OPES Working Group

Callout Protocol Design: Major Decision Points

OPES Callout Protocol (OCP)

Location: OPES dispatcher $\Longleftrightarrow$ OPES callout server

Purpose: Adaptation of application messages

Apps: HTTP, RTSP, SMTP, but application agnostic

Features: Internet-friendly, fast, efficient, simple

Performance benchmark:
no-adaptation overhead of two application proxies.

Design decision points

- Current decision points (March)
- Future decision points (April)

Current decision points

Initiation: Which side can send unsolicited OPES messages?

ACK: Responses to OPES messages: required, optional, none?

ACKs: Can one OPES message trigger more than one response?

Granularity: What application message parts are passed or addressed?

Copy: Is application data copied or moved to the other side?

Priority: Can OPES messages be given a handling priority?

Future decision points

- Transport binding
  (TCP, SCTP, BEEP/TCP, HTTP/TCP, SOAP/?, ...)
- Message encoding
  (XML, MIME, simple XML, binary MIME)
- Application protocol binding
  (HTTP, SMTP, RTSP, ...)
- Error handling
  (lenient, strict, ...)
- ignore these as long as we can

Initiation: Who can talk first?

- OPES dispatcher is a client (should always talk first), callout server is a server (should never talk first)
- specific roles simplify protocol
- ICAP has clear client and server roles

Initiation: Who can talk first? (cont.)

but: callout server may need extra information (e.g., a content-specific query for OPES rules or user preferences)

but: required keep-alive mechanism violates simple roles

but: feature negotiation may violate simple roles

but: callout server may send several "responses"
(dispatcher must be ready for "unsolicited" messages)

Initiation: Who can talk first? (cont.)

- initiate *what*?
- dispatcher MUST initiate OPES connections
- dispatcher MUST initiate OPES transactions in reaction to application transactions
- other kinds of exchanges (meta-queries, keep-alives, feature negotiations) can be initiated by either side
- *naturally*: exchange type defines who can talk first!

Current decision points (check)

Initiation: Depends on message exchange

ACK: ⇐=

ACKs:

Granularity:

Copy:

Priority:

ACK: responses required, optional, none?

- required ACKs simplify protocol
  (every request has a matching response)

- some messages require responses
  (e.g., to support required keep-alive mechanism)

- ACKs tell us more about the other side state, speed

ACK: responses required, optional, none?

but: reliable transport – we know the other side will get the message (eventually)

but: the other side state changes after it ACKs

but: speed == amount of work done != messages ACKed

ACK: responses required, optional, none?

- avoid duplication of information (TCP has ACKs)
- require responses only if they carry important info
- add optional ACKs for debugging?

Current decision points (check)

Initiation: depends on message exchange

ACK: only when responses carry info (and for debugging?)

ACKs: $\Longleftarrow$

Granularity:

Copy:

Priority:

ACKs: multiple responses to a request

- multiple responses complicate protocol

but: dispatcher should drain buffers ASAP (large chunks);
callout server should drain buffers ASAP (small chunks)

- multiple data responses are unavoidable for performance reasons

Current decision points (check)

Initiation: depends on message exchange

ACK: only when responses carry info (and for debugging?)

ACKs: when draining buffers

Granularity: $\Leftarrow\!=$

Copy:

Priority:

Granularity: addressable data parts

- "entire message" is simple but inefficient
- "sequential bytes" do not let us skip
- "sequential bytes with gaps" assume serialized application
- "arbitrary bytes" is flexible but may be inefficient

Which one is the best for OPES?

Granularity: addressable data parts

- "entire message" is simple but inefficient
- "sequential bytes" do not let us skip
- "sequential bytes with gaps" assume serialized application
- "arbitrary bytes" is flexible but may be inefficient
- we support the most flexible scheme?
- implementations use application-specific scheme?

Current decision points (check)

Initiation: depends on message exchange

ACK: only when responses carry info (and for debugging?)

ACKs: when draining buffers

Granularity: support arbitrary? use appropriate

Copy: $\Leftarrow\!=$

Priority:

Copy or move data to the other side?

- "move" is simpler and uses less storage on dispatcher

but: "copy" allows callout server to get out of the loop
(which is probably a common need!)

but: dispatcher may copy anyway, for non-OCP reasons
(caching or smooth recovery from OPES failure)

- make copying an optional dispatcher-driven optimization?

- require callout servers to report copying support?

Current decision points (check)

Initiation: depends on message exchange

ACK: only when responses carry info (and for debugging?)

