPV6 位址或者 IPV4 廣播）。因此一個端必須做好接收這些消息的準備，但如果沒有提供多播發現服務的話，可以忽略它們。

8.1. 消息層

多播請求的特點是目的地址由具體的CoAP端位址變成了IP多播位址，多播請求必須是不需應答消息。

服務端應該能夠識別出這些通過多播發來的請求，例如，儘量使用現有的 API，如 IPV6_RECVPKTINFO [RFC3542].

為了避免錯誤的回應，當服務端發現該請求是通過多播接收的，必須不能返回 RST 消息給 NON。如果沒有發現（為多播請求），服務端可能會返回一個 RST 消息給 NON。由於這種 RST 消息看起來和發送者發送的單播消息相同，發送者必須避免使用一個可能依然在其他接收到多播消息的端中依舊使用的 Message ID。

在撰寫本文的時侯，多播消息只能用於不含 DTLS 的 CoAP。因此意味著在本文中為 CoAP 定義的安全模式並不適用於多播。

8.2. 請求/回應層

當服務端發現某個多播請求，它可能會一直忽略這個請求，特別是當它沒有任何有用的回復（比如說，它只有一個空的有效負載或者一個錯誤回應）。這個判斷是由應用決定的（例如，在請求過濾 [RFC6690]中，當過濾條件不滿足時，服務端不會回應多播請求。更多例子見[GROUPCOMM]）。

如果服務端決定回應一個多播請求，它不應該立即回應。相反它應該會等待一段時間才進行回應。我們把這段時間稱為空閒（Leisure）時間。閒置時間的值可能由應用決定，也可能由下面的描述中得到。服務端應該在選取的閒置時間中的某個隨機時間發送一個單播回應給該多播請求。如果再次收到同一個多播組的請求，一個新的閒置時間最早需在上一個閒置時間結束後才能開始。

為了計算出一個閒置時間，服務端必須有一個組大小估計 G，一個目標資料傳輸率 R（二者都必須謹慎的選擇），一個估計的回復大小 S，一個大致的閒置時間計算公式如下：

$$lb\_Leisure = S * G / R$$

舉個例子，一個在 2.4G 頻段，採用 IEEE 802.15.4（6LoWPAN）的本地網路中，G 可能被設置成 100，S 為 100 位元組，目標速率 R 為 8kbits/s = 1kB/s。那麼閒置時間為 100 * 100 / 1000 = 10 秒。

如果一個 CoAP 端不能獲得足夠的資料來計算閒置時間，它可能採用 DEFAULT_LEISURE。

當匹配一個多播請求的回復時，僅僅 token 必須要匹配。回復的源端不需要（可能也不會）和原始請求的目的端匹配。

為瞭解釋表現中的 Location-*選項和任何嵌入的連結，請求 URI（例如，回應對應的基本 URI）通過將原始請求 URI 的 host 元件中的多播位址替換成實際回應的端的 ip 地址。

8.2.1. Caching

當用戶端發出多播請求，它經常生成一個新的請求給多播組（因為組中可能有新成員，或者沒有收到之前的請求）。它可能將接收到的回復的緩存更新。然後，它同時採用 cached-still-fresh 和新的回應作為請求的回應。

向一個多播組發送的 GET 請求的回應，可能被用於滿足相關的單播請求 URI 的後續請求。單播請求 URI 中的 authority 部分即回復包中的傳輸層源位址。

通過發送一個 get 請求給涉及到的單播請求 URI，一個回應回復可能重新生效。

向多播組發送的 get 請求必須不能包含 ETag 選項。抑制用戶端已有的回應包的機制仍有待進一步研究。

8.2.2. 代理

當一個正向代理收到一個帶有 Proxy-Uri 或從 Proxy-Scheme 中構建的 URI，且其為多播位元址，代理如上所述，會獲得一個回應集合，並且會發送所有的所有的回應包（包括 cached-still-fresh 的和新的）給原始用戶端。

協定並沒有提供對被修改的單播請求 URI（基本 URI）的返回方式，因此轉發。轉發多播請求在[GROUPCOMM]中有更仔細的討論，一種定位基本 URI 的方案可在[CoAP-MISC]第 3 章中找到。

9. 安全 CoAP

本章節定義 CoAP 中的 DTLS 綁定。

在配置階段，要為 CoAP 設備提供它需要的安全資訊，包括金鑰卡片和存取控制表。本規範在章節 9.1.3.2.1 中定義了 RawPublicKey 模式下的配置。在配置的最後階段，設備會處於下面四種安全模式下的一種，並帶有模式的附屬資訊。本規範中 NoSec 和 RawPublicKey 模式都是強制執行的。

NoSec：沒有協定層的安全(禁用 DTLS)。合適的情況下，應當使用其他技術來提供底層的安全機制。在[IPsec-CoAP]中討論 IPsec 的使用。某些使用受限節點的鏈路層也提供鏈路層安全機制，它可能適用於合適的金鑰管理。

PreSharedKey：DTLS 被啟用，有一個預先共用的金鑰列表 [RFC4279]，而且每個金鑰都包括節點列表(參見章節 9.1.3.1)。極端情況下，一個金鑰對應一個節點(1:1 node/key 比例)。如果兩個以上的實體共用一個特定的預先共用金鑰，那麼該金鑰只把實體認證為那個組的成員，而不是單獨的一個物件。

RawPublicKey：DTLS 被啟用，設備有一個非對稱密匙對，但是沒有證書（一個原始公開金鑰），它是使用 out-of-band 機制 [RFC7250]來驗證的，如章節 9.1.3.2 中所述。該設備也有一個從公開金鑰計算來的身份和一個可以溝通的節點的身份列表。

Certificate:DTLS 被啟用，設備有一個帶有 X.509 證書[RFC5280]的非對稱金鑰對，與它的目錄綁定，並由一些常見的受信任的根憑證授權頒發，如章節 9.1.3.3 所述。設備也有根信任錨的列表，可用於驗證證書。

在"NoSec"模式下，系統只需向某個 IP 和埠發送帶"coap" scheme 頭的 UDP 資料封包即可。但只有當攻擊者不能利用 CoAP 節點收發包時才是安全的；在 11.5 節中有關於這個問題的附加描述。

另外 3 種安全模式都利用 DTLS 實現，並且由"coaps" scheme 和 DTLS 下的 CoAP 預設埠標識。這是是一個能用於認證(帶有安全模型的限制)的安全關聯，基於這個認證，可以授權其他的通信端。CoAP 本身不提供用於認證或者授權的協定原語；當需要的時候，由通信安全(比如 IPSec 或 DTLS)或者物件安全(帶有負載)提供。在特定操作需要認證的設備通常使用這兩種安全方式。在涉及中間人的地方，只有當中間人是信任體系中的一部分時才能保證通信安全。CoAP 並沒有提供一種方法來轉發不同級別的授權，用戶端可能與其他中間人或原始伺服器有這些授權，因此可能需要在第一個中間人的時候就要執行所有授權。

9.1. DTLS-Secured CoAP

如同 HTTP 在 TCP 中使用 TLS 來保證安全一樣，CoAP 在 UDP 之上使用 DTLS[RFC6347]來保證安全(圖 13)。本節定義了 CoAP 綁定到 DTLS 和適合受限環境的必須強制執行最低的配置。綁定是通過一系列的單播 CoAP 增量來定義的。實際上，DTLS 就是 TLS 和附加功能來處理 UDP 傳輸的不可靠特性。

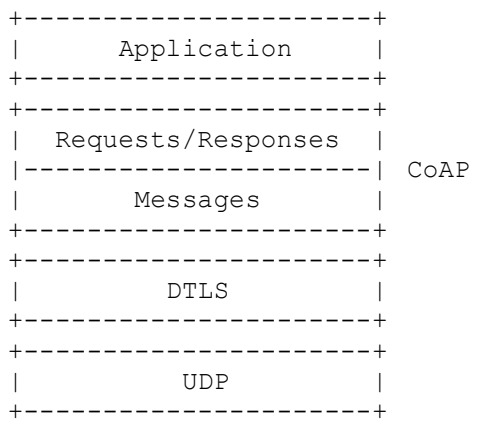


圖 13: dtls 安全 CoAP 的分層

在一些受限節點(有限的 flash 和 RAM)和網路(受限頻寬或者高可擴充性要求)中，依賴於正在使用的特定密碼組合，DTLS 的所有模式可能是不可用的。一些 DTLS 密碼組合在設置安全關聯時，可能增加一些複雜的重要實現以及一些所需的初始信號交換開銷。一旦完成最初的握手，DTLS 添加一個大約 13 位元組的有限的 per-datafgram 開銷，不包括任何的初始化向量/亂數(例如 8 個位元組的 TLS_PSK_WITH_AES_128_CCM_8[RFC6655]),完整性檢查值(例如 8 個位元組的 TLS_PSK_WITH_AES_128_CCM_8[RFC6655]),和一些密碼組合要求的填充值。考慮到可能適用的特定密碼組，應該仔細權衡 DTLS 的給定模式是否適用於以 CoAP 為基礎的應用程式，會話維護是否與應用流程相容，用於約束節點和額外的網路開銷的資源是否足夠。(對於一些使用 DTLS 的模式，本規範確定了一個強制執行的金鑰組。當這些金鑰組的確合適時，能實現最大化的互通性。應用的特定安全性原則可以決定使用的實際金鑰組的實際設置)。DTLS 不適用於組金鑰(多播通信)；然而，它可能是未來組金鑰管理協定中的一部分。

9.1.1. 消息層

一個端能夠作為 CoAP 用戶端也應該能夠作為 DTLS 用戶端。它應該在合適的埠向服務端發起會話。當 DTLS 的握手結束時，用戶端可能發起第一個 CoAP 請求。所有的 CoAP 消息必須當做 DTLS"應用資料"發送。

為了將 ACK 消息或 RST 消息匹配到 CON 消息，或者 RST 消息匹配到 NCON 消息，附加上下面的規則：DTLS 會話必須一致，時間段必須一致。

當消息在發送時有一致的 DTLS 會話、一致的時間段和相同的消息 ID，那麼消息就是一致的。

注意：當重傳一個 CON 消息，儘管 CoAP 的消息 ID 一致，但每次嘗試都會使用一個新的 DTLS 序號。因此接收者必須按照章節 4.5 中描述的去執行重復資料刪除。重傳不能跨時間段執行。

