Saratoga: A Scalable File Transfer Protocol

draft-wood-tsvwg-saratoga-09

http://saratoga.sf.net/

Lloyd Wood
Centre for Communication Systems Research, University of Surrey
l.wood@surrey.ac.uk

Charles Smith
Commonwealth Scientific and Industrial Research Organisation (CSIRO)
charles.smith@csiro.au

Wesley M. Eddy
MTI Systems
wes@mti-systems.com

Will Ivancic
NASA Glenn Research Center
william.d.ivancic@nasa.gov

Chris Jackson
Surrey Satellite Technology Ltd (SSTL)
C.Jackson@sstl.co.uk

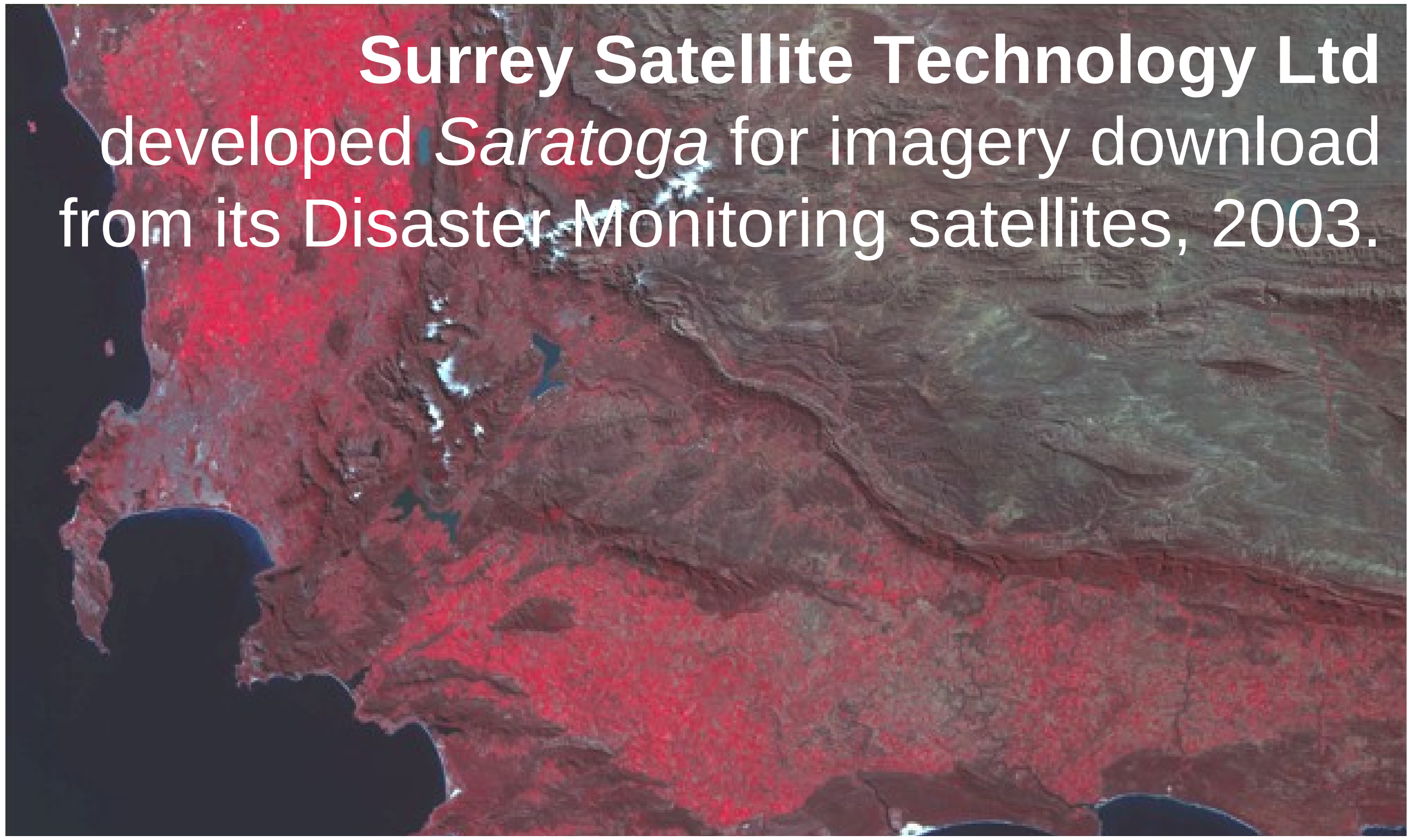
Private sensor networks

- Must deliver sensor data – very quickly.
- Want to use Internet technologies – cheap, reliable, robust.
- Want more speed than TCP can offer.
- Congestion is not a problem; private single-owner managed network with scheduled traffic, single flow per link with no competition. **This is not the shared public Internet!**
- Sensor capabilities are ever-increasing (side-effects of Moore’s law). Need to scale for ever-growing data sizes.
- Support for streaming and simultaneous delivery to multiple receivers is also useful.
- ***Saratoga* protocol designed to meet these needs.**

Saratoga in brief

- *Saratoga* is a high-speed, UDP-based, peer-to-peer reliable transport protocol, providing error-free guaranteed delivery of files, or streaming of data.
- Send data packets out as fast as you can. No specified congestion control is required, since data is usually only going one hop over a private link, or across high-speed, low-congestion private networks.
- Some implementations have a rate-limiting option for restricted downstream links where line rate may not match downstream radio link.
- No specified timers means no forced timeouts, so *Saratoga* can be used with links with very long propagation delays.
- *Saratoga* is an excellent choice for highly asymmetric forward/back path capacities.