ACKs: when draining buffers

Granularity: support arbitrary? use appropriate

Copy: optional, servers must declare support

Priority: $\Leftarrow\!=$

Can OPES messages be given a handling priority?

- priority handling is not required (only an optimization)

but: fast abort saves resources and
helps cope with DoS attacks

but: QoS is a popular selling point

but: does not complicate protocol specs by much?

- make priority handling an optional optimization?

- do not require support declarations??

Current decision points (check)

Initiation: depends on message exchange

ACK: only when responses carry info (and for debugging?)

ACKs: when draining buffers

Granularity: support arbitrary? use appropriate

Copy: optional, servers must declare support

Priority: optional

OPES Working Group

Callout Protocol Predraft

Why now?

- OCP has too many related design options
- hard to see the big picture when choosing an option
- need a framework to evaluate suggestions
- want to design the "best" protocol
  to compare with existing ones and their NG versions

Why pre-draft?

- OCP has to cover many aspects
- we concentrate on just a few
- convert to ID when coverage is nearly complete?

Key Ideas

- build general message adaptation framework now;
  application agnostic functional layer;
  provide specific bindings and encodings when needed

- pipeline – to scale with message sizes

- relaxed message exchange requirements – to scale with the number of applications and adaptation kinds

- isolate dispatcher from callout servers – to scale with the number of implementations and their needs

- simple and consistent design (duh!)

Major OCP Objects

draft-ietf-opes-protocol-reqs-03.txt:

- callout message (unit or atom of communication)
- callout transaction (processing of a single app. message)
- callout instruction* (a message outside of xaction flow)
- callout connection (logical abstraction) to maintain state of a group of transactions)
- callout agent (OPES dispatcher or callout server)

application specification (e.g., *RFC 2616*):

- application transaction (often vague)
- application message, message part, or stream!

Callout Message

- communication atom or unit
- single source (dispatcher or callout server)
- single destination (callout server or dispatcher)
- has name (e.g., "i-am-here")
- may have attributes (e.g., "xid" or OPES transaction ID)

Callout Transaction

- sequence of callout messages and associated state;
  mostly data exchange
- each side maintains associated transaction state for the life of a transaction
- initiated by OPES dispatcher
- can be terminated by either side
- loosely associated with application transaction
- has an ID,
  unique across all cc transactions from one dispatcher
- may have a priority [?]

Callout Instruction

- command or request:
  "abort transaction X"
  "do you make use of data copying feature?"
- information or response:
  "I am still alive, working on message M"
  "I use data copying feature when possible"
- may appear at any place in the message stream
- consists of exactly one message
- sent by either side (by default)
- may affect the state of OPES agent, connection, or transaction

Callout Connection

- caries callout transactions and/or instructions
- transactions may be multiplexed within a connection [?]
  but may not span multiple connections [?]
- instructions may appear at any time
- initiated by OPES dispatcher, closed by either end,
  kept open by default
- each side maintains associated connection state;
  used for maintaining common transaction properties [?]
- may have a priority [?]
- possibly unrelated to application connections, if any

Callout Agent

- OPES dispatcher or callout server
  (a connection "side" or "end")
- maintains state common to all callout connections
- may maintain expected state of agents on the other end
- has an ID,
  unique across all agents it may communicate with [?]

Common message properties

- xid, amid, source, destinations, services
- data size, data offset [?]
- sizep (application message size prediction, bytes)
- modp (modification prediction, 0-100)
- error (all related information may have been wrong)

Transaction messages from dispatcher

- xaction-start xid services ...
- app-message-start xid amid src dests kind [copied] ...
- app-message-end xid amid [error] reason ...
- xaction-end xid [error] reason

Transaction messages from dispatcher

- data-have xid amid offset size [copied]
- data-pause xid amid
- data-end xid amid [error] reason
- data-ack xid amid offset size

Transaction messages from callout server

- app-message-start xid amid src dests sizep modp
- app-message-end xid amid [error] reason
- xaction-end xid [error] reason

Transaction messages from callout server

- data-need xid amid offset size
- data-have xid amid offset size sizep modp
- data-as-is xid amid offset size
- data-end xid amid [error] reason
- data-ack xid amid offset size [wont-need]
- data-pause xid amid

Callout instructions

keep-alive: i-am-here [xid [amid]]
are-you-there [xid [amid]]

abort: xaction-end xid amid error reason

negotiations: do-you-support feature-id question-id
i-do-support feature-id ... [question-id]
i-do-not-support feature-id [question-id]

Connection management messages

- connection-open dispatcher-id [priority]
- connection-priority priority
- connection-services service-ids ...
- connection-close

Also see keep-alive and feature negotiation instructions.