在 RawPublicKey 和認證模式中的 DTLS 連接被設置為使用相互驗證，因此它們能在兩個方向上維持連接並重用於未來的消息交換。當設備需要恢復資源時，設備可以關閉一個 DTLS 連接，但是通常它們必須盡可能保持長連接。在每次 CoAP 消息交換後關閉 DTLS 連接是很低效的。

9.1.2. 請求/回應層

為了將回應匹配到請求上，加上下面的規則：DTLS 會話必須一致，時間段必須一致。

這意味著對於一個 DTLS 安全請求的回應必須一直使用相同的安全會話和時間段。任何想提供一個 DTLS 請求的 NoSec 回應的嘗試並不匹配這個請求，因此必須被拒絕（除非它匹配一個無關的 NoSec 請求）。

9.1.3. 端點身份

按照[RFC6066]的章節3的定義，設備應當支援伺服器名稱指示(SNI)來指示在SNI主機名稱稱域中的授權。有了這個特性，當一個作為多個權威的虛擬伺服器的主機接收到新的DTLS連接時，它知道這個DTLS會話要使用哪個金鑰。

9.1.3.1. Pre-Shared Keys

當形成一個新節點的連接時，系統根據它試圖到達哪些節點來選擇一個適當的金鑰，然後使用 DTLS PSK（預共用金鑰）形成一個 DTLS 會話。在這些模式中的執行必須支援強制執行密碼組 TLS_PSK_WITH_AES_128_CCM_8，如[RFC6655]中所規定的。

根據調試模型，應用可能需要為身份提示定義應用規範(參見[RFC4279]的 5.2 節)，以支援 PSK 身份提示的使用。

應用了[RFC4279]中章節 7 的安全性考慮。應用應當仔細權衡它是否需要完全正向保密(PFS)，並且選擇一個合適的金鑰組([FRC4279]的 7.1 節).PSK 的資訊熵必須足夠應付強力攻擊(PSK

不是隨機選擇的而是人為選擇的)和字典式攻擊([RFC4279]的 7.2 節)。用戶端身份的明文通信可能會洩露資料或者隱私([RFC4279]的 7.3 節)。

9.1.3.2. 原始公開金鑰證書

在這種模式下，設備有個非對稱金鑰對但是沒有 X.509 證書(原始公開金鑰)；例如，非對稱秘鑰對是由廠商生成並安裝在設備上(參見章節 11.6)。一個設備可能配置了多個原始公開金鑰。原始公開金鑰的類型和長度取決於所使用的金鑰組。在 RawPublicKey 模式中執行必須支援強制執行的密碼組 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8，如[RFC7251]，[RFC5246]和[RFC4492]中所規定的。使用的秘鑰必須帶有 ECDSA。curve secp256r1 必須支持[RFC4492]；這和 NIST P-256 curve 相等。這個雜湊演算法是 SHA-256。實現必須使用 Supported Elliptic Curves 和被支援的點格式擴充[RFC4492]；必須支援未壓縮的點格式；[RFC6090]可以作為一個實現方法。一些有關實現這個金鑰組的指導可以在[W3CXMLSEC]中找到。使用原始公開金鑰與 TLS 的機制在[RFC7250]中有規定。

實現注意：這意味著在圖 14 中列出的帶有至少一個值的擴充將出現在 DTLS 握手中。

```
Extension: elliptic_curves
 Type: elliptic_curves
 (0x000a) Length: 4
 Elliptic Curves
 Length: 2 Elliptic
 curves (1 curve)
   Elliptic curve: secp256r1 (0x0017)

Extension: ec_point_formats
 Type: ec_point_formats
 (0x000b) Length: 2
 EC point formats Length: 1
 Elliptic curves point formats
 (1)
   EC point format: uncompressed (0)

Extension: signature_algorithms
 Type: signature_algorithms
 (0x000d) Length: 4
 Data (4 bytes): 00 02 04 03
   HashAlgorithm: sha256 (4)
```

```
SignatureAlgorithm: ecdsa (3)
```

Figure 14: DTLS Extensions Present for TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8

9.1.3.2.1. 配置

RawPublicKey 模式被設計成可輕易配置在 M2M 部署裡。假定每個設備有一個已經安裝好的非對稱公開金鑰對。在[RFC6920]的章節 2 中描述了端點從公開金鑰中計算出識別字。所有支持檢查 RawPublicKey 識別字的執行必須至少支援 sha-256-120 模式 (SHA-256 截斷為 120 位).實現也應當支援更長的長度識別字和可以支援更短的長度。請注意更短的長度在面對攻擊時提供更少的安全保護，因此不推薦使用。

通過 URI、二進位、與/或人可識別的格式，識別字被給到驗證它們的系統。 [RFC6920]。所有的實現應當支援二進位元模式，並且包含使用者介面的實現也要支援人可識別的格式。

在配置期間，收集每個節點的識別字，例如通過讀取設備外部的條碼或者獲取識別字的預編譯列表。這些識別字安裝在相關的節點中，例如一個 M2M 的資料收集服務端。識別字用於兩個目的，與端點連接更多的設備資訊和執行存取控制。在（最初的和進行的）配置期間，識別字的存取控制清單也應該安裝和維護，用這些識別字，設備可以開始 DTLS 會話。

9.1.3.3. X.509 證書

在證書模式下實現必須支援強制實施的密碼組 TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 如[RFC7251]，[RFC5246]和[RFC4492]中所規定。即證書包括 SubjectPublicKeyInfo，它會顯示帶有 danamedCurves secp256r1 的 id-ecPublicKey 的演算法 [RFC5480]；公開金鑰格式是未壓縮的 [RFC5480]；雜湊演算法是 SHA-256；如果包括的話，密匙使用擴充顯示數位簽章。證書必須使用 secp256r1 簽署 ECDSA，而且簽名必須使用 SHA-256。使用的密匙必須是 ECDSA capable。必須支持 curve secp256r1[RFC4492；這個曲線和 NIST P-256 曲線是

等效的。雜湊演算法是 SHA-256。實現必須使用 Supported Elliptic Curves 和 Supported Point Formats Extensions[RFC4492]；必須支援未壓縮的點格式；[RFC6090]可以作為一個實現方法。

證書的主語會從該設備長期的惟一識別字中建立，例如 EUI-64 [EUI64]。這個主語也可以基於完全限定功能變數名稱(FQDN)，它是被用作 CoAP URI 的主機部分。然而，設備的 IP 位址一般不能當做主語，因為它是可變的。系統中的發現過程會建立給定設備的 IP 位址和每個設備主語之間的映射。一些含有多於一個主語的設備也必須包含多個證書。

當生成一個新的連接，遠端設備的證書就需要驗證。如果 CoAP 節點有一個絕對時間的來源，那麼節點應當檢查證書的有效時間在範圍之內。為了達到安全要求，證書必須被驗證為適合於安全需求，使用的功能相當於[RFC5280]章節 6 中指定的演算法。如果認證包含一個 SubjectAltName，那麼請求 URI 的授權必須匹配至少一個在 SubjectAltName 族中 URI 類型域中的任何 CoAP URI 的授權。如果在證書中沒有 SubjectAltName，那麼除了有萬用字元的證書不被允許以外，請求 URI 的授權必須匹配證書中的通用名(CN)，使用[RFC3280]中定義的匹配規則。

證書狀態檢查的核心支援需要進一步研究。由於線上證書狀態協定(OCSP)[RFC6960]到 CoAP 的映射當前沒有被定義，而且 OCSP 也可能不是很容易適用於所有環境，所以另一種方法可能是使用 TLS 證書狀態請求擴充([RFC6066]章節 8;也被稱為“OCSP 裝訂”）或最好是多個證書狀態擴充（[RFC6961]），如果可用的話。

如果系統除了證書外還有一個共用金鑰，那麼應該使用一個包括共用金鑰（如 TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA[RFC5489]）的密碼組。

10. CoAP 和 HTTP 的跨協定代理

CoAP 支援 HTTP 功能的有限子集，因此從 CoAP 代理到 HTTP 是很簡單的。在 CoAP 和 HTTP 之間採用代理可能有幾個原因，例如，在設計一個這兩種協定都可以使用的網站介面，或在實現

CoAP-HTTP 代理時。同樣地，CoAP 也可以代理到諸如 XMPP[RFC6120] 或者 SIP[RFC3264]等協定；這些代理機制的定義超出本規範的範圍。

通過一個正向代理來訪問資源有兩個可能的方向：

CoAP-HTTP 代理：通過一個中間人使得 CoAP 用戶端訪問 HTTP 服務端的資源。這是通過在 CoAP-HTTP 代理的 CoAP 請求裡包含帶有“http”或“https”URI 的 Proxy-Uri 或 Proxy-Scheme 選項發起的。

HTTP-CoAP 代理：通過一個中間人使得 HTTP 用戶端訪問 CoAP 服務端的資源。這是通過在 HTTP-CoAP 代理的 HTTP 請求的 Request-Line 中指定“coap”或“coaps”URI 發起的。

無論哪種方式，只有 CoAP 的請求/回應模型被映射到 HTTP。CON 或者 NON 消息等模型應該是透明的，對代理功能沒有影響。下面的章節描述對正向代理的請求的處理。沒有提及反向代理，因為代理功能對用戶端是透明的，就相當於原始伺服器一樣。然而，對反向代理的考慮和對正向代理的考慮應該是一樣的，而且通常會期望反向代理以與正向代理類似的方式運行。實現中需注意，HTTP 用戶端函式程式庫沒有提供一種方法將 CoAP URI 放在 HTTP 的 request-Line 中，使得操作 HTTP-CoAP 正向代理變得困難；反向代理可能因此有更好的適用性。另外一份規範會定義例如 HTTP-CoAP 反向代理的的 URIs 操作的規定[MAPPING]。

10.1. CoAP-HTTP 代理

如果‘http’或‘https’URI 中的請求包含 Proxy-Uri 或者 Proxy-Scheme 選項[RFC2616]，那麼接收的 CoAP 端點(今後稱為“代理”)要求對指明的 HTTP 資源執行請求方法中指定的操作，並向用戶端返回結果。(可參考章節 5.7，如何生成包含安全要求的代理請求)

