EMOJI AS TRANSPORT PROTOCOL
A SEMI-SERIOUS STUDY OF HUMAN COMMUNICATION

DORIAN VERLAINE

Abstract. 本文提出一個半嚴肅的模型：現代即時通訊中的 Emoji 並不只是情緒圖示，而可以被視為人類社交協議棧中的一組輕量級控制訊號。透過 TCP、UDP、QUIC、WebSocket、Server-Sent Events、WebRTC 與 MQTT 等網路模型，我們分析 👀、👍、🤡、💀、🫠 等常見 Emoji 如何承擔連線建立、確認、狀態同步、語氣標記、錯誤處理與會話恢復等功能。本文進一步提出一個虛構的 Emoji Pragmatic Layer，並討論 Emoji 在高語義密度、低傳輸成本、上下文依賴與協議歧義方面的特性。研究發現，👀 在大量場景下的行為異常接近 SYN、PING、SUBSCRIBE 或 SESSION_RESUME，而 💀 則可以被合理建模為 500 Internal Human Error。本文不保證符合任何 IETF、IEEE、ACM 或正常人類行為規範。

1. Introduction

人類發明自然語言之後，很快又花了幾千年時間想辦法少打幾個字。從書信縮寫、短信、Internet slang，到今天的 Emoji，通訊的歷史某種程度上也是一部持續壓縮 Application Payload 的歷史。考慮一個非常普通的例子：某人本來可以輸入「我看到你剛才說的內容了，而且我對這件事產生了一定程度的興趣，請繼續」，但在即時通訊軟體裡，他往往只會發：

👀

奇怪的是，這條訊息幾乎沒有任何明確的命題內容，卻仍然可以被另一個人類 endpoint 正確解碼。接收者通常知道這並不代表「我有兩隻眼睛」，而更接近「我在看」、「繼續」、「有情況？」、「你最好把後面的說完」，甚至「我已經察覺到你準備說一些不太正常的東西」。

這意味著 Emoji 傳輸的內容，很多時候不是傳統意義上的 semantic payload，而是 **pragmatic metadata**：語氣、關係狀態、回應期待、情緒權重、是否帶有諷刺、是否需要建立會話，以及發送者目前的 process 是否仍然穩定。

因此，本文提出一個核心假設：

> **Emoji** 可以被視為存在於自然語言之上、社交應用層之下的一種輕量級控制協議。

如果這個假設成立，那麼 👀 就可能是一個 connection probe，👍 是 acknowledgement，🤡 是 irony flag，🫠 是 local resource exhaustion，而 💀 則意味著 remote application process 已因 payload 過於荒謬而暫時崩潰。

這個模型當然不是對人類語言的嚴格語言學描述。但它非常適合解釋一個計算機科學學生每天在聊天軟體裡目睹的大量不標準網路行為。

2. A Proposed Human Communication Stack

為了討論方便，可以把現代文字聊天粗略表示為以下協議棧：

Date: 2026-09-10.
2020 *Mathematics Subject Classification.* Human Networking & Pragmatics (Extremely Experimental).
Key words and phrases. emoji, network-protocols, pragmatics, internet-culture, satire.

Layer	Payload or role
Application Layer	「今晚吃什麼？」
Emoji / Pragmatic Layer	👀 🤡 😈 💀 🫠 🙂
Conversational Session Layer	Friendship / Context / Shared State
Transport Model	TCP / UDP / QUIC / WebSocket
Physical Interface	Fingers / Screen / Voice / Face

在這個模型中，普通文字主要負責「說了什麼」，而 Emoji 更偏向描述「這句話應該怎麼解碼」。例如以下兩條訊息擁有完全相同的文字 payload：

> 你今晚有空嗎。
> 你今晚有空嗎 👀

但任何具有基本社交協議相容性的人都知道，它們不完全是一回事。第一句可以是一個中性的 availability query；第二句則可能帶有期待、暗示、試探、好奇，甚至是一個尚未公開完整 payload 的 connection initiation。換句話說，Emoji 並不是單純增加資訊，而是在修改整個 packet 的 application semantics。

