MIKE BISHOP
ALAN FRINDELL
BUCK KRASIC
ROBERTO PEON
MARTIN THOMSON
DMITRI TIKHONOV

HEADER COMPRESSION DESIGN TEAM

LATENCY IS THE ENEMY
(AND POOR COMPRESSION IS LATENCY)

- Head-of-line blocking
  - Reordering
    - Particularly from loss, but also network and even internal
    - Always impacts the current stream, can impact other streams
  - Data loss
    - Packet drops in combination with RST_STREAM (i.e. never retransmitted)
- Bandwidth limitations
  - Fit more requests into allowed bytes

THE MISSION

Static HPACK suffers from poor compression efficiency

Serialized HPACK suffers from head-of-line blocking

THE MISSION

Static HPACK suffers from poor compression efficiency

Serialized HPACK suffers from head-of-line blocking

Efficiency *and* Blocking Avoidance?

OPERATING CONDITIONS

- Reordering is common
  - Network reordering varies widely across networks
  - Loss and retransmission is fundamentally a reordering event
  - Multi-threaded implementations may induce reordering internally
- Many connections experience no loss
  - Not so many that we can discount this
  - Not so few that we should penalize the majority for the minority’s crummy link
- Request cancellations occur with some frequency
  - Only ~0.8% of **requests** are reset (Facebook)
  - ~51% of **connections** experience at least one reset (Akamai)

HOW TO HANDLE REORDERING: BLOCKING

FULL ORDERING

- Risks false sharing in head-of-line blocking
  - Single packet lost from this stream blocks headers on all streams
- Worst possible HOLB rates

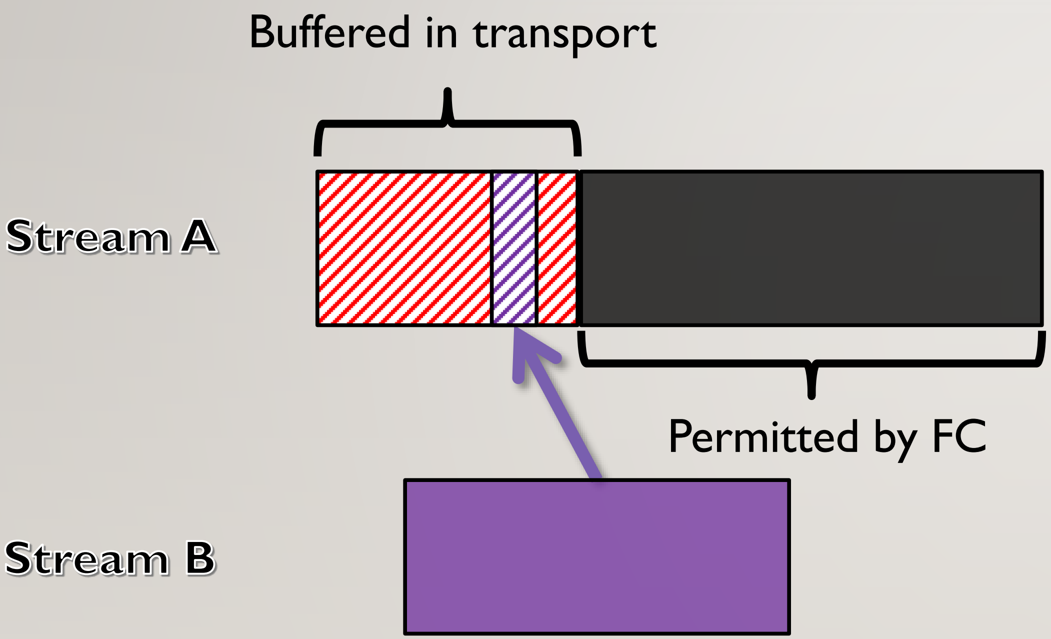
OPTIMISTIC CONCURRENCY

- Assumes state has arrived
  - Block only if necessary state is missing
- Uses flow control to provide back-pressure and control memory consumption
- Risks deadlocks

NEVER RISK BLOCKING

- Robustness
  - Avoids risks of deadlock, memory consumption, etc.
- Efficiency suffers noticeably
  - Must add headers to table at least 1 RTT in advance of using them, or else send them multiple times during first RTT of use

HOW TO DEADLOCK



- Interpretation of Stream B depends on data from Stream A
- Flow control prevents data on Stream A from being sent
- Lack of progress on Stream B prevents new flow control credit from being issued to Stream A

HOW TO NOT DEADLOCK

Don't Do That!