Saratoga's development

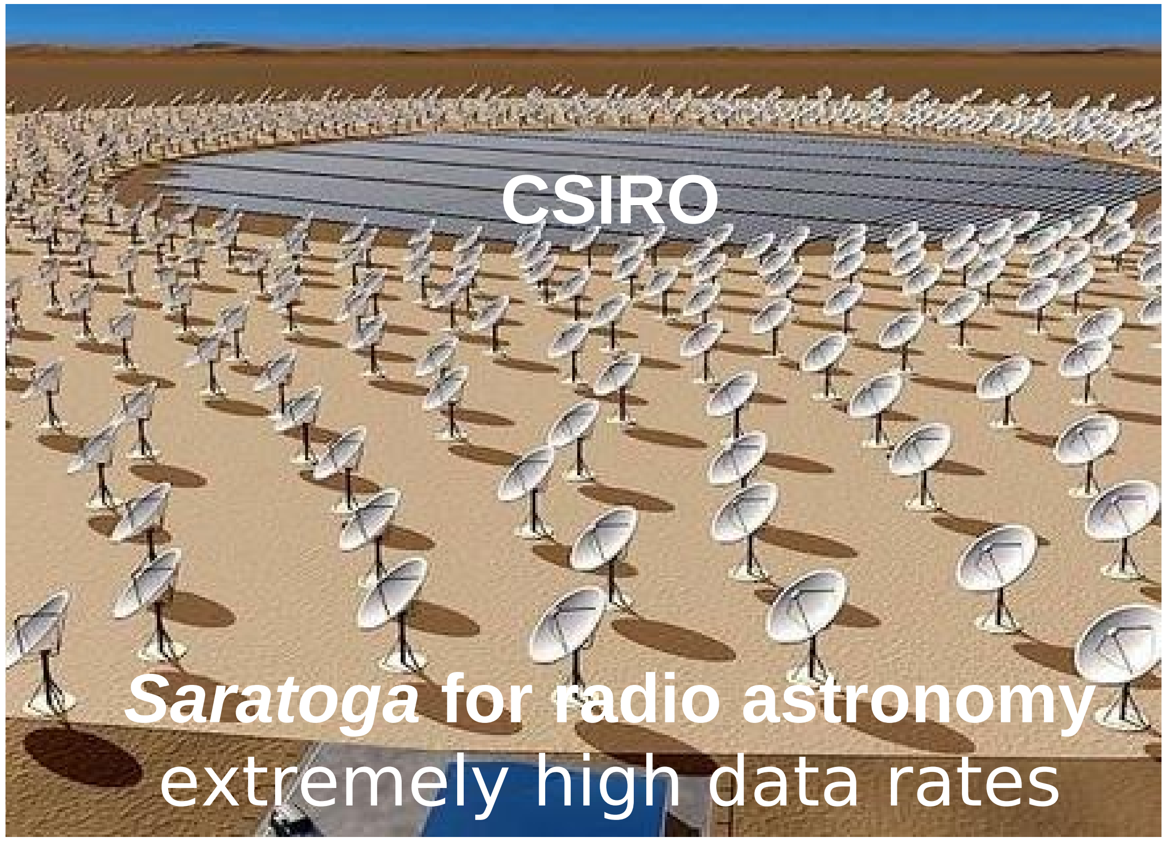


Surrey Satellite Technology Ltd developed *Saratoga* for imagery download from its Disaster Monitoring satellites, 2003.

NASA Glenn Research Center
Saratoga for sensors on UAVs

Saratoga redesigned, specified to the Internet Engineering Task Force, 2007.

NASA Glenn uses *Saratoga* to test DTN and Interplanetary Internet on UK-DMC, 2008.



CSIRO

***Saratoga* for radio astronomy:**
extremely high data rates

Multiple *Saratoga* implementations in progress with interoperability testing.

Research led to new users

- SSTL remote-sensing images grew to cross 4GiB file size, needing >32-bit pointers.
- How to design a *scalable* file transfer protocol able to handle any size file, without requiring separate incompatible implementations for big files?
- Solved this problem with 16/32/64/128-bit pointers and advertising capabilities.

not needed - yet!

- 64-bit pointers support files up to 16 exabytes in size.
- 128-bit pointers can support files up to 256 exa-exabytes.
- Support for scalability and streaming introduced new users: **high-speed networking for radio astronomy in Very Long Baseline Interferometers.**

Saratoga operation

Simple sliding window with selective acknowledgments.

- The receiver requests the sender retransmit frames that have not been properly received by sending a STATUS with the list of HOLESTOFILL (a SNACK).