這非常接近網路協議中的 flags、headers 與 control frames：資料本身可能沒變，但解碼方式改變了。

3. TCP: Reliable Human Interaction

TCP 最重要的特徵是 connection-oriented、reliable delivery、ordered delivery，以及對 acknowledgement 的執著。這使它非常適合描述一類人類社交行為：不只要把訊息發出去，還非常關心對方究竟有沒有收到、看懂、回覆，以及為什麼已讀了三分鐘還沒有 ACK。

3.1 👀 as SYN

考慮最小化社交封包：

> 👀

它沒有明確 topic，沒有完整問題，也沒有真正 payload，卻經常能觸發會話建立。因此可以近似定義：

> 👀 = SYN

典型握手可能是：

```
Dorian -> Peer: 👀
Peer -> Dorian: 👀
Dorian -> Peer: 😈
```

如果硬要套 TCP 狀態機，可以得到：

```
👀 = SYN
👀 = SYN-ACK
😈 = ACK + suspicious application data
```

至此雙方進入 `ESTABLISHED` 狀態，而實際 topic 甚至還沒有出現。這是一種非常高效、同時也非常危險的會話建立方式。尤其是最後的 😈，通常意味著 connection 雖然成功，但後續 payload 的安全性已經無法保證。

3.2 👍 as ACK

👍 是另一個幾乎完美的 transport-layer Emoji。它最核心的意思往往不是「我很喜歡」，而只是：

```
Packet received.
```

問題在於 transport success 不代表 application success。假設：

> A: 我花了兩個小時整理這個方案，你覺得怎麼樣？
> B: 👍

從傳輸層來看，這是一個成功案例。訊息抵達，對方確認收到，connection healthy。但從社交 application layer 看，結果可能是：

```
ERROR: meaningful_response_not_found
```

這是一個非常典型的跨層錯誤。底層完全正常，上層卻陷入哲學危機。
因此 RFC 式的描述應該是：

> 👍 MUST be interpreted as acknowledgement of receipt. It MUST NOT, without additional context, be interpreted as evidence of enthusiasm.

3.3 Retransmission and the Human RTO Problem

TCP 在 ACK 超時後會重傳，人類也有非常相似的實作：

> 在嗎

過了一會：

> 👀

再過一會：

> ?

然後：

> ??

最後：

> ???

這本質上就是 application-layer retransmission。問題在於很多人類 endpoint 沒有正確實現 exponential backoff，反而會在沒有回覆時縮短 retransmission timeout：

```
RTO1 = 10 min
RTO2 = 3 min
RTO3 = 30 sec
RTO4 = 8 sec
```

這是極其不健康的 congestion-control implementation，容易導致 social network saturation，甚至被 remote endpoint 直接執行：

> RST

或者更嚴重：

> BLOCK

因此，一個成熟的人類 networking stack 應該具有合理的 retransmission policy，並理解「沒有 ACK」本身有時也是一種 application-level signal。

4. UDP: Fire-and-Forget Social Communication

如果 TCP 的核心精神是「我要確認你收到」，那 UDP 的精神就是：

> 我已經發了，剩下的不歸 transport layer 管。

這種模式在群聊、朋友之間的突發思想，以及凌晨兩點的大腦異常活動中非常常見。
例如：

> 02:17
> 我突然想到如果企鵝都有 IPv6 address 會怎樣

緊接著：

> 💀

發送者隨後直接睡覺。這個行為幾乎可以用以下程式完整描述：

```
sendto(peer, packet, ...);
exit(0);
```

沒有 ACK，沒有 handshake，沒有 delivery guarantee，甚至沒有證據表明 sender 在發送後仍然在線。這就是純正的 UDP 社交。
群聊裡突然出現的：

> 💀 💀 💀

則更像 multicast 或 broadcast datagram。發送者沒有指定誰必須回，也不要求所有成員都理解。理解的人自然完成 application parsing，不理解的人安靜 drop packet 即可。

這也解釋了為什麼有些人的聊天人格非常 UDP：他們可以上午丟一個 👀，下午突然補一句完全不同的 🤡，晚上又發一篇論文，隔天再回到原來的 topic，完全不認為中間需要正式維護 conversation state。這並不是沒有狀態，而是把 state management 移交給了 application layer，而且實作相當自由。

5. QUIC: Modern Friendship with Multiplexing

相比 TCP，QUIC 更適合描述現代高頻聊天，因為現代人幾乎從不只維護一個 topic。真正的對話往往長這樣：

```
Stream 0: 👀 今晚吃什麼
Stream 4: 🤓 昨天那個線性代數問題
Stream 8: 😈 我想到一個非常離譜的主意
Stream 12: 🫠 今天累死
```

