

Coded Hashes of Arbitrary Images

(or: the last frontier of emoji encoding)

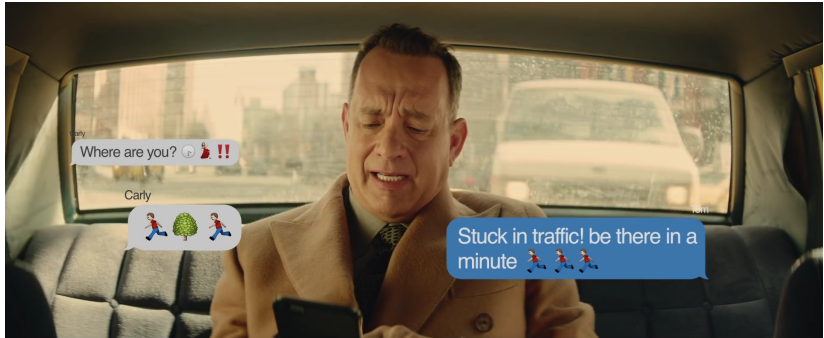
Steven R. Loomis
individual contribution

Keith Winstein
Stanford University

Jennifer 8. Lee
Emojination

May 10, 2016

Inline pictographs (emoji) are a popular means of expression.



"I Really Like You" (Jepsen 2015)

Status quo: prerequisites to successful interchange

- ▶ New emoji assigned a code point in the normative Unicode/UCS specifications.
- ▶ Sender's keyboard and font includes new emoji.
- ▶ Receiver's font includes new emoji.

Challenges of status quo

- ▶ ① Sender, ② Unicode Standard, and ③ receiver must each upgrade before interchange can be successful.
- ▶ Process takes > 18 months end-to-end.
- ▶ Version mismatch leads to lack of interoperability
- ▶ Unlikely to see an end to the creation of new pictographs.
- ▶ Technical standards committees may not be equipped to serve as juries of universality, taste, or unmet need of a new emoji.

Unicode Technical Report #51

“The longer-term goal for implementations should be to support embedded graphics, in addition to the emoji characters. Embedded graphics allow arbitrary emoji symbols, and are not dependent on additional Unicode encoding.”

Our proposal: Implementation-defined emoji

1. **Sender** implementations can define their own emoji.
2. **Unicode** provides a normative encoding that **uniquely identifies** the emoji.
3. Non-normative what **receiver** does if it doesn't already know about the emoji (a few reasonable options).

① Straw-person format for implementation-defined emoji

Canonical Emoji Description could be a JSON object with:

- ▶ name (for accessibility)
- ▶ type (MIME media-type)
- ▶ image (backup rendering if no local glyph exists)

② Uniquely identifying an implementation-defined emoji

- ▶ Encode fallback character [any base character]
- ▶ With sequence of combining characters, encode SHA-256 **secure hash** of Canonical Emoji Description
- ▶ Combining characters are “Coded Hash of an Arbitrary Image” (CHAI) characters.

<basechar> + <CHAI char> + <CHAI char> + <CHAI char> + ...




- ▶ Sender encodes as many bits of the hash as it wants.
Intention: infeasible to make two distinct emoji with same hash.

Aside: How many bytes will a CHAI take on the wire?

- ▶ If allocated a plane, each CHAI char represents 16 bits of hash.
- ▶ With 256 code points, each CHAI char represents 8 bits of hash.
- ▶ Code points outside BMP take 4 bytes each in UTF-8/16/32.

⇒ A 96-bit hash prefix will take **24 bytes** or **48 bytes** on the wire.

- ▶ Today, emoji ZWJ sequences can require 27 bytes of UTF-8.

 27 bytes +  27 bytes +  27 bytes

③ At the receiver, several possibilities

- ▶ Receiver recognizes the identifier and displays a local glyph
- ▶ Receiver recognizes identifier and displays image from the Canonical Emoji Description
- ▶ Receiver displays fallback glyph (base character) and prompts user: “Would you like to upgrade your emoji set?”
 - ▶ Then downloads new emoji database from vendor
- ▶ If protocol permits, receiver prompts user: “Retrieve unknown emoji from sender?”
 - ▶ Asks sender: “Send Canonical Emoji Description for hash xyz”

How will vendors exchange Canonical Emoji Descriptions?

- ▶ Emoji Subcommittee could publish a new emoji database every week, with much-reduced approval threshold.
 - ▶ Could simply be a compendium of Canonical Emoji Descriptions submitted by members of the Unicode Consortium
- ▶ Direct interchange among vendors
- ▶ Emoji search engine
- ▶ Community emoji repository
 - ▶ Could serve weekly dump of top 99.9% of emoji by usage

Unicode Technical Report #51

“There are also privacy aspects to implementations of embedded graphics: if the graphic itself is not packaged with the text, but instead is just a reference to an image on a server, then that server could track usage.”

- ▶ For unrecognized identifiers, need to prompt user before loading. Similar to “load external images?” prompt in email client.
- ▶ **Expectation:** if vendors provide regular updates to clients, unrecognized emoji will be rare.

Benefits to Unicode

Unicode can get out of the business of **approving** emoji and simply provide a means for **uniquely identifying** arbitrary emoji in plain text.

Benefits to senders

Sender implementers can create their own emoji without waiting for standardization process.

Benefits to receivers

Receivers can download new emoji from their vendor or from senders, without needing to upgrade OS or install new fonts.

Objections

- ▶ “The encoding will be really long.”
 - ▶ A 104-bit hash is still shorter than 🥰 (27-byte UTF-8 sequence)
- ▶ “Allocating a whole plane? Are you crazy?”
 - ▶ Could use just 256 code points, but doubles coded length
- ▶ “How do I know if the receiver can display my emoji?”
 - ▶ Even today, no assurance receiver has the right Unicode version and font. With CHAI, receivers will be built to retrieve new emoji over time.
- ▶ “This isn’t character encoding.”
 - ▶ We think it’s similar to the Regional Indicator Symbols: a mechanism for *identifying* a pictograph from a set outside the normative Unicode/UCS.
- ▶ Would we have to specify JSON and SHA-256 specifically?
 - ▶ Not necessarily, but seems like a good idea for consistency.
- ▶ Could this be done in a Private Use Area?

Coded Hashes of Arbitrary Images

Unicode can get out of the business of **approving** emoji and simply provide a means for **uniquely identifying** arbitrary emoji in plain text.

Sender implementers can create their own emoji without waiting for standardization process.

Receivers can download new emoji from their vendor or from senders, without needing to upgrade OS or install new fonts.

Steven R. Loomis
srl@icu-project.org
individual contribution

Keith Winstein
keithw@cs.stanford.edu
Stanford University

Jennifer 8. Lee
jenny@jennifer8lee.com
Emojination