- The receive window only advances when offsets are contiguous. The left edge of the transmit window does not advance until the holes have been acknowledged by a HOLESTOFILL frame with an advanced offset.
- The UDP checksum is used per packet to cover both the header and payload. It is consistent, but not strong (one’s complement), and does not provide end-to-end guarantees for payloads sent using multiple packets.
- An optional end-to-end checksum over an entire file being transferred, using one of CRC32/MD5/SHA-1, increases confidence that a reliable copy has been made, and that fragments have been reassembled correctly.

Features of *Saratoga* version 1

Major features

- Scalable to handle large files. 16-bit descriptors for efficiency with small files <64K. 128-bit descriptors can cope with *huge* files. 32- and 64-bit descriptors are most useful.
- Streaming of data is supported. This allows *Saratoga* to be used for real-time delivery outside the file-based store-and-forward paradigm.

Minor features

- Supports link-local multicast to advertise presence, discover peers and for delivery to multiple receivers simultaneously for *e.g.* file or code image updates. (Will outperform TFTP trivial file transfer.)
- Optional UDP-Lite use for tolerating errors in payloads and minimizing checksum computation overhead. The UDP-Lite checksum covers a minimum of IP/UDP-Lite/*Saratoga* headers. The header content is always checked so that the information *about* the data is error-free.
- Optional “DTN bundle” delivery as a “bundle convergence layer”. Shown with tests from the UK-DMC satellite.