這一節為所有 CoAP 請求指定了代理應該返回到用戶端的 CoAP 回應。代理實際如何回應請求是一個實現細節，期望的典型情況是代理轉發請求到 HTTP 源服務端。

由於 HTTP 和 CoAP 共用基本的請求方法集，因此在 HTTP 資源上執行 CoAP 請求與在 CoAP 資源上執行它並沒有什麼不同。本節的下麵的小節中將解釋在 HTTP 資源上執行的每個 CoAP 方法的含義。

如果代理不能或者不願服務帶有 HTTP URI 的請求，那麼向用戶端返回 5.05(Proxying Not Supported)回應。如果代理通過與協力廠商交互(例如 HTTP 原始服務端)來服務請求，並且無法再合理的時間內獲得結果，返回一個 5.04(GateWay Timeout)的回應；如果可以獲取結果但是不能解釋該結果，返回 5.02(Bad Gateway)的回應。

10.1.1. GET

GET 方法請求代理返回一個由請求 URI 定位的 HTTP 資源表現。

一旦成功了，必須返回 2.05(Content)回應碼。回應的有效負載必須是目標 HTTP 資源的表現，而且必須設定相應的 Content-Format 選項。回應必須攜帶 Max-Age 值，這個值不大於該資源的刷新剩餘時間。如果 HTTP 實體有一個實體標記，代理必須在回應中包含 ETag 選項，並且按照下面的描述處理請求中的 ETag 選項。

用戶端可以通過包含下面的選項來影響 GET 請求的流程：

Accept：請求可以包含一個 Accept 選項，標識優先的回應內容格式。

ETag：請求可以包含一個或多個 ETag 選項，標識用戶端存儲的回應。這就要求當代理需要發送 2.03(Valid)回應時，轉而發送帶有請求的實體標記的 2.05(Content)回應。請注意 CoAP ETags 是 HTTP 看來的強 ETags；CoAP 沒有 HTTP 的弱 ETags，並且在跨協定代理中沒有好的方式來使用它們。

10.1.2. PUT

PUT 方法要求代理按照請求 URI 定位的路徑來更新或者創建 HTTP 資源。

如果按照請求 URI 創建了一個新的資源，那麼必須向用戶端發送 2.01(Created)的回應。如果修改了一個已經存在的資源，那麼必須返回一個 2.04(Changed)回應來標示請求的成功完成。

10.1.3. DELETE

DELETE 方法請求代理在 HTTP 源服務端按照請求的 URI 刪除 HTTP 資源。

如果成功了或者在請求時不存在該資源，那麼向用戶端發送一個 2.02(Deleted)回應。

10.1.4. POST

POST方法要求代理將包含在請求中的表現更新到HTTP原始服務端。POST方法的實際執行結果由原始服務端決定，並且依賴於請求URI定位的資源。

如果通過POST方法執行的動作不會產生一個可以由URI來標識的資源，那麼必須向用戶端回復2.04(Changed)的回應。如果資源在源服務端被創建，那麼必須返回2.01(Created)的回應。

10.2. HTTP-CoAP 代理

如果一個 HTTP 請求包含帶有“coap”或者“coaps”URI 的 Request-URI，那麼接收的 HTTP 端(今後稱為“代理”)要求對指明的 CoAP 資源執行請求方法中指定的操作，並向用戶端返回結果。

本小節定義了針對任何 HTTP 請求，代理應該向用戶端返回的回應。除非另有說明，所有的申明都是推薦的行為，在一些特別受

限的實現需要使用簡化方式。代理如何回應請求是一個執行細節，然而期望的典型情況是代理轉發請求到 CoAP 原始服務端。在下面的小節中將解釋在 CoAP 資源上執行的每個 HTTP 方法的含義。

如果代理不能或者不想服務帶有 CoAP URI 的請求，將會向用戶端返回 501(Not Implemented)回應。如果代理通過與協力廠商交互(比如 CoAP 原始服務端)來服務請求，並且無法再合理的時間內獲得結果，返回一個 5.04(GateWay Timeout)的回應；如果可以獲取結果但是不能解釋該結果，返回 5.02(Bad Gateway)的回應。

10.2.1. OPTIONS and TRACE

由於 CoAP 不支援 OPTIONS 和 TRACE 方法，必須向用戶端返回 501(Not Implemented)錯誤。

10.2.2. GET

GET 方法要求代理返回由 Request-URI 定位的 CoAP 資源的表現。

一旦成功，返回 200(OK)回應碼。回應的有效負載必須是目標 CoAP 資源的表現，而且必須設定相應的 Content-Type 和 Content-Encoding 域。回應必須攜帶 Max-Age 值，這個值不大於該資源的刷新剩餘時間。如果 CoAP 回應有一個 ETag 選項，代理必須在回應中包括一個 ETag 頭。

用戶端可以通過包含下面的選項來影響 GET 請求的流程：

Accept：在請求中的 HTTP Accept 頭的優先採用的媒體類型被映射到一個 CoAP Accept 選項。HTTP Accept 媒體類型的範圍，參數和擴充都不被 CoAP 的 Accept 選項支援。如果代理不能找到雙方（服務端和用戶端）同時可以接受的的 Accept 域，那麼代理將發送 406 回應。代理可能使用從 HTTP Accept 頭的其他媒體類型來重新發起請求。

Conditional GETS：有條件的 HTTP GET 要求能夠被映射到對應 CoAP 請求的的“If-Match”或“If-None-Match”請求頭域。“If-Modified-Since”和“If-Unmodified-Since”請求頭域不能直接被 CoAP 支持，但是能夠被緩存代理在本地執行。

10.2.3. HEAD

除了服務端必須不能在回應中返回一個消息體，HEAD 方法與 GET 是一樣的。

儘管在 CoAP 中沒有與 HTTP 的 HEAD 方法直接等效的方法，HTTP-CoAP 代理能夠對 CoAP 資源的 HEAD 請求作出回應，並且返回不帶消息體的 HTTP 頭。

實現注意：HTTP-CoAP 代理可能試著使用[BLOCK]傳輸選項來減小每次傳輸的資料量，但是需要注意原始服務端是否支援塊傳輸。

10.2.4. POST

POST 方法要求代理將包含在請求中的表現更新到 CoAP 原始服務端。POST 方法實際執行的功能由原始服務端決定，並且依賴於請求 URI 指定的資源。

如果通過 POST 方法執行的動作不會產生一個可以由 URI 來標識的資源，那麼必須向用戶端返回一個 200(OK)或者 204(No Content)的回應。如果在源服務端已經創建了資源，那麼必須返回 201(Created)回應碼。

如果在 CoAP 回應中有任何的 Location-*選項，將返回由這些選項的值構造的 Location header 域。

10.2.5. PUT

PUT 方法要求代理更新或者創建由請求 URI 定位的 CoAP 資源。

如果根據請求 URI 創建了一個新的資源，那麼向用戶端返回 201(Created)回應碼。如果修改了存在的資源，發送 200(OK)回應碼或者 204(No Content)回應碼來標示請求的成功完成。

10.2.6. DELETE

DELETE 方法要求代理在 CoAP 原始服務端刪除由請求 URI 定位的 CoAP 資源。

如果操作成功，那麼向用戶端發送 200(OK)的回應；如果請求時不存在該資源，那麼向用戶端返回 204(No Content)回應。

10.2.7. CONNECT

由於沒有 TLS 到 DTLS 的通道，當前 HTTP-CoAP 代理功能不支援本方法。因此，會向用戶端返回 501(Not Implemented)錯誤。

11. 安全事項

節分析協定面臨的可能漏洞，並告知協定和應用開發者在本文中 CoAP 的安全限制。因為 CoAP 實現了 HTTP/1.1 屬性中的一個子集，在[RFC2616]的章節 15 中的安全考慮同樣與 CoAP 相關。本節集中描述 CoAP 特有的局限性。

11.1. 解析協定和處理 URIs

網路解析的應用可以顯示收包處理邏輯的漏洞。複雜的解析器可能是很多漏洞的根源，比如能夠遠端的破壞一個節點，或者甚至在節點上執行任意的代碼。CoAP 嘗試通過降低解析器的複雜性、限定可編碼的值一個範圍、主動降低在多個代表相同事件的表現上請求不常用的方法的複雜度，來減少引入這類漏洞的可能性。許多 URI 處理都放在用戶端，更進一步減少了往服務端引入漏洞的可能性。即便這樣，CoAP 執行中的 URI 處理代碼仍然可能存在許多漏洞，因此需要謹慎處理。CoAP 的存取控制的實現必須保證不因為 URI 和通過 URI 定位的資原始程式碼的差異而引入漏洞。最複雜的解析器可能是 CoRE Link Format，儘管它也可能是

按照減少執行複雜度的目標設計的[RFC6690]，也可以參考[RFC2616]的章節 15.2。

11.2. 代理和緩存

按照[RFC2616]中的章節 15.7 提到的，代理由於其本身的中間人性質,可能會破壞直接 CoAP 消息交換的 IPsec 或者 DTLS 保護。因此,它們是 CoAP 消息交換中最可能破壞機密性或者完整性的。按照[RFC2616]中提到的，它們也可能破壞可用性。

當代理採用緩存的時候，會加大請求/回應資料的機密性和完整性的威脅。注意，CoAP 沒有定義任何能夠為 HTTP/1.1 提供更好的敏感性資料保護的 cache-suppressing Cache-Control 選項。

對於包含 cache 的情況，存取控制的實現需要注意，如果請求有對應的 cache 條目，對應的 cache 資料也需要應用控制規則。這對實現多個安全域的用戶端以及服務於多個用戶端的代理很有意義。同樣，有緩存的代理必須不能轉發緩存資料給那些傳輸安全性更低的代理。

與"coap"機制不同，對“coaps”認定的請求的回應不是“public”的，因此不能重用於共用的緩存，除非緩存能夠對緩存條目進行等效的存取控制。如果 CoAP 中預設緩存了消息，它們能在私有的緩存中被重新利用。