- **Problem:** Can all application protocols avoid this all the time?
- **Problem:** Really hurts compression performance

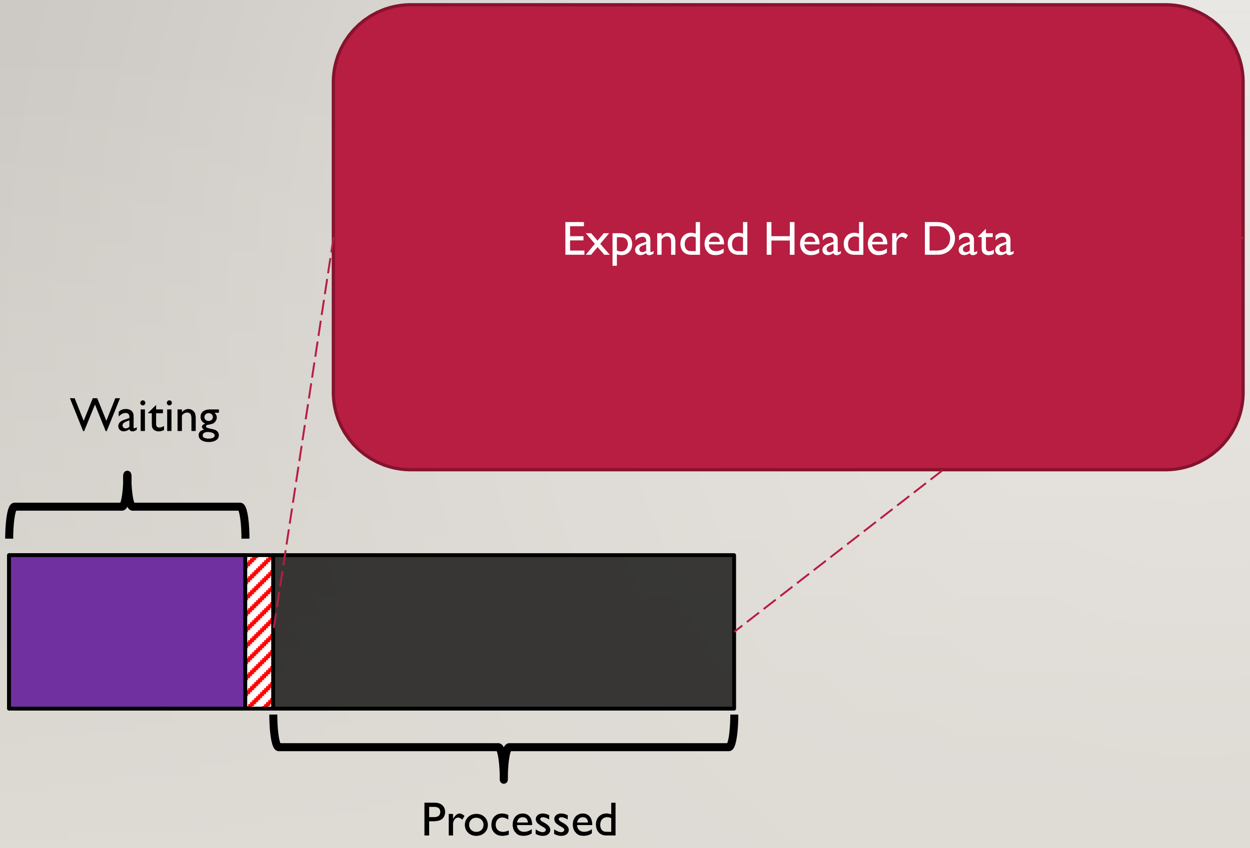
Prioritization Between Streams

- Ensure Stream A makes progress with any new flow control credit that becomes available
- **Problem:** Priorities are currently:
  - Purely advisory => optional
  - Internal to the transport implementation's design

