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

DORIAN VERLAINE

Abstract. This paper proposes a semi-serious model in which emoji in modern instant messaging are not merely pictograms of emotion, but may instead be understood as a set of lightweight control signals within the human social protocol stack. Using network models including TCP, UDP, QUIC, WebSocket, Server-Sent Events, WebRTC and MQTT, we analyse how common emoji such as 👀, 👍, 🤡, 💀 and 🫠 perform functions including connection establishment, acknowledgement, state synchronisation, tone marking, error handling and session resumption. We further propose a fictional Emoji Pragmatic Layer and discuss the properties of emoji with regard to high semantic density, low transmission cost, contextual dependence and protocol ambiguity. Our findings suggest that, in a surprisingly large number of scenarios, 👀 behaves suspiciously like SYN, PING, SUBSCRIBE or SESSION_RESUME, while 💀 may reasonably be modelled as 500 Internal Human Error. This paper makes no claim of compliance with any IETF, IEEE, ACM or normal-human-behaviour specification.

1. Introduction

After inventing natural language, humanity promptly spent the next several thousand years trying to type fewer characters. From abbreviations in letters and text messages to Internet slang and, eventually, emoji, the history of communication is, in one sense, also a history of continuously compressing the Application Payload. Consider a perfectly ordinary example. Someone could type, "I have seen what you just said, I have developed a certain degree of interest in the matter, and I would like you to continue." In an instant-messaging application, however, they are far more likely to send:

👀

Strangely, this message contains almost no explicit propositional content, yet another human endpoint can usually decode it correctly. The recipient generally understands that it does not mean "I possess two eyes", but something closer to "I'm watching", "go on", "what's going on?", "you had better finish that thought", or even "I have detected that you are about to say something mildly abnormal".

This suggests that what emoji transmit is often not semantic payload in the traditional sense, but **pragmatic metadata**: tone, relationship state, expectations of response, emotional weight, whether irony is present, whether a conversational session should be established, and whether the sender's current process remains operationally stable.

This paper therefore proposes a central hypothesis:

> **Emoji may be treated as a lightweight control protocol operating above natural language and below the social application layer.**

If this hypothesis holds, then 👀 may function as a connection probe, 👍 as an acknowledgement, 🤡 as an irony flag, 🫠 as local resource exhaustion, and 💀 as evidence that the remote application process has temporarily crashed because the payload was excessively absurd.

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

This model is, of course, not intended as a rigorous linguistic description of human language. It is, however, remarkably well suited to explaining the enormous quantity of non-standard network behaviour observed daily by a computer science student using chat applications.

2. A Proposed Human Communication Stack

For convenience, modern text-based conversation can be represented approximately by the following protocol stack:

Layer	Payload or role
Application Layer	"What shall we eat tonight?"
Emoji / Pragmatic Layer	👀 🤡 😈 💀 🫠 🙂
Conversational Session Layer	Friendship / Context / Shared State
Transport Model	TCP / UDP / QUIC / WebSocket
Physical Interface	Fingers / Screen / Voice / Face

In this model, ordinary text is primarily responsible for describing "what was said", whereas emoji are more concerned with specifying "how this packet should be decoded". Consider the following two messages, which contain exactly the same textual payload:

```
Are you free tonight?
Are you free tonight? 👀
```

Anyone with even basic compatibility with the human social protocol knows that these are not quite equivalent. The first may be a neutral availability query. The second may carry expectation, implication, tentative probing, curiosity, or even a connection initiation whose full payload has not yet been disclosed. In other words, emoji do not merely add information. They may modify the application semantics of the entire packet.

This is remarkably similar to flags, headers and control frames in network protocols. The data itself may be unchanged, while the method of decoding it has changed substantially.

3. TCP: Reliable Human Interaction

The defining properties of TCP include connection orientation, reliable delivery, ordered delivery and a near-obsessive concern with acknowledgement. This makes it an excellent model for a certain class of human social behaviour: not merely sending a message, but also caring very deeply about whether the other party received it, understood it, replied to it, and why they have left it on read for three minutes without sending an ACK.