最終,給多個原始請求端服務的代理可能發送單獨回應(和附帶回應相反)來提供額外的增幅(參考 11.3)。

11.3. 增幅的風險

CoAP 服務端通常用回應包來回復請求包。回應包可能比請求包大許多。網路攻擊者可能利用 CoAP 節點將一個小的攻擊包變為一個大的攻擊包，這個過程稱為增幅。這就存在 CoAP 節點利用協定的增幅特性產生一個 DoS 攻擊的風險：由於網路限制，攻擊者佔據的頻寬有限，但通過擴大特性，能夠使攻擊者突破頻寬限制。

由於 UDP 協定不提供驗證請求包中的源位址的方法，因此在使能 NoSec 訪問的節點中，網路攻擊者能夠訪問它，並且能夠訪問潛在的被攻擊者，這就存在風險。網路攻擊者只需要將被攻擊者的 IP 位址放在合適的請求包中的源位址中，就能產生定向到被攻擊者的更大的包。

作為降低影響的方法，很多受限制網路只能產生很小的通信量，這樣使得 CoAP 節點在攻擊中不容易被注意。然而，受限制網路的有限頻寬使得網路本身容易成為增幅攻擊中的受害者。

因此，如果請求沒有認證，在回應中不能有大的增幅因數。CoAP 服務端通過使用 CoAP 的 slicing/blocking 模式和大的資源表現採用小的分配來減少為攻擊者提供的增幅量。舉個例子，對於一個 1000 位元組的資源，一個 10 位元組的請求引起一個 80 位元組的回應(64 位元組的塊)，而不是 1016 位元組的回應，這樣就相當的減少了提供的增幅。

CoAP 也能在請求中支援多播 IP 位址的使用，這在 M2M 中是一項重要的要求。多播 CoAP 請求也許是事故或者 DoS 攻擊的源頭，尤其在受限制網路中。本規範試圖在回應返回時加一些限制來減少多播請求的增幅效果。為了減少惡意使用的可能性，CoAP 服務端不能接受沒被認證的多播請求，也對潛在的源加上一些多播的邊界限制。可能的話，CoAP 服務端應當限制對特定資源的多播請求的支援。

在提供 POSIX 介面 API[IEEE1003.1]的通用作業系統中，很難查出一個包是否指向多播位址。很多實現能夠知道自己是否已經加入多播組，採用 FF0x::1 格式指向多播位址的包會產生問題，這種包會被所有 IPV6 節點接收。實現時必須使用諸如 IPV6_RECVPKTINFO[RFC3542]的新的 API。

11.4. 地址欺騙攻擊

由於 UDP 中沒有握手機制，一個欺詐端點能夠自由的讀寫受限制網路中的消息(例如 NoSec 或 PreSharedKey 中的 nodes/key 比率

大於 1：1)，這樣就能輕易的使用下面手段來攻擊單個端點，一組端點，甚至整個網路：

1. 在對 CON 消息或者 NONCON 消息的回應中回復一個虛假的重定消息，這樣使得端點"deaf"

2. 在對 CON 消息的回應中回復一個虛假的 ACK，這樣可能會阻止 CON 消息的發送者重傳，並讓實際的回應失效。

3. 利用偽造的有效負載/options 來欺騙整個回應(有幾個不同層級的影響：從單個回應的破壞到配套基礎設施的攻擊，例如破壞代理緩存或者在資原始目錄中欺騙 validation/lookup 介面，更普遍的，任何儲存網路狀態的元件和利用 CoAP 為來處理和更新狀態的通信設備體都是一個潛在的攻擊目標。

4. 對目標節點多播欺騙請求，這會導致網路擁塞或崩潰，DoS 攻擊或者從強行喚醒。

5. 欺騙 observe 消息。

儘管沒有實現基於隨機 token 詢問的安全傳輸層,不明來源的回應攻擊可以被檢測和緩和。[RFC4086]討論了隨機詢問的安全機制。

特別的，使用 CON 消息的 CoAP 能偵測到其他類型的欺騙，因為從被欺騙的端點中有攻擊源發來的 ACK 或者 RST 消息。但是欺騙保持 message ID 的追蹤是很難的，另一方面在被攻擊之後，利用此對探測欺騙非常有用。這些攻擊可以利用安全模式而不是 NoSec 來防範。

無論有沒有源位址欺騙，用戶端能試圖往服務端發送複雜的請求來加重服務端的負載;地址欺騙使得回溯,阻塞和攻擊更加困難。由於 CON 請求的花銷很小，所以很容易執行這種攻擊。在這種攻擊下,帶有受限制電源的受限節點會比預期更快的耗盡能量(電量耗盡攻擊)。如果用戶端利用 CON 消息和伺服器利用分離的 CON 回應一個不會回應的位址(可能是欺騙)，伺服器就需要為其分配記憶體和對每一個未回應的節點進行回應這回耗盡服務為

的 MAX_TRANSMIT_SPAN，使伺服器沒有資源處理合法的交互。後面的問題可以可通過限制回應的速率以減緩，見 4.7.攻擊者可以偽裝合法用戶端的位址進行欺騙這可能引起服務端阻塞針對用戶端的合法的回應，因為 NSTART=1。攻擊是針對非安全模式的,這些攻擊在安全模式被避免，而不是 NoSec。

11.5. 跨協定攻擊

CoAP 端點向虛假的源位址發送包不僅能用於增幅，也能用於對一個監聽給定位址(IP 位址和埠)的受害者進行跨協定攻擊。按照下面步驟就會發生：

- 攻擊者往 CoAP 端點發送一個帶有將虛假源位址當做給定位址的消息。
- CoAP 端點向給定的源位址發送回應訊息。
- 在給定位址接收UDP包的受害者就會按照不同協定的規則進行解析。

它可以用於繞過阻止從攻擊者向受害者通信的防火牆規則，但是將允許從 CoAP 端點(在其他協定中也擔當有效角色)到受害者通信。

另外，CoAP 端點可能成為由其他基於 UDP 協定(譬如 DNS)的端點發起的跨協定攻擊的受害者。在這些情況下，如果端點的安全屬性依賴於檢查 IP 位址(和使用虛假 IP 位址來切斷外界攻擊的防火牆)，就有可能遭受攻擊。通常，由於基於 UDP 的協定缺乏上下文，因此它們更容易成為跨協定攻擊的目標。

最後，由其他方式傳輸的 CoAP URI 能夠用於使得用戶端往其他協定的端點發送消息。

一個減輕跨協定攻擊的措施是嚴格檢查接收包的語法和語法中的足夠多的差異。舉個例子，如果很難使得 DNS 伺服器向 CoAP 端點發送一個傳遞檢查的 DNS 回應，那麼就會起作用。可惜的是，DNS 回復的前兩個位元組是能被攻擊者選中的 ID，並且映

射為 CoAP 頭部的關鍵部分，後面兩個位元組被當做 CoAP 的消息 ID(任何值都可行)。DNS 的計數字可以當做一個(不存在但是可選的)CoAP 選項 0 的多重實例，或者當做一個 Token。攻擊者利用重複的查詢來在 CoAP 端點上達到一個預期效果；服務端增加的回應(如果有的話)將被當做額外的負載。

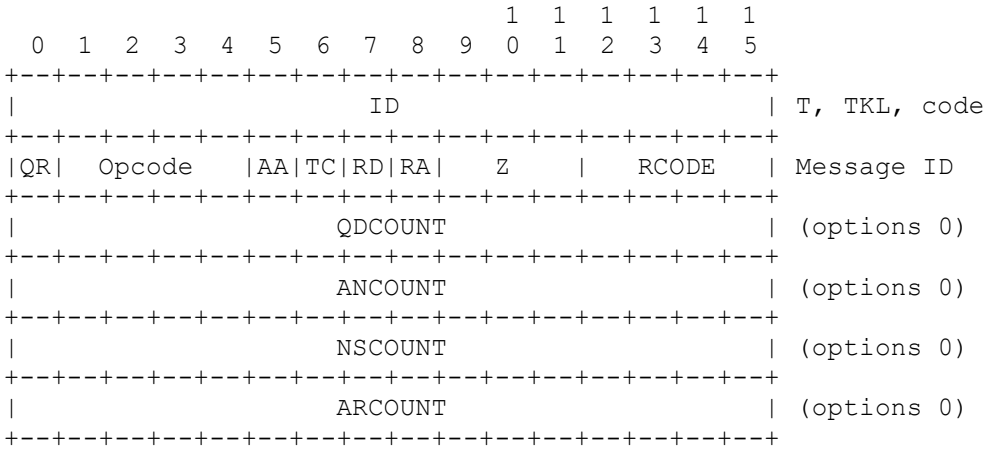


圖 15: DNS 標頭 ([RFC1035], Section 4.1.1) vs. CoAP訊息

一般來說，對於任何一對協定，協定之一很容易設計為使得攻擊者生成類似另一種協定的消息的回復。比起保證或者證明不存在可實行的攻擊，生成可能未完全啟動的但可能被二次開發的攻擊的實例通常更加困難。如果端點依據受信任包的源 IP 位址而沒有授權攻擊者，那麼跨協定攻擊才能完全的減輕。相反的，完全依賴防火牆的 NoSec 環境下的 CoAP 安全不僅需要防火牆來切斷 CoAP 的端點，並且使得所有其它的端點使用一些基於 UDP 的協定來往 CoAP 端點發送 UDP 消息。

除了上面的考量，也要考慮跨協定攻擊的 DTLS 安全。舉例，如果相同的 DTLS 安全連接("connection")用於傳輸多種協定的資料，那麼 DTLS 就不對這些協定提供應對跨協定攻擊的保護。

11.6. 受限節點的注意事項

受限節點上的實現經常發現它們沒有好的資訊熵來源[RFC4086]。在這種情況下，節點必須不能用於對資訊熵要求高的處理過程，

比如說 key 的生成。另外，在製造或者運行中可以生成 keys 並添加到設備中。

由於受限節點的低處理能力，它們最容易受到時序攻擊的影響。在實現密碼原語時需多加注意。

在暴露環境中安裝大量的受限節點會使得它們對竄改沒有抵抗能力，包括密匙內容的恢復。當定義分配給它們的認證資訊時需要考慮到這一點。尤其是，當向一組節點分配一個共用的 key 可能使得任何一個單一的受限節點成為破換整個組的目標。

12. 網際網路地址分配注意事項

12.1. CoAP 代碼註冊

這篇文件定義了兩個 sub-registries，給 CoAP 頭部代碼欄位的值包含“Constrained RESTful Environments (CoRE) Parameters”註冊表。將來參考“CoRE 參數”註冊表。

這兩個 sub-registries 都是 8 位值，可以用三個十進位的符號表示為“c.dd”,用一個句號將第一位和第二位元數字分離開；第一位數字 c 為 07，表示代碼種類；第二個和第三個數字 dd 為 0031 的十進位數字，表示細節。

所有的代碼值都按照 sub-registries，按照如下範圍安排：

0.0 表示一個空的消息（見 4.1 章)

0.01-0.31 表示一個請求。這個範圍的值是根據“CoAP Method Codes”的 sub-registry 分配的（見 12.1.1 節）

1.00-1.31 保留

.