這些話題共享同一個 relationship context，卻可以獨立處理。對方完全可能先對 Stream 8 回：

```
💀
```

再處理 Stream 4，最後才回到 Stream 0 決定究竟吃漢堡還是可頌。

這正是 QUIC-style multiplexing 的社交價值：某個 stream 卡住，不應該阻塞其他 stream。

5.1 Head-of-Line Blocking and Conversation Design

假設某人連續發三件事：

```
1. 今晚去哪吃？
2. 我剛看到一篇有趣的論文。
3. 我想到一個非常危險的問題。
```

人類接收端幾乎不會按照嚴格 ordered application processing 回覆。第三條如果足夠離譜，可能立刻得到：

```
3 -> 👀
3 -> 😈
3 -> 💀
```

而第一條仍處於：

```
PENDING_RESTAURANT_SELECTION
```

這是一種非常自然的 stream-level scheduling。從這個角度看，高品質聊天通常並不是「按順序處理 queue」，而是根據 priority、interest 和 cognitive load 動態調度多個 streams。

5.2 0-RTT Friendship

陌生人建立聊天需要大量 setup：

```
你好。
你好。
最近怎麼樣？
還好，你呢？
```

這些其實都是 handshake overhead。

熟人則可能直接：

```
👀
```

對方：

```
😈
```

然後 application data 馬上開始。

這極其接近 0-RTT：雙方已經擁有足夠的 cached session state，因此不需要重新協商「我們是誰」、「怎麼聊天」、「什麼語氣是允許的」。

可以說，真正穩定的友情會逐漸把大量寒暄 cache 掉。

5.3 Connection Migration

QUIC 的另一個特點是 connection identity 可以不完全綁定傳統 IP/port tuple。這也非常符合現代社交：

```
Morning:
MacBook -> 👀

Afternoon:
iPhone -> 😈

Night:
Desktop -> 😈
```

設備變了、網路變了、出口 IP 變了，但對話仍然被雙方識別為同一個 session。

從這個角度來看，一段成熟的人類關係天然支援 connection migration。真正重要的不是 endpoint 的 IP，而是 shared conversational state。

6. WebSocket: Persistent Full-Duplex Nonsense

HTTP request-response 模型很適合查資料，但並不適合描述真正的朋友聊天。朋友不會遵循：

```
request
response
request
response
request
response
```

真正的聊天更接近一條 persistent full-duplex channel。任何一方都可以隨時 push：

👀

😈

😈

💀

😭

🥲

沒有人需要等待完整 request 才能發下一個 frame。

從這個角度看，朋友之間關係升級的一個重要事件其實是：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
```

最初可能還是：

最近怎麼樣？
還不錯。
你呢？

但某一刻突然：

👀

接著：

我有一個問題。

對方回：

😈

此時雙方已經從 `Polite Small Talk Protocol` 成功 upgrade 到 `Persistent Full-Duplex Nonsense Protocol`。

6.1 Ping, Pong, and Presence

WebSocket 還提供了一個非常自然的 👀 解釋：

```
👀 = PING
👀 = PONG
```

當 👀 單獨出現時，它甚至不需要攜帶真正的 semantic payload。它只是確認：

```
Are you alive?
Is this channel still open?
Can I send something strange?
```

對方回一個 👀，即可完成 presence check。之後 connection 可以繼續 idle，也可以立即進入大量 data transmission。

7. Server-Sent Events: The Human EventSource

有些聊天並不是 full-duplex 的，而更接近一個人持續輸出，另一個人偶爾處理。
例如：

```
👀 我突然想到一件事
🤡 等等
💀 不對
😈 我又想到另一個
```

這種模式高度接近 SSE：

```
Content-Type: text/event-stream

data: 👀
data: 🤡
data: 💀
data: 😈
```

發送端沒有要求每一條 event 都必須收到 response，它只是持續：

```
emit()
emit()
emit()
emit()
```

這種人可以被視為 walking `EventSource`。尤其在群聊中，有些人會持續 publish cognitive events，而其他人只是偶爾消費 stream。整個系統依然可以工作，前提是 event rate 沒有超過 receiver 的 cognitive bandwidth。
一旦超過，接收端就會進入：