What *Saratoga* doesn't do

- There is no MTU discovery mechanism, so you have to know the maximum packet size your network can transmit at. *i.e.* dictated by the frame size. This is fine for your own private network, but would be troublesome if used across the Internet.
- ***Saratoga* does not include “slow-start” or congestion control.** That is considered poor, unsociable behavior on the Internet. *Saratoga* just blasts away on a link with no regard for other flows – which is the exact behavior that makes it desirable in private networks and these environments!
  - Simulations have shown that it is possible to implement congestion control mechanisms in *Saratoga* if desired – see University of Oklahoma paper at 2011 IEEE Aerospace Conference, describing *Sender-Based TCP-Friendly Rate Control.*
    - Saratoga’s timestamp option can be used to implement such *closed-loop* mechanisms without datagram changes.
    - Simple *open-loop* rate-limiting output to *X* Mbps can also allow Saratoga to coexist with other traffic.

Saratoga version 1 implementations

C (Charles Smith under contract to Cisco Systems)

- Implementation licensed to CSIRO by Cisco.
- Built for *speed* (raw I/O).

Streaming to be implemented in FPGA. File transfer may also go to FPGA.

C (Surrey Satellite Technology Limited – SSTL)

- Implemented for high-speed image transfers from Low Earth Orbiting (LEO) remote-sensing satellites over highly asymmetric links.

PERL (NASA Glenn Research Center)

- Also supports sequential file transfer and rate limiting.

C++ (NASA Glenn Research Center)

- Discovery, multiplexed file transfer, hooks for bundling and streaming and rate-limiting to be implemented.

Wireshark Dissector (Charles Smith)

- Already available from Sourceforge.

We hope to make some of these implementations available to the public.

Our approach to the IETF

- We plan to take *Saratoga* version 1 through as an individual submission for Experimental status.
  - Rationale: Keep the current implementations progressing and maintain interoperability between them.
  - Keep detailed discussion on version 1 *implementations* to the *saratoga-discussion* list. (To join that list, please contact Lloyd Wood.)
- However, we would like TSVWG to begin reviewing this version 1 specification to provide constructive comments and criticism, as well as to gauge interest in adopting work on version 2 as a Proposed Standard.
- Comments or suggestions?

Saratoga transactions

GET	Get a named file from the peer
GETDIR	Get a directory listing of files from the peer
DELETE	Delete a named file from the peer
PUT	Put a named file or stream data to the peer

Saratoga frame types

BEACON

Sent periodically. Describes the *Saratoga* peer:
Identity (e.g. EID)
capability/desire to send/receive packets.
max. file descriptor handled: 16/32/64/128-bit.

REQUEST

Asks for a file initiating ‘get’ transaction
get file
get directory listing
delete a file.

METADATA

Sent at start of transaction. Initiates a ‘put’ transaction.
Describes the file, bundle or stream:
set identity for transaction
file name/details, including size.
set descriptor size offsets to be used for this transaction
(16/32/64/128-bit pointer sizes.)