2.00-5.31 表示一個回應。這個範圍的值是根據“CoAP Response Codes”的 sub-registry 分配的（見 12.1.2 節）

6.00-7.31 保留

12.1.1. 方法碼

這個 sub-registry 是“CoAP Method Codes”

每次進入這個 sub-registry 都必須包含在 0.01-0.31 範圍內的 Method Code，方法的名字，方法文件的參考。

初始化進入這個 sub-registry 如下：

Code	Name	Reference
0.01	GET	[RFC7252]
0.02	POST	[RFC7252]
0.03	PUT	[RFC7252]
0.04	DELETE	[RFC7252]

表 5: CoAP Method 代碼

其他 Method Codes 沒有安排。

網際網路號碼分配政策在未來為這個 sub-registry 額外定義描述在“IETF Review or IESG Approval” [RFC5226].

方法代碼的文件需要指定這個請求的語義，包含如下屬性：

- 這個方法的回應碼成功返回。
- 這個方法是否冪等，安全或兩者都滿足。

12.1.2. 回應碼

這個 sub-registry 的名字是“CoAP Response Codes”

每個 sub-registry 必須包含在 2.00-5.31 範圍內的狀態碼，狀態碼的描述，回應碼的文件參考。

初始化進入這個 sub-registry 如下：

Code	Description	Reference
2.01	Created	[RFC7252]
2.02	Deleted	[RFC7252]
2.03	Valid	[RFC7252]
2.04	Changed	[RFC7252]
2.05	Content	[RFC7252]
4.00	Bad Request	[RFC7252]
4.01	Unauthorized	[RFC7252]
4.02	Bad Option	[RFC7252]
4.03	Forbidden	[RFC7252]
4.04	Not Found	[RFC7252]
4.05	Method Not Allowed	[RFC7252]
4.06	Not Acceptable	[RFC7252]
4.12	Precondition Failed	[RFC7252]
4.13	Request Entity Too Large	[RFC7252]
4.15	Unsupported Content-Format	[RFC7252]
5.00	Internal Server Error	[RFC7252]
5.01	Not Implemented	[RFC7252]
5.02	Bad Gateway	[RFC7252]
5.03	Service Unavailable	[RFC7252]
5.04	Gateway Timeout	[RFC7252]
5.05	Proxying Not Supported	[RFC7252]

表 6: CoAP 回應代碼

回應碼 3.00-3.31 是預留給將來使用。所有其他回應碼都沒有被安排。

網際網路號碼分配政策為這個 sub-registry 以後額外的定義描述在“IETF Review or IESG Approval”[RFC5226].

回應碼的文件需要指定這個回應的語義，包含如下屬性：

- 回應碼應用方式
- 是否需要攜帶有效負載，option。
- 有效負載的語義。舉個例子，2.05（內容）回應的有效負載是目標資源的展示；有效負載在錯誤的回應中是可讀和診斷的。

- 有效負載的格式。舉個例子，這個格式在 2.05（內容）回應是通過內容格式選項表示；有效負載的格式在一個錯誤的回應中總是 Net-Unicode 文本

- 回應是否可以緩衝，取決於 freshness model

- 回應是否通過合法性檢查，取決於 validation model

- 回應是否導致一個 cache 來標誌回應已經存儲，表明這個請求的 URI 不是最新的。

12.2. CoAP 選項號碼登錄

這篇文件為 CoAP options 中“CoRE Parameters”註冊表中的 option 編號定義了一個 sub-registry。這個 sub-registry 的名字是“CoAP Option Number”。

每個 sub-registry 必須包含這個 option 編號，option 的名稱，還有 option 文件參考。

初始化進入這個 sub-registry 如下：

Number	Name	Reference
0	(Reserved)	[RFC7252]
1	If-Match	[RFC7252]
3	Uri-Host	[RFC7252]
4	ETag	[RFC7252]
5	If-None-Match	[RFC7252]
7	Uri-Port	[RFC7252]
8	Location-Path	[RFC7252]
11	Uri-Path	[RFC7252]
12	Content-Format	[RFC7252]
14	Max-Age	[RFC7252]
15	Uri-Query	[RFC7252]
17	Accept	[RFC7252]
20	Location-Query	[RFC7252]
35	Proxy-Uri	[RFC7252]
39	Proxy-Scheme	[RFC7252]
60	Size1	[RFC7252]
128	(Reserved)	[RFC7252]
132	(Reserved)	[RFC7252]
136	(Reserved)	[RFC7252]
140	(Reserved)	[RFC7252]

表 7: CoAP 選項編號

網際網路號碼分配政策為這個 sub-registry 以後額外的定義分為了如下 3 層。0..255 是為 option 保留，在 IETF 有被定義（IETF Review or IESG Approval）。256..2047 是為普通使用的包含公開規格（Specification Required）的 options 保留的。2048..64999 是為了所有其他 options，包含私人的或者贊助商規格的，這些需要經過一個特定的專家審核來確定這個 option 語法是定義正確的。在 6500 和 65535 之間的 Option 編號是保留下來用於實驗。他們不是給贊助商使用，而且是禁止用於操作部署。

Range	Registration Procedures
0-255	IETF Review or IESG Approval
256-2047	Specification Required
2048-64999	Expert Review
65000-65535	Experimental use (no operational use)

表 8: CoAP 選項編號:註冊程序

這個 option 編號的文件應該指定這個 option 和它的編號，包含以下屬性：

- option 在請求中的意義。
- option 在回應中的意義。
- 這個 option 是 critical 還是 elective，由 option 編號決定。
- 這個 option 是否是 safe-to-forward，如果是，是否是 cache-key 的一部分，這些由 option 編號決定（見 5.4.2 節）
- option 值的格式和長度。
- 是否 option 只一次出現，還是它能出現很多次。
- 預設值。對於一個有預設值的 critical option，存在這樣的討論，預設值如何能處理通過實施，而這個實施又不支持 critical option（見 5.4.4 節）

12.3. CoAP 內容格式登錄

網際網路媒體類型被定義成一個字串，例如 “application/xml”[RFC2046]。為了最小化使用這些媒體類型表示格式所帶來的有效負載開銷，這篇文件為網際網路媒體類型子集定義了一個 sub-registry，這些子集在 CoAP 中使用，並且每個都分配好，和 content-coding 合併，有數位識別碼。這個 sub-registry 的名字是“CoAP Content-Formats”，在“CoRE Parameters”表裡面

每次進入到 sub-registry 必須包含這些 IANA 註冊過的媒體類型，用於定義 CoAP 中的媒體類型數字的範圍是 0-65535，cotent-coding 和這些定義相關，一個文件參考描述了有效負載和媒體類型的表達語義。

CoAP 不包括用分離的方式來轉移 content-encoding 資訊和請求或回應，因為 content-encoding 每一個識別字也是特定的。如果多種 content-encodings 和媒體類型一起使用，然後每一個分離的 content-format 識別字將會被註冊。相似的，其他參數和網際網路

媒體類型關聯，比如 level，也能被定義為 CoAP Content-Format 條目。

初始化進入這個 sub-registry 如下：

Media type	Encoding	ID	Reference
text/plain; charset=utf-8	-	0	[RFC2046] [RFC3676] [RFC5147]
application/link-format	-	40	[RFC6690]
application/xml	-	41	[RFC3023]
application/octet-stream	-	42	[RFC2045] [RFC2046]
application/exi	-	47	[REC-exi-20140211]
application/json	-	50	[RFC7159]

表 9: CoAP 內容格式

65000 到 65535 之間所包含的識別字是預留用於實驗。他們不是給贊助商使用，也禁止用於操作部署。2569999 之間的識別字預留是用於未來 IETF 規格（IETF Review or IESG Approval）。所有其他識別字都尚未安排。

由於一個位元組識別字的命名空間非常小，IANA 政策對於 sub-registry 未來額外的定義在 0~255 範圍，是“Expert Review”描述在[RFC5226]。IANA 政策對於額外的定義在 1000-64999 包含了“First Come First Served”，描述在[RFC5226]。總結在下表中。

Range	Registration Procedures
0-255	Expert Review
256-9999	IETF Review or IESG Approval
10000-64999	First Come First Served
65000-65535	Experimental use (no operational use)

表 10: CoAP 內容格式: 登記程序

在 M2M 的應用中，不希望這些常用的媒體類型，比如 text/plain,application/xml 或者 application/octet-stream 在實際應用中長期有效。推薦 M2M 的應用中使用 CoAP 請求新的網際網路媒體類型，這些媒體類型來自於 IANA 表明的語義，該語義資訊規定了如何創建和理解一個有效負載。舉個例子，一個智慧的有

效負載攜帶的一個 XML，可能請求一個更特定的類型，比如 application/se+xml 或者 application/se-exi。

12.4. URI 方案登記

URI 名
coap

Status.
Permanent.

URI 規範語法.
在[RFC7252]的章節 6.1 中定義.

URI 規範的語義
CoAP URI 規範提供了一種能在 CoAP 協定上定位和訪問資源的的方法。通過 CoAP 伺服器可以定義和操作資源。該規範是參考 Http 的 URI 見 RFC2616

編碼注意項
該規範的編碼規則遵循已近制定的[RFC3986]，例如其網際網路和資源章節表明 UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
CoAP 端點利用該規範訪問 CoAP 資源

交交互操作注意項
None.