3.1 👀 as SYN

Consider the minimal social packet:

```
👀
```

It contains no explicit topic, no complete question and no meaningful payload, yet it is frequently capable of establishing a conversation. We may therefore define, approximately:

```
👀 = SYN
```

A typical handshake might look like this:

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

If we insist on forcing this through a TCP state machine, we obtain:

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

At this point both parties enter the `ESTABLISHED` state, despite the actual topic not having appeared yet. This is an extremely efficient, and simultaneously rather dangerous, method of session establishment. The final 😈 is especially noteworthy: it usually means that the connection has succeeded, but the safety properties of the subsequent payload can no longer be guaranteed.

3.2 👍 as ACK

👍 is another almost perfect transport-layer emoji. Its core meaning is often not "I like this", but simply:

```
Packet received.
```

The difficulty is that transport success does not imply application success. Suppose:

```
A: I spent two hours putting this proposal together. What do you
think?
B: 👍
```

From the transport layer's perspective, this is a success. The message arrived, the peer acknowledged receipt, and the connection is healthy. From the social application layer, however, the result may instead be:

```
ERROR: meaningful_response_not_found
```

This is a classic cross-layer failure. The lower layers are functioning perfectly, while the upper layer has entered a philosophical crisis.

An RFC-style description should therefore read:

> 👍 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 retransmits after an ACK timeout. Humans have implemented an alarmingly similar mechanism:

```
You there?
```

A little later:

```
👀
```

Then:

```
?
```

Then:

```
??
```

Finally:

```
???
```

This is, in essence, application-layer retransmission. The problem is that many human endpoints do not implement exponential backoff correctly. Instead, they shorten the retransmission timeout when no reply is received:

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

This is an exceptionally unhealthy congestion-control implementation. It can lead to social network saturation and may ultimately cause the remote endpoint to issue:

```
RST
```

or, in more severe cases:

```
BLOCK
```

A mature human networking stack should therefore implement a sensible retransmission policy and understand that, at times, the absence of an ACK is itself an application-level signal.

4. UDP: Fire-and-Forget Social Communication

If the central spirit of TCP is "I need to know that you received this", then the spirit of UDP is:

> I have sent it. Everything after that is no longer the transport layer's problem.

This model appears frequently in group chats, spontaneous thoughts between friends, and abnormal brain activity at two o'clock in the morning.

For example:

```
02:17
I have just wondered what would happen if every penguin had an IPv6 address.
```

Immediately followed by:

> 💀

The sender then goes directly to sleep. The entire behaviour can almost be described by the following programme:

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

No ACK, no handshake, no delivery guarantee, and not even any evidence that the sender remains online after transmission. This is pure UDP socialisation.

A sudden:

> 💀 💀 💀

in a group chat behaves more like a multicast or broadcast datagram. The sender does not specify who must respond, nor do they require every member to understand it. Those who can parse the application payload do so; everyone else may quietly drop the packet.

This also explains why some people have extremely UDP-like conversational personalities. They may send a 👀 in the morning, append an entirely unrelated 🤡 in the afternoon, drop a research paper into the channel in the evening, then return to the original topic the next day without believing that any formal conversation-state maintenance was required. This is not necessarily statelessness. It is simply the delegation of state management to the application layer, implemented with considerable creative freedom.

5. QUIC: Modern Friendship with Multiplexing

Compared with TCP, QUIC is a better model for modern high-frequency conversation because people almost never maintain only one topic. Real conversations tend to look more like this:

```
Stream 0: 👀 What shall we eat tonight?
Stream 4: 🤓 That linear algebra problem from yesterday
Stream 8: 😈 I have just thought of an extremely ridiculous
proposition
Stream 12: 🫠 I am absolutely exhausted today
```

These topics share the same relationship context while remaining independently processable. The other party may reply to Stream 8 first with:

> 💀

then handle Stream 4, and only afterwards return to Stream 0 to decide whether dinner should consist of hamburgers or croissants.

This is precisely the social value of QUIC-style multiplexing: one blocked stream should not block the others.