Consume Flow Control Sooner

- Flow control consumed on write completion, not on transmission
- Application responsible to make sure data written to A before beginning write to B
- **Problem:** Application-level retransmits

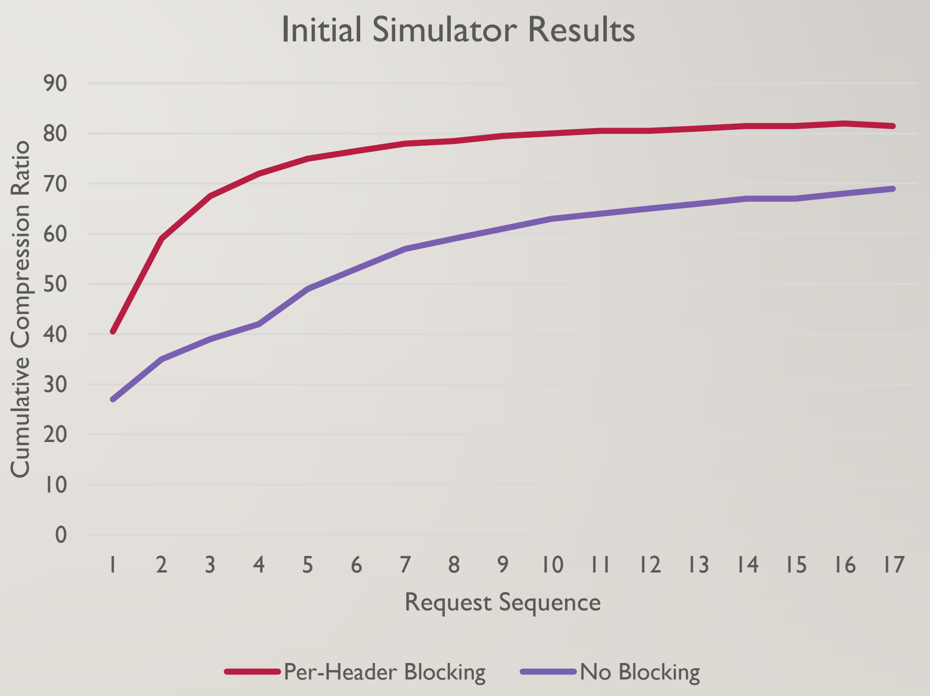
LIMITING MEMORY CONSUMPTION



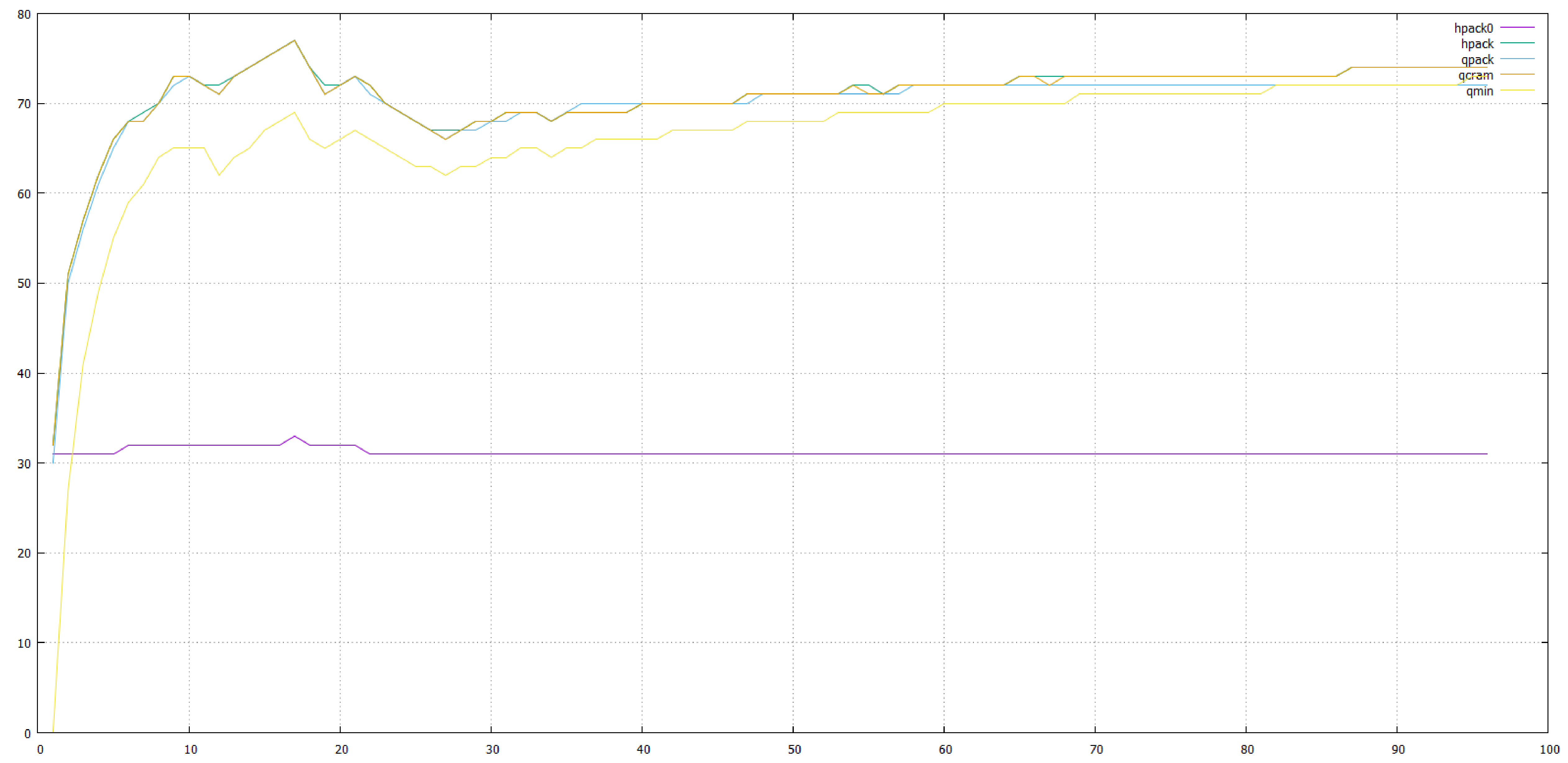
- Discovering a blocking reference mid-frame means you already have uncompressed data in memory
- Suggestion: Don't begin reading a frame until you have all necessary state to finish
  - Uses flow control for back pressure
  - Requires frame preface describing encoder state
- Separate from blocking on missing data

SIMULATOR RESULTS

- Allowing blocking means carefully balancing ways to avoid deadlocks
- Noticeable compression gains early in the connection
  - No simulator yet for per-set blocking
- No data yet on exactly how this translates to latency



SIMULATOR RESULTS: LONGER SESSION



HOW TO BUILD CONTROL STREAMS

MANY CONTROL STREAMS

- Mitigates the impact of loss between unrelated entries
- Requires transport features to guarantee no deadlocks

SINGLE CONTROL STREAM

- Simplifies deadlock avoidance
- Efficiency suffers in the presence of loss

MINIMIZE THE CONTROL STREAM

- Simplifies common case
- After aborted stream, re-writes critical data on control stream

HOW TO TRACK DATA

DATA PER HEADER

- Each header is individually added, referenced, and deleted
- DT has largely eliminated due to memory/CPU overhead

CHECKPOINTS

- Groups of header entries
- Track which/how many checkpoints reference entry
- When all referencing checkpoints are gone, header is removed

ROTATING WINDOW

- Headers added in sequence (HPACK-style)
- When table size reaches limit, old entries roll off

WHERE PROPOSALS LAND

	Data per header	Checkpoints	Rotating window
Full ordering			HPACK ⟶
Optimistic concurrency			
Never blocking			

WHERE PROPOSALS LAND

	Data per header	Checkpoints	Rotating window
Full ordering			HPACK →
Optimistic concurrency			
Never blocking			

WHERE PROPOSALS LAND

	Data per header	Checkpoints	Rotating window
Optimistic concurrency	QPACK		
Never blocking			QCRAM