安全注意項
見[RFC7252]的章節 11.1。

聯繫方式
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References. [RFC7252]

12.5. 安全 URI 規範註冊表

本文包含註冊統一資源識別項(URI)方案“coaps”的請求。註冊請求符合[RFC4395]。

URI 正式名稱
coaps

Status.
Permanent.

URI scheme syntax.
Defined in Section 6.2 of [RFC7252].

URI 規範語義
CoAP URI 規範提供了一種能在 CoAP 協定上標識資源的的方法，該方法把 DTLS 作為其安全傳輸層。通過 CoAP 伺服器可以定義和操作資源。該規範是參考 Http 的 URI [RFC2616] 和 [RFC7252]。

編碼注意項
該規範的編碼規則遵循已近制定的[RFC3986]，例如其網際網路和資源章節表明 UTF-8-based percent-encoding。

Applications/protocols that use this URI scheme name.
CoAP 端點利用該規範通過 DTLS 訪問 CoAP 資源

交互操作注意項
None.

安全注意項
見[RFC7252]的章節 11.1。

聯繫方式
IETF Chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFC7252]

12.6. 安全服務名稱和埠號表

CoAP 的功能之一是資源發現:CoAP 客戶機可以向 CoAP 伺服器詢問它提供的資源(參見第 7 節)。

IANA已經根據[RFC6335]分配了埠號5683和服務名稱“coap”。

除了單播之外，CoAP 還可以用於多播和任意播。

服務名稱
coap

傳輸協定.
udp

代理
IESG <iesg@ietf.org>

聯繫方式.
IETF Chair <chair@ietf.org>

描述
Constrained Application Protocol (CoAP)

參考
[RFC7252]

埠號
5683

12.7. 保全服務名稱與埠號登記

使用 dtls 保護的 CoAP"coaps"方案也可以提供 CoAP 資源發現。因此，需要對用於安全資源發現的 CoAP 埠進行標準化。

IANA 根據[RFC6335]分配了埠號 5684 和服務名稱"coaps"。

除了單播之外，受 dtls 保護的 CoAP 還可以與 anycast 一起使用。

Service Name.
coaps

Transport Protocol.
udp

Assignee.
IESG <iesg@ietf.org>

Contact.
IETF Chair <chair@ietf.org>

Description.
DTLS-secured CoAP

Reference.
[RFC7252]

Port Number.
5684

12.8. 多播地址表

第 8 章，“多播 CoAP”，定義了多播的使用。IANA 已經安排了如下被用於 CoAP 節點多播地址：

IPv4-- "All CoAP Nodes"地址為 224.0.1.187，源自“IPv4 Multicast Address Space Registry”。當這個地址用於發現

IPv6-- "All CoAP Nodes" address FF0X::FD, from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a

distinct multicast address for each scope that interested CoAP nodes should listen to； CoAP needs the Link-Local and Site-Local scopes only.

13. 致謝

Brian Frank was a contributor to and coauthor of early versions of this specification.

Special thanks to Peter Bigot, Esko Dijk, and Cullen Jenningsfor substantial contributions to the ideas and text in the document, along with countless detailed reviews and discussions.

Thanks to Floris Van den Abeele, Anthony Baire, Ed Beroset, Berta Carballido, Angelo P. Castellani, Gilbert Clark, Robert Cragie, Pierre David, Esko Dijk, Lisa Dusseault, Mehmet Ersue, Thomas Fossati, Tobias Gondrom, Bert Greevenbosch, Tom Herbst, Jeroen Hoebeke, Richard Kelsey, Sye Loong Keoh, Ari Keranen, Matthias Kovatsch, Avi Lior, Stephan Lohse, Salvatore Loreto, Kerry Lynn, Andrew McGregor, Alexey Melnikov, Guido Moritz, Petri Mutka, Colin O'Flynn, Charles Palmer, Adriano Pezzuto, Thomas Poetsch, Robert Quattlebaum, Akbar Rahman, Eric Rescorla, Dan Romascanu, David Ryan, Peter Saint-Andre, Szymon Sasin, Michael Scharf, Dale Seed, Robby Simpson, Peter van der Stok, Michael Stuber, Linyi Tian, Gilman Tolle, Matthieu Vial, Maciej Wasilak, Fan Xianyou, and Alper Yegin for helpful comments and discussions that have shaped the document. Special thanks also to the responsible IETF area director at the time of completion, Barry Leiba, and the IESG reviewers, Adrian Farrel, Martin Stiemerling, Pete Resnick, Richard Barnes, Sean Turner, Spencer Dawkins, Stephen Farrell, and Ted Lemon, who contributed in- depth reviews.

Some of the text has been borrowed from the working documents of the IETF HTTPBIS working group.

14. 引用文件

14.1. 規範性引用文件

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC768, August 1980.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose InternetMail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose InternetMail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs toIndicate Requirement Levels", BCP 14, RFC 2119, March1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee,"Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June1999.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XMLMedia Types", RFC 3023, January 2001.

[RFC3629] Yergeau, F., "UTF-8, a transformation format ofISO 10646", STD 63, RFC 3629, November 2003.

[RFC3676] Gellens, R., "The Text/Plain Format and DelSp Parameters", RFC 3676, February 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter,"Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 35,RFC 4395, February 2006.

[RFC5147] Wilde, E. and M. Duerst, "URI Fragment Identifiers forthe text/plain Media Type", RFC 5147, April 2008.

[RFC5198] Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writingan IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport LayerSecurity (TLS) Protocol Version 1.2", RFC 5246, August2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate RevocationList (CRL) Profile", RFC 5280, May 2008.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T.Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation forIPv6 Address Text Representation", RFC 5952, August2010.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010. [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
Extension Definitions", RFC 6066, January 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, August 2012.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Thingswith Hashes", RFC 6920, April 2013.

[RFC7250] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S.,and T. Kivinen, "Using Raw Public Keys in TransportLayer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, June 2014.

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES- CCM Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 7251, June2014.

14.2. 參考目錄

[BLOCK] Bormann, C. and Z. Shelby, "Blockwise transfers in CoAP", Work in Progress, October 2013.

[CoAP-MISC] Bormann, C. and K. Hartke, "Miscellaneous additionsto CoAP", Work in Progress, December 2013.

[EUI64] IEEE Standards Association, "Guidelines for 64-bit Global Identifier (EUI-64 (TM))", Registration Authority Tutorials, April 2010, <http://standards.ieee.org/regauth/oui/tutorials/EUI64.html>.

[GROUPCOMM] Rahman, A. and E. Dijk, "Group Communication for CoAP", Work in Progress, December 2013.

[HHGTTG] Adams, D., "The Hitchhiker’s Guide to the Galaxy",Pan Books ISBN 3320258648, 1979.

[IEEE1003.1] IEEE and The Open Group, "Portable Operating System Interface (POSIX)", The Open Group Base Specifications Issue 7, IEEE 1003.1, 2013 Edition,

<http://pubs.opengroup.org/onlinepubs/9699919799/>.

[IPsec-CoAP]
Bormann, C., "Using CoAP with IPsec", Work in Progress, December 2012.

[MAPPING] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for HTTP-CoAP Mapping Implementations", Work in Progress, February 2014.

[OBSERVE] Hartke, K., "Observing Resources in CoAP", Workin Progress, April 2014.

[REC-exi-20140211]
Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", W3C Recommendation REC-exi-20140211, February 2014, <http://www.w3.org/TR/2014/REC-exi-20140211/>.

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

[RFC0020] Cerf, V., "ASCII format for network interchange", RFC 20, October 1969.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1035] Mockapetris, P., "Domain names -

implementationand specification", STD 13, RFC 1035, November1987.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264,June 2002.

[RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo,"Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API)for IPv6", RFC 3542, May 2003.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E.,and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March2006.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May2006.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE

802.15.4 Networks", RFC 4944, September 2007.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5489] Badra, M. and I. Hajjeh, "ECDHE_PSK Cipher Suites for Transport Layer Security (TLS)", RFC 5489, March2009.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February2011.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC6282, September 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., andS. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, July2012.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, April 2013.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol -OCSP",

RFC 6960, June 2013.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, June 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminologyfor Constrained-Node Networks", RFC 7228, May 2014.

[RTO-CONSIDER]
Allman, M., "Retransmission Timeout Considerations", Work in Progress, May 2012.

[W3CXMLSEC]
Wenning, R., "Report of the XML Security PAG", W3CXML Security PAG, October 2012, <http://www.w3.org/2011/xmlsec-pag/pagreport.html>.

附錄 A. 範例

本節給出了一些關於 GET 請求的消息流的簡短操作。這些操作演示了基本操作、存在重發和多播的操作。

圖 16 顯示了一個導致負載回應的基本 GET 請求:用戶端使用消息 ID 0x7d34 向伺服器發送資源 coap://server/temperature 的可驗證 GET 請求。該請求包含一個 uri 路徑選項(Delta 0 + 11 = 11，長度 11，值“temperature”)；權杖是空的。這個請求總共有 16 個位元組長。在確認消息中返回一個 2.05 (Content)回應，確認可確認的請求，回應訊息 ID 0x7d34 和空權杖值。回應包含一個“22.3 C”的有效負載，長度為 11 位元組。