5.1 Head-of-Line Blocking and Conversation Design

Suppose someone sends three consecutive messages:

```
1. Where shall we eat tonight?
2. I have just found an interesting paper.
3. I have thought of an extremely dangerous question.
```

A human receiving endpoint is unlikely to enforce strict ordered application processing. If the third message is sufficiently outrageous, it may immediately receive:

```
3 -> 👀
3 -> 😈
3 -> 💀
```

while the first remains in:

```
PENDING_RESTAURANT_SELECTION
```

This is a perfectly natural form of stream-level scheduling. From this perspective, high-quality conversation is not about processing a queue in order. It is about dynamically scheduling multiple streams according to priority, interest and cognitive load.

5.2 0-RTT Friendship

Conversation between strangers requires considerable setup:

```
Hello.
Hello.
How have you been?
Not bad. You?
```

All of this is handshake overhead.

Friends may instead begin with:

👀

The response:

😈

and application data begins immediately.

This is extremely close to 0-RTT. Both parties already possess sufficient cached session state and therefore have no need to renegotiate who they are, how they speak to one another, or what tonal parameters are permitted.

One might say that a genuinely stable friendship gradually caches away a large quantity of small talk.

5.3 Connection Migration

Another property of QUIC is that connection identity need not be bound entirely to the traditional IP/port tuple. This is also highly compatible with modern social interaction:

```
Morning:
MacBook -> 👀

Afternoon:
iPhone -> 🤡

Night:
Desktop -> 😈
```

The device has changed, the network has changed, and the egress IP has changed, yet both parties still recognise the conversation as the same session.

From this perspective, a mature human relationship has native support for connection migration. What matters is not the endpoint's IP address, but the shared conversational state.

6. WebSocket: Persistent Full-Duplex Nonsense

The HTTP request-response model is excellent for retrieving information, but rather poor at describing genuine conversation between friends. Friends do not normally obey:

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

Real conversation is much closer to a persistent full-duplex channel. Either party may push at any moment:

👀
🤡
😈
💀
😅
🫠

Nobody needs to wait for a complete request before sending the next frame.
From this perspective, one important event in the escalation of a friendship is effectively:

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

At first, the interaction may still be:

```
How have you been?
Not bad.
You?
```

But at some point, somebody suddenly sends:

👀

followed by:

```
I have a question.
```

The peer replies:

😈

At that moment, both endpoints have successfully upgraded from the `Polite Small Talk Protocol` to the `Persistent Full-Duplex Nonsense Protocol`.

6.1 Ping, Pong, and Presence

WebSocket also gives us a particularly natural interpretation of 👀:

```
👀 = PING
👀 = PONG
```

When 👀 appears alone, it need not carry any genuine semantic payload at all. It merely checks:

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

The peer may reply with another 👀, completing the presence check. The connection may then remain idle or immediately enter heavy data transmission.

7. Server-Sent Events: The Human EventSource

Some conversations are not full-duplex. They are closer to one person continuously producing output while the other processes it only occasionally.

For example:

```
👀 I have just thought of something
🤡 Wait
💀 No, that is wrong
😈 I have thought of something else
```

This pattern strongly resembles SSE:

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

data: 👀
data: 🤡
data: 💀
data: 😈
```

The sender does not require a response to every event. They simply continue:

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

Such a person may reasonably be regarded as a walking `EventSource`. This is particularly visible in group chats, where some individuals continuously publish cognitive events while everyone else merely consumes the stream from time to time. The system remains operational provided that the event rate does not exceed the receiver's cognitive bandwidth.

Once that threshold is exceeded, the receiving endpoint enters:

🫠

which is to say, local resource exhaustion.

8. WebRTC: When the Face Becomes a Side Channel

WebRTC complicates the human communication model because we now have not only a text channel, but also an audio track, a video track and a data channel. The largest problem is that these channels may contradict one another.

For example:

> "Did you secretly do something yesterday?"

Response:

```
audio track:
"No."