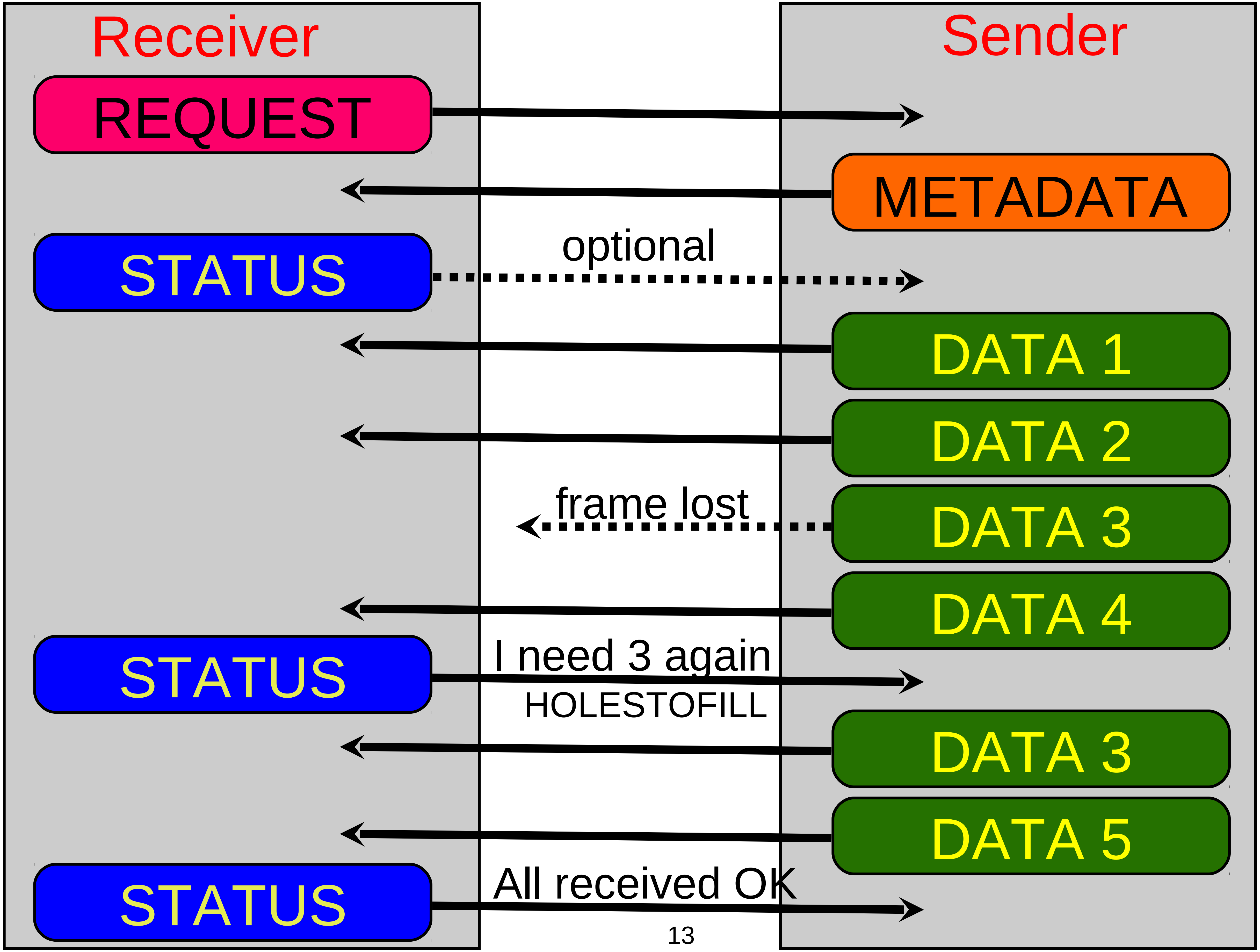
DATA

Actual Data.
Uses descriptor of chosen size to indicate offset for data segment in the file/bundle or stream.
May request an ‘ack’ (send me a holestofill).

STATUS

Missing Data Offsets / Error & Status Messages
Selective negative ack (‘snack’) HOLESTOFILL data.
Set left window edge for successful transfer so far
List of offsets and lengths indicate missing ‘holes’ in data.