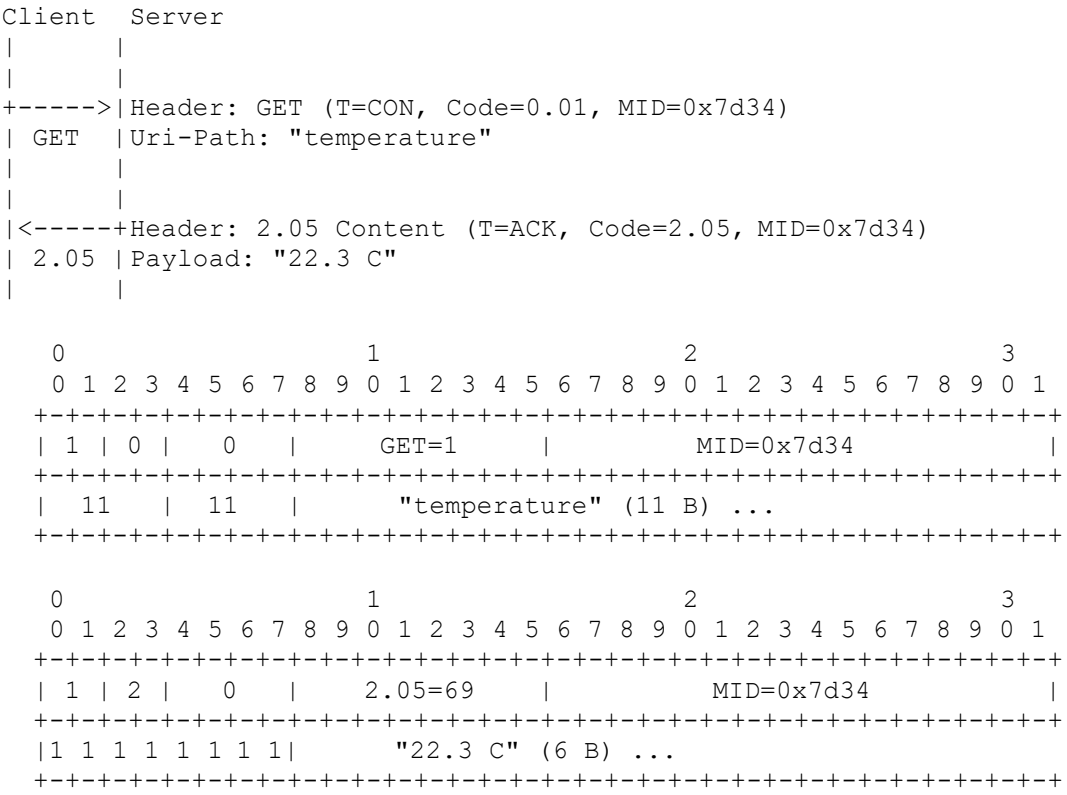
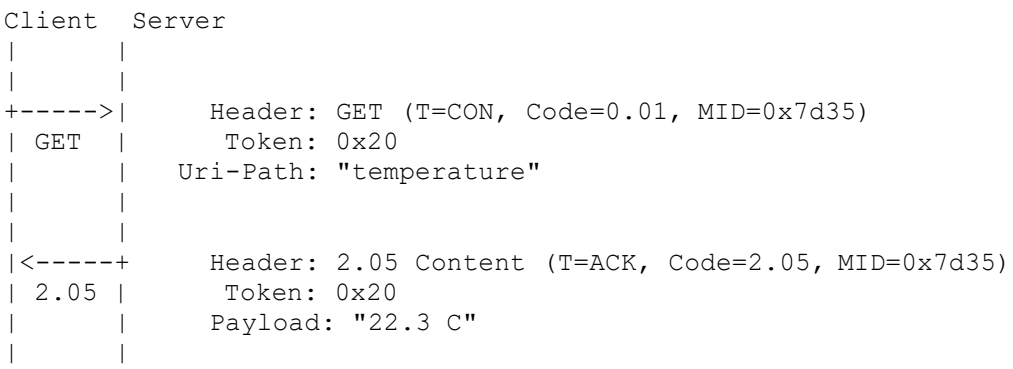


圖 16: 可證實的請求；附帶回應

圖 17 顯示了一個類似的範例，但是在請求和回應中包含了一個非空白標記(值 0x20)，分別將大小增加到 17 和 12 位元組。



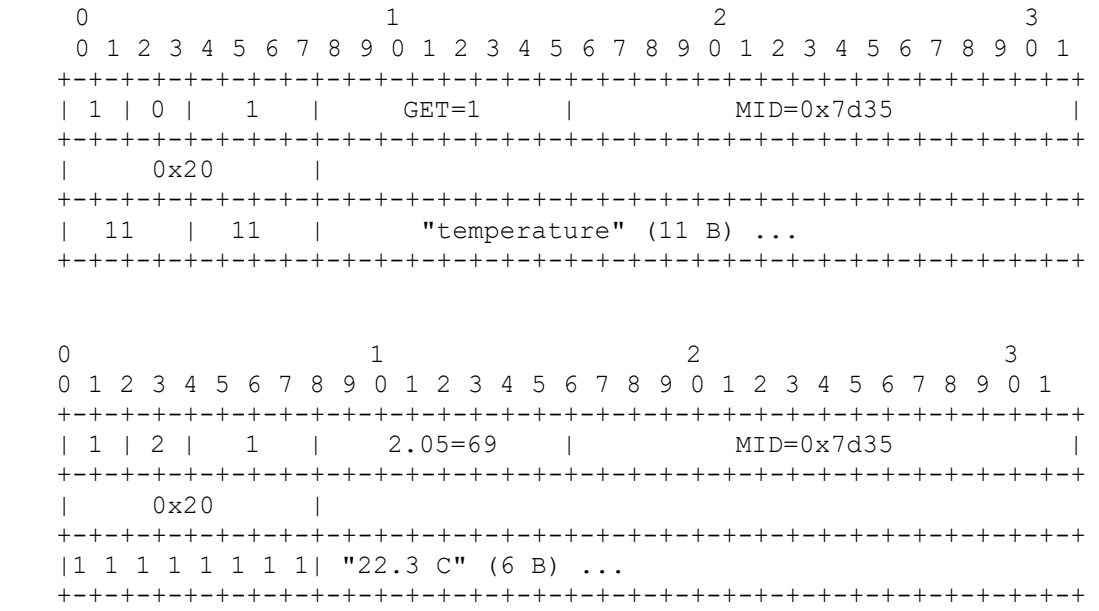


圖 17: 可證實的請求；附帶回應

在圖 18 中，可確認的 GET 請求丟失。ACK_TIMEOUT 秒後，用戶端重新傳輸請求，產生與前一個示例相同的負載回應。

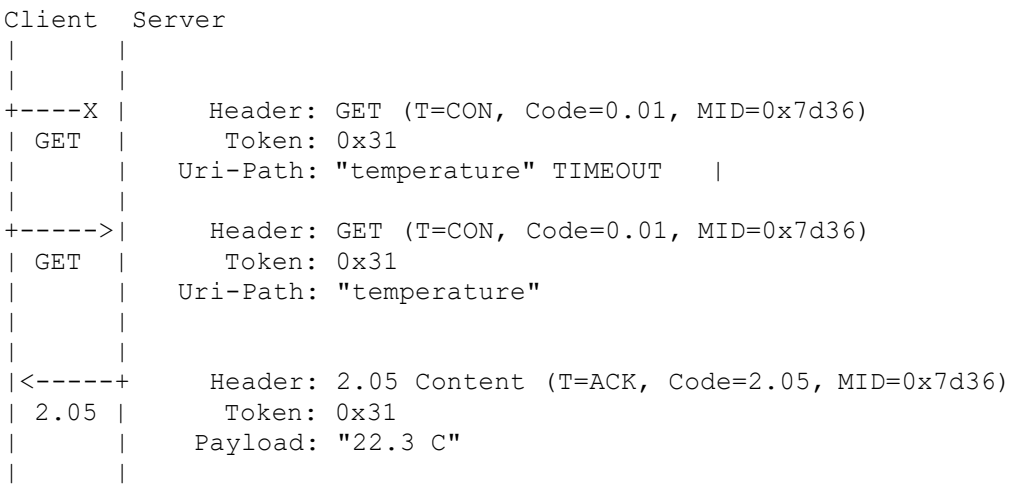


圖 18: 可證實的請求(傳送)； 附帶回應

在圖 19 中，從伺服器到客戶機的第一個確認消息丟失。
ACK_TIMEOUT 秒後，客戶機重新傳輸請求。

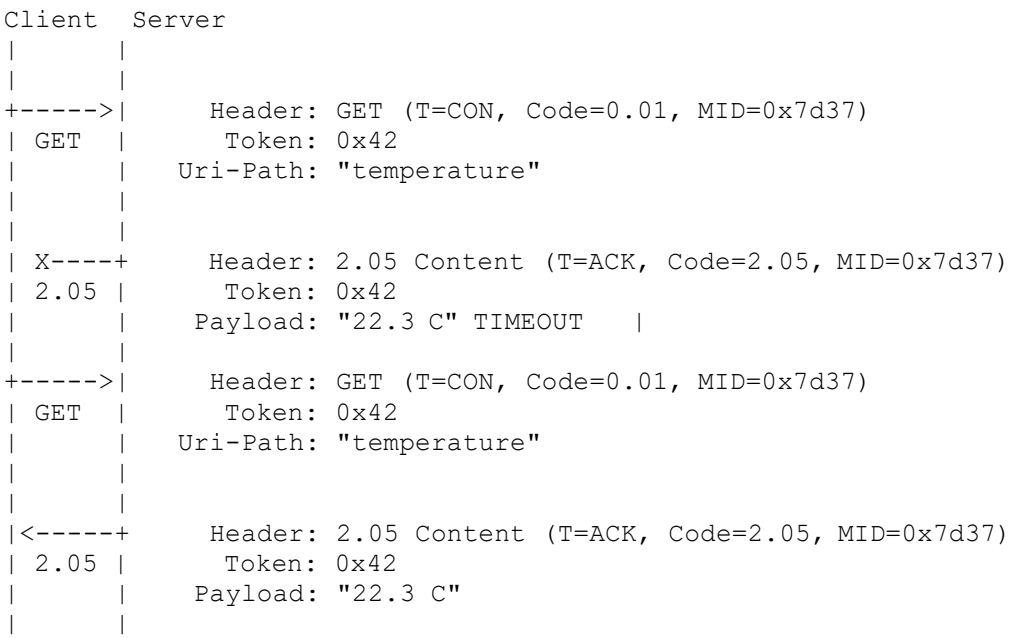


圖 19: 可證實的請求； 偷偷的回應(傳送)

在圖 20 中，伺服器確認確認請求，並在確認消息中單獨發送一個 2.05 (Content)回應。請注意，確認消息和可確認的回應不一定按照發送它們的順序到達。用戶端確認可確認的回應。