video track:
👀😳

data channel:
"Really, I didn't."
```

The transport layer is functioning perfectly and packet loss may even be zero, yet the application-level trust score has fallen sharply.

This demonstrates that human communication is not a single data stream, but a multimodal distributed system. Language can transmit `false` while facial expression simultaneously emits `highly_suspicious=true`. Here, emoji need not even be explicitly transmitted. Human facial expressions themselves become the physical representation of emoji semantics.

In other words, emoji may simply be a textual-interface simulation of signals that originally existed in a visual side channel.

9. MQTT: Mood as Publish/Subscribe State

MQTT is well suited to describing emoji that represent current state. Suppose a person exposes the following topics:

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

On a particular day, the broker state might be:

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

Friends do not even need to issue an active query such as:

```
How are you right now?
```

They need only:

```
SUBSCRIBE dorian/#
```

and may continuously receive state updates.

This also explains features such as statuses, stories and notes on social platforms. They are fundamentally unlike request-response interactions. They are closer to a publisher exposing its state on a topic and allowing interested parties to subscribe at will.

9.1 Retained Messages

MQTT retained messages are even more socially useful. Suppose the final publication before bed is:

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

A friend comes online the next day, subscribes, and immediately receives:

🥲

That is retained state. Even if the publisher has not sent anything new, the broker still preserves the most recent value.

Thus emoji such as 🥲, 😴, 📚 and ✈️ are often not merely messages, but state advertisements.

10. Emoji Reactions as Distributed Metadata

Reaction emoji on platforms such as Discord, Slack and Telegram are something else again. They are not new messages, but are closer to metadata updates.

Suppose the original message is:

```
message_id = 4821
```

Seven people add 💀, four add 😭, and two add 👀. From an API perspective, this looks rather more like:

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

The final state becomes:

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

```
"👀": 2
}
}
```

This kind of reaction is remarkably efficient because it simultaneously provides acknowledgement and semantic classification. 💀 can compress an excessively verbose natural-language message such as:

> I have received and read this message, and I consider its level of absurdity sufficient to induce a brief simulated-death response.

into nothing more than:

> 💀

From a protocol-design perspective, emoji reactions therefore constitute an exceptionally efficient low-bandwidth semantic metadata system.

11. Proposed Emoji Control Codes

Based on the observations above, we can construct an informal Emoji control table:

Emoji	Proposed protocol semantic	Typical human interpretation
👀	SYN / PING / PROBE / RESUME	I'm watching, go on, what's going on?
👍	ACK	Received
🤔	PROCESSING	I'm processing this
🤓	ANALYTICAL_MODE	Entering serious analysis mode
😈	EXPERIMENTAL_FLAG	Strange content may follow
🤡	IRONY_FLAG	Do not parse entirely literally
😭	EMOTIONAL_OVERFLOW	Too funny, too tragic, cannot cope
💀	REMOTE_CRASH	I'm dead, absurdity overflow, processing failed
🫠	LOCAL_OVERLOAD	CPU at 100%, human has melted
😇	RECOVERY	Pretending everything is normal again
🥺	RESPONSE_REQUEST	Could I have a reply, please?
🤙	FIN-ACK	Received, task complete
👋	FIN	Leaving the conversation
🙂	UNDEFINED_BEHAVIOUR	Requires contextual parsing

The one requiring the most serious safety warning is:

> 🙂

Theoretical specification:

```
friendly smile
```

Real-world implementation:

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

Therefore:

Implementations MUST NOT parse 🙂 without conversational context.

This may be the most severe interoperability problem in the entire Emoji protocol.

12. The Dorian Emoji Transport Protocol

Combining the observations above, we can propose an entirely non-standard experimental protocol that nobody requested:

DETP: Dorian Emoji Transport Protocol

Its ideal protocol stack is:

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

with additional support for:

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

Core control codes:

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

A typical session:

👀 -> 🤓 -> 😈 -> 🤡 -> 💀 -> 🫠 -> 😇 -> 👀

The most notable feature is that:

```
FIN
```

usually never truly appears.

Good conversation between friends is not a short-lived request-response exchange. It is a long-term persistent connection. It may idle for hours, days or even months, after which a single:

👀

is sufficient to resume application traffic.

13. Failure Modes and Security Considerations

Every protocol has an attack surface, and the Emoji protocol is no exception.

The first obvious problem is a **Social SYN Flood**:

👀
👀
👀
👀
👀

If too many incomplete conversation initiations accumulate, the remote endpoint may develop a cognitive backlog. This is very close in spirit to a SYN flood. The attacker need not transmit a large amount of actual payload. They need only continue requesting connection state.

The second problem is **ambiguous control codes**. 🙂, for example, may be decoded completely differently across cultures, age groups, relationships and conversational contexts, making protocol mismatch exceptionally likely. The sender may believe they emitted:

```
friendly_ack
```

while the receiver parses:

```
passive_aggressive_warning
```

The third problem is **ACK ambiguity**. 👍 may indicate genuine approval or merely "acknowledged". If the sender mistakenly promotes a transport-layer acknowledgement into application-level endorsement, incorrect decisions may follow.

The fourth problem is a **retransmission storm**. Repeated `?`, `??` and `???` messages, in the absence of congestion control, can degrade communication quality rapidly.

The fifth problem is **context loss**. Emoji depend heavily on shared state. A close friend who receives:

😈

may immediately understand that "another strange question is incoming", while a stranger may be left with little more than a highly ambiguous threat model. In other words, emoji achieve a very high encoding ratio, but only because both endpoints share a substantial quantity of decoder state.

This is the familiar trade-off of compression systems: the more heavily one relies on a shared dictionary, the shorter the transmission itself can become.

14. Why Emoji Works So Well

If we temporarily set the joke aside, the genuinely interesting thing about emoji is that they expose a basic fact of human communication: **conversation has never consisted solely of semantic content.**

A single message contains at least:

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

Plain text often encodes these dimensions poorly. Face-to-face communication can supplement language with speaking rate, pauses, gaze, facial expression and gesture, but text chat lacks these side channels. Emoji therefore function as a cheap and efficient metadata layer.

For example:

```
All right.
```

and:

```
All right 😈
```

are almost identical at the level of literal proposition, yet the latter plainly contains additional protocol information.

Likewise:

```
Go on.
```

can be compressed into:

👀

while:

```
I have seen this absurd content and, because it is excessively funny, have temporari
```

can be compressed into:

💀

This suggests that the central advantage of emoji may not be expressive bandwidth, but **semantic density**. Each perceptible emoji grapheme or emoji sequence carries relatively little propositional information by itself, yet when combined with shared context it can recover a remarkably large amount of social meaning.

15. Discussion: Humans as Stateful Endpoints

The network-protocol model also reveals something more interesting: human beings are highly stateful endpoints.

Without shared state, the following packet:

👀

is almost impossible to decode.

But if two people have been talking for years and have accumulated substantial shared context, the same 👀 may mean, with surprising precision:

```
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.
```

In other words, the information is carried not only by the packet, but also by connection history.

This resembles a stateful protocol. Short packets work because both endpoints have retained substantial prior state.

The more familiar a relationship becomes, the less verbose its protocol tends to be. Strangers require complete sentences. Friends can send only emoji. In extreme cases, one may send nothing more than:

```
?
```

and the receiving endpoint can reconstruct the full context.

From an information-theory perspective, this may be described as a high shared-state communication system. From an ordinary human perspective, it is simply:

"He knows what I mean."

16. Conclusion

Emoji have not replaced natural language. More accurately, they have supplied natural language with a control plane.

Text is primarily responsible for:

```
DATA
```

while emoji frequently handle:

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

Thus:

👀

may, depending on context, approximate:

```
SYN
PING
SUBSCRIBE
ATTENTION_REQUEST
SESSION_RESUME
```

while:

💀

may reasonably be described as:

```
500 Internal Human Error
```

🤡 approximates:

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

🫠 means:

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

As for 🙂:

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

We may therefore arrive at a conclusion that is not entirely serious, but perhaps not entirely unserious either:

> Modern humans have not stopped using complex communication protocols. On the contrary, they are operating a highly stateful, context-sensitive, multiplexed and occasionally unreliable social networking protocol. It merely happens that some of its control frames are now drawn as little yellow faces.

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 behaviour 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.