Transaction GET or GETDIR



Receiver

Sender

REQUEST

METADATA

STATUS

optional

DATA 1

DATA 2

frame lost

DATA 3

DATA 4

STATUS

I need 3 again

HOLESTOFILL

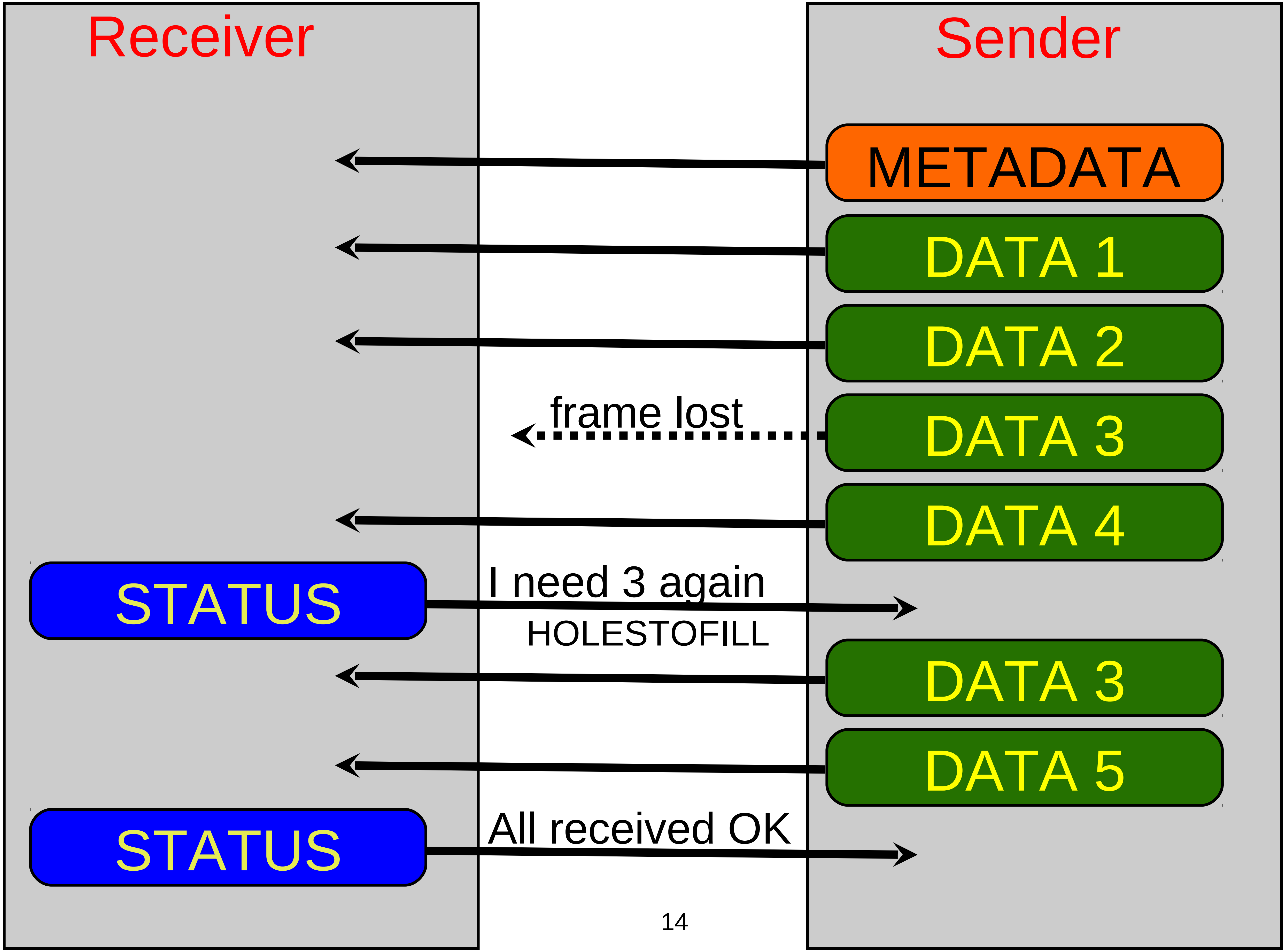
DATA 3

DATA 5

STATUS

All received OK

Transaction “blind direct” PUT



Receiver
Sender
METADATA
DATA 1
DATA 2
frame lost
DATA 3
DATA 4
STATUS
I need 3 again
HOLESTOFILL
DATA 3
DATA 5
STATUS
All received OK