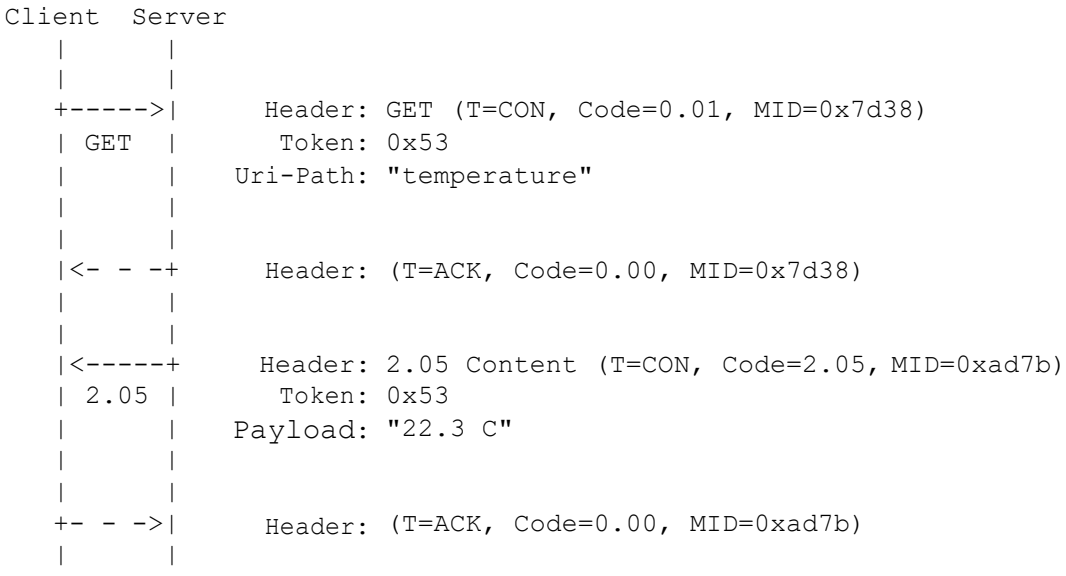


圖 20: 可證實的請求；單獨回應(傳送)

圖 21 顯示了一個示例，其中客戶機在發送一個可確認的請求後立即丟失其狀態(例如，崩潰並重新啟動)，因此稍後到達的單獨回應是意料之外的。在這種情況下，用戶端拒絕帶有重置消息的可確認回應。注意，意外的 ACK 將被忽略。

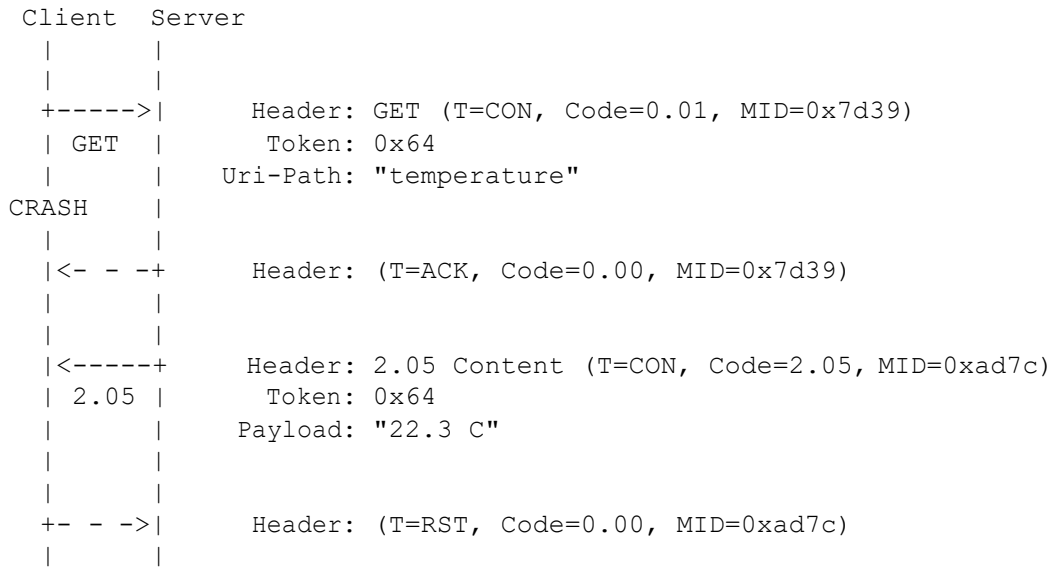


圖 21: 可證實的請求；單獨回應(意外)

圖 22 顯示了一個基本的 GET 請求，其中請求和回應是不可驗證的，因此可能會在不通知的情況下丟失。

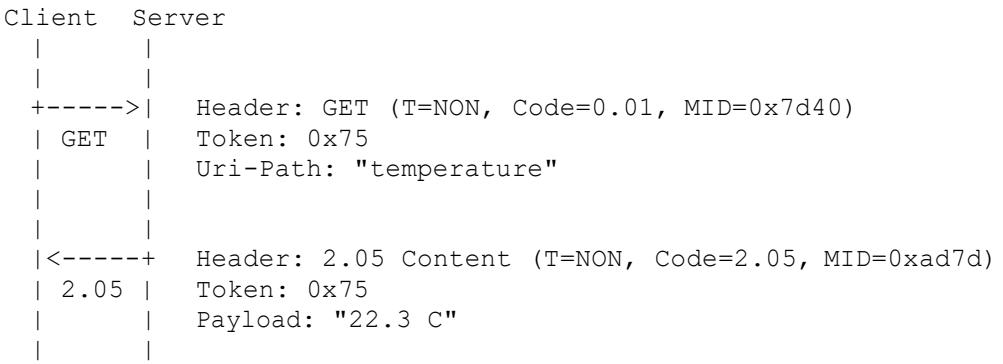


圖 22: 不可證實的請求；NON 回應(意外)

在圖23中，用戶端向多播位址發送一個不可確認的GET請求:
連結-本地範圍內的所有節點。該連結上有3個伺服器:A、B和c。
伺服器A和B有一個匹配的資源，因此它們返回一個不可驗證
的2.05(內容)回應。B發送的回應丟失了。C沒有匹配的回應，
因此它發送一個不可驗證的4.04(未找到)回應。

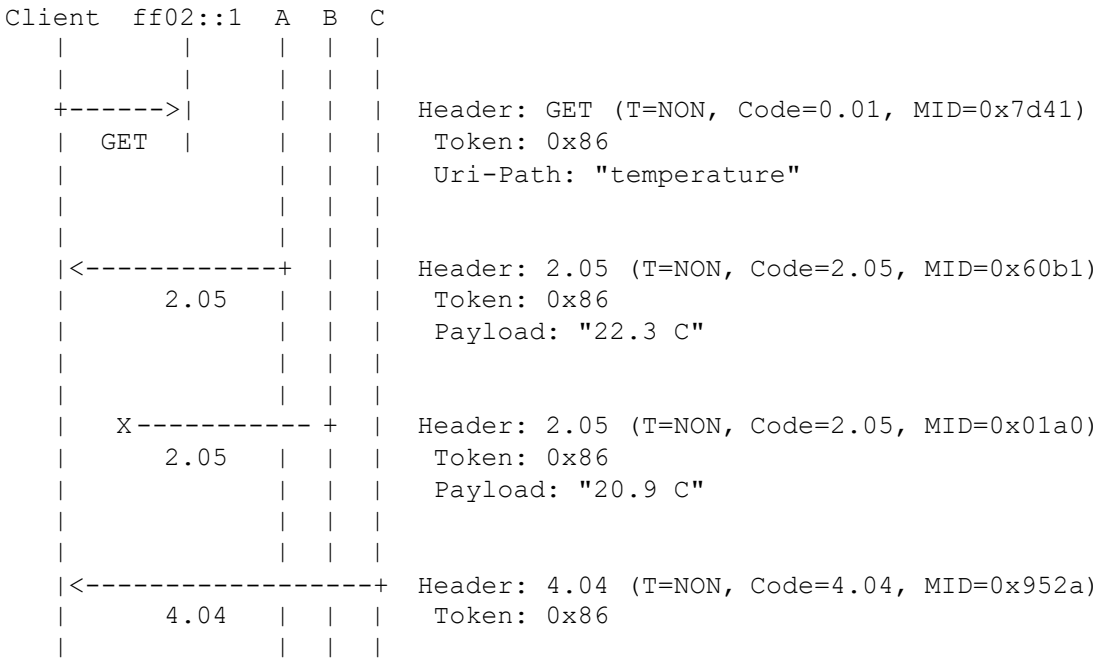


圖 23: 不可證實的請求；NON 回應(多播)；不可證實的回應

附錄 B. URI 範例

下面的範例演示了不同的 Uri 選項集，以及從中構造 Uri 之後的結果。除了選項之外，第 6.5 節還提到了目標 IP 位址和埠，但是並不是演算法的所有路徑都導致目標 IP 位址和埠包含在 URI 中。

- Input:

  Destination IP Address = [2001:db8::2:1] Destination UDP Port = 5683

Output:

coap://[2001:db8::

2:1]/ o Input:

Destination IP Address =
[2001:db8::2:1] Destination UDP Port
= 5683
Uri-Host =

"example.net" Output:

coap://example.net/

o Input:

Destination IP Address =
[2001:db8::2:1] Destination UDP Port
= 5683
Uri-Host =
"example.net"
Uri-Path =
".well-known"
Uri-Path = "core"

Output:

coap://example.net/.well-known/

core o Input:

Destination IP Address =
[2001:db8::2:1] Destination UDP Port
= 5683
Uri-Host = "xn--18j4d.example"
Uri-Path = the string composed of the Unicode characters
U+3053 U+3093 U+306b U+3061 U+306f, usually represented in
UTF-8 as E38193E38293E381ABE381A1E381AF hexadecimal

Output:

coap://xn--18j4d.example/
%E3%81%93%E3%82%93%E3%81%AB%E3%81%A1%E3%81%AF

(The line break has been inserted for readability; it is
not part of the URI.)

- Input:

```
Destination IP Address =
198.51.100.1 Destination UDP Port
= 61616
Uri-Path =
"" Uri-Path
= "/"
Uri-Path =
"" Uri-Path
= ""
Uri-Query =
"//"
Uri-Query =
"?&"
```

Output:

```
coap://198.51.100.1:61616//%2F//?%2F%2F&?%26
```

作者資訊

地址
Zach Shelby
ARM
150 Rose Orchard San Jose, CA 95134
USA

電話號碼: +1-408-203-9434
電子郵件: zach.shelby@arm.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

電話號碼: +49-421-218-63905
電子郵件: hartke@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359

Germany

電話號碼: +49-421-218-63921
電子郵件: cabo@tzi.org