```
🫠
```

也就是 local resource exhaustion。

8. WebRTC: When the Face Becomes a Side Channel

WebRTC 讓人類通信模型更複雜，因為此時不只存在文字 channel，還有 audio track、video track 與 data channel。最大的問題是：這些 channel 可能互相矛盾。
例如：

「你昨天是不是偷偷做了什麼？」

回覆：

```
audio track:
" 沒有啊。"

video track:
👀😳

data channel:
```

" 真的沒有"

Transport layer 完全正常，packet loss 甚至可能為零，但 application-level trust score 已經急劇下降。

這說明人類溝通不是單一資料流，而是一個 multimodal distributed system。語言可以說 `false`，臉部表情可以同時 emit `highly_suspicious=true`。在這裡 Emoji 甚至不需要被真正發送，人的面部表情本身就成了 physical representation of emoji semantics。

換句話說，Emoji 很可能只是人類在文字介面中模擬原本存在於視覺 side channel 的訊號。

9. MQTT: Mood as Publish/Subscribe State

MQTT 很適合描述「目前狀態」類 Emoji。假設一個人對外暴露以下 topics：

```
dorian/mood/current
dorian/study/current
dorian/idea/current
```

某一天 broker 裡的狀態可能是：

```
dorian/mood/current = 🫠
dorian/study/current = 🤓
dorian/idea/current = 😈
```

朋友甚至不需要主動 query：

你現在怎麼樣？

只需要：

```
SUBSCRIBE dorian/#
```

即可持續接收 state update。

這也可以很好地解釋社交平台上的 status、story、note 等功能。它們本質上都不像 request-response，更接近 publisher 把自己的狀態掛在某個 topic 上，由有興趣的人自行 subscribe。

9.1 Retained Messages

MQTT retained message 更有社交價值。假設凌晨最後一次 publish：

```
dorian/mood/current = 🫠
retain = true
```

第二天朋友上線，一 subscribe 就立即收到：

🫠

這就是 retained state。即使 publisher 當前沒有重新發送，broker 仍然保留最新狀態。

因此 🫠、😴、📚、✈️ 這類 Emoji 常常不只是 message，而是 state advertisement。

10. Emoji Reactions as Distributed Metadata

Discord、Slack、Telegram 等平台的 reaction emoji 又是另一種東西。它們並不是新的 message，而更像 metadata update。

例如原始訊息：

```
message_id = 4821
```

七個人加 💀，四個人加 😭，兩個人加 👀，從 API 思維來看更接近：

```
PATCH /messages/4821/reactions
```

最終狀態：

```
{
"message_id": 4821,
"reactions": {
"💀": 7,
"😭": 4,
```

```
"👀": 2
}
}
```

這種 reaction 非常高效，因為它同時提供 acknowledgement 和 semantic classification。💀 可以壓縮一句非常冗長的自然語言：

> 我已經收到並閱讀這條訊息，而且我認為其荒謬程度足以使我產生短暫的模擬性死亡反應。

壓縮後只有：

> 💀

因此從 protocol design 的角度看，Emoji reaction 是一種極具效率的 low-bandwidth semantic metadata system。

11. Proposed Emoji Control Codes

基於以上觀察，我們可以建立一個非正式 Emoji control table：

Emoji	Proposed protocol semantic	Typical human interpretation
👀	SYN / PING / PROBE / RESUME	我在看、繼續、有情況
👍	ACK	收到
🤔	PROCESSING	我正在處理
🤓	ANALYTICAL_MODE	進入認真分析
😈	EXPERIMENTAL_FLAG	接下來可能有奇怪內容
🤡	IRONY_FLAG	不要完全按字面解析
😭	EMOTIONAL_OVERFLOW	太好笑、太慘、受不了
💀	REMOTE_CRASH	笑死、荒謬、處理失敗
😵‍💫	LOCAL_OVERLOAD	CPU 100%，人融化了
😇	RECOVERY	假裝一切正常
🥺	RESPONSE_REQUEST	可以回一下嗎
🤏	FIN-ACK	收到，任務結束
👋	FIN	離開會話
🙂	UNDEFINED_BEHAVIOR	需上下文解析

其中最需要安全警告的是：

> 🙂

理論 specification：

```
friendly smile
```

現實 implementation：

```
okay
fine
interesting
sure
I disagree
I am annoyed
this conversation is over
you have approximately three seconds
```

因此：

> **Implementations MUST NOT parse 🙂 without conversational context.**

這可能是整個 Emoji protocol 中最嚴重的 interoperability problem。

12. The Dorian Emoji Transport Protocol

綜合以上觀察，我們可以提出一個完全非標準、也沒有人要求建立的實驗性協議：

DETP: Dorian Emoji Transport Protocol

其理想協議棧如下：

```
Application Payload
        ↓
Emoji Pragmatic Layer
        ↓
Persistent Conversational Session
        ↓
QUIC-like Multiplexing
        ↓
Network
```

並額外支援：

```
WebSocket -> persistent duplex conversation
WebRTC    -> realtime audiovisual side channels
SSE       -> unilateral thought streaming
MQTT      -> mood and presence publication
```

核心 control codes：

```
👀 INIT / PROBE / RESUME
😁 ANALYSIS_MODE
😈 EXPERIMENTAL_PAYLOAD
🤡 IRONY_FLAG
💀 REMOTE_EXCEPTION
🫠 LOCAL_RESOURCE_EXHAUSTION
😇 RECOVERY_COMPLETE
```

一個典型 session：

```
👀 -> 😁 -> 😈 -> 🤡 -> 💀 -> 🫠 -> 😇 -> 👀
```

最值得注意的是：

```
FIN
```

通常沒有真正出現。

因為良好的朋友聊天不是 short-lived request-response，而是一個長期 persistent connection。它可以 idle 幾個小時、幾天，甚至幾個月，之後只需一個：

👀

就重新恢復 application traffic。

13. Failure Modes and Security Considerations

任何 protocol 都有 attack surface，Emoji protocol 也不例外。

第一個明顯問題是 **Social SYN Flood**：

👀
👀
👀
👀
👀

如果大量未完成的 conversation initiation 堆積，remote endpoint 可能出現 cognitive backlog。這和 SYN flood 的精神高度一致：攻擊者不一定傳輸大量實際 payload，只需要持續要求 connection state。

第二個問題是 **ambiguous control codes**。例如 🙂 在不同文化、年齡、關係與聊天情境中可能被完全不同地解碼，因此非常容易造成 protocol mismatch。發送端認為自己 emit 了：

```
friendly_ack
```

接收端卻解析成：

```
passive_aggressive_warning
```

第三個問題是 **ACK ambiguity**。👍 可以表示 genuine approval，也可以只是「我知道了」。如果 sender 將 transport-layer acknowledgement 錯誤提升成 application-level endorsement，就可能產生錯誤決策。

第四個問題是 **retransmission storm**。連續的 ?、??、??? 在缺乏 congestion control 時會迅速降低 communication quality。

第五個問題是 **context loss**。Emoji 對 shared state 的依賴極高。一個熟悉的朋友看到：

😈

可能立即知道「又有奇怪問題來了」，而一個陌生人可能只能得到非常模糊的 threat model。換言之，Emoji encoding ratio 很高，但前提是雙方共享大量 decoder state。

這正是壓縮系統常見的 trade-off：越依賴 shared dictionary，傳輸本身就越短。

14. Why Emoji Works So Well

如果暫時放下梗，Emoji 真正有趣的地方其實在於它展示了人類溝通的一個基本事實：**conversation 從來不只有 semantic content**。

一條訊息至少包含：

```
semantic payload
+ tone
+ timing
+ relationship state
+ expected response
+ emotional weight
+ prior context
```

純文字往往很難完整編碼這些資訊。面對面交流可以靠語速、停頓、眼神、表情和手勢補足，但文字聊天缺乏這些 side channels。Emoji 因此成為一種廉價而高效的 metadata layer。

例如：

> 可以。

和：

> 可以 😈

在字面命題上幾乎相同，但後者顯然存在額外 protocol information。

同樣：

> 你繼續說。

可以被壓縮成：

👀

而：

> 我已經看到了這個荒謬內容，並且因為它過於好笑而出現短暫語言失能。

可以被壓縮成：

💀

這意味著 Emoji 的關鍵優勢可能不是 expressive bandwidth，而是 **semantic density**。每一個可感知的 Emoji grapheme 或 Emoji sequence 本身攜帶的命題資訊並不多，但和 shared context 結合後，可以恢復出大量社交含義。

15. Discussion: Humans as Stateful Endpoints

網路協議模型還揭示了一個更有趣的事實：人類是高度 stateful 的 endpoint。
如果完全沒有 shared state，那麼：

👀

幾乎不可解碼。
但如果兩個人已經聊了幾年，建立了大量 shared context，同一個 👀 可能精確表示：

```
I have noticed something.
I expect you to understand what category of thing this is.
I am not yet revealing the payload.
You are invited to continue.
This will probably become weird.
```

也就是說，真正承載資訊的並不只有 packet，還包括 connection history。
這和 stateful protocol 很像：短 packet 能工作，是因為雙方 endpoint 已經保存大量先前狀態。
因此，一段關係越熟悉，往往越不需要 verbose protocol。陌生人需要完整句子，熟人可以只發 Emoji；極端情況下，甚至只發：

？

接收端就能恢復完整 context。
從資訊理論角度看，這可能是一種高 shared-state communication system。從普通人角度看，就是：

「他知道我在說什麼。」

16. Conclusion

Emoji 沒有取代自然語言。更準確地說，它替自然語言補上了一個 control plane。
文字主要負責：

```
DATA
```

Emoji 則大量負責：

```
STATE
TONE
ACKNOWLEDGEMENT
INTENT
ERROR REPORTING
SESSION CONTROL
PRIORITY
```

因此：

👀

在不同場景下可以同時接近：

```
SYN
PING
SUBSCRIBE
ATTENTION_REQUEST
SESSION_RESUME
```

而：

💀

可以被合理描述為：

```
500 Internal Human Error
```

🤡 則近似：

```
Warning:
The following payload SHOULD NOT be interpreted literally.
```

🥲 表示：

```
503 Human Temporarily Unavailable
Reason: Resource Exhaustion
```

至於 🙂：

```
451 Semantic Meaning Unavailable For Legal Reasons
```

最終，我們可以提出一個並不嚴肅、但也沒有那麼不嚴肅的結論：

> 現代人類並沒有停止使用複雜的通訊協議。相反，他們正在使用一套高度 stateful、context-sensitive、multiplexed、occasionally unreliable 的社交網路協議。只不過其中一部分 control frame，現在被畫成了黃色小臉。

Appendix A. Minimal Emoji RFC

```
RFC XXXX
Emoji Pragmatic Transport Semantics
Status: Extremely Experimental

1. 👀 SHOULD initiate, resume, probe, or inspect a
conversational session.

2. 👍 MUST acknowledge receipt. It MUST NOT be interpreted
as enthusiasm without additional application context.

3. 🤡 SHOULD indicate irony, self-deprecation, absurdity,
or non-literal interpretation.

4. 💀 MUST NOT be interpreted as literal death unless
corroborated by an out-of-band mechanism.

5. 🥲 SHOULD indicate temporary local resource exhaustion.

6. 😈 SHOULD indicate that unusual application payload
may follow.

7. 🙂 MUST be considered context-dependent and MAY contain
passive-aggressive semantics.

8. Endpoints SHOULD support multiple concurrent
conversational streams.

9. Endpoints SHOULD implement reasonable retransmission
backoff and MUST NOT repeatedly transmit "???" at
sub-minute intervals.

10. A peer receiving 👀 MAY respond with 👀.

11. A long-idle session MAY be resumed using a single 👀
if sufficient shared state exists.

12. Implementations MUST accept that human protocol behavior
is undefined in several edge cases.
```

Appendix B. Example Trace

```
22:03:14 A -> B 👀
22:03:19 B -> A 👀
22:03:22 A -> B 😈
22:03:24 B -> A 💀
22:03:27 A -> B 🤡
22:03:32 B -> A 😝
22:03:40 A -> B 😇
22:04:11 B -> A 👀
```

Protocol analysis:

```
Connection state: ESTABLISHED
RTT: acceptable
Packet loss: negligible
Semantic stability: questionable
Application throughput: high
Conversation quality: excellent
Standards compliance: none
```

Appendix C. Suggested MIME Type

For future standardisation, the following media type is proposed:

```
Content-Type: application/vnd.human.emoji+utf8
Transport: QUIC
Connection: persistent
Sanity: optional
```

No working group has requested this proposal. This has not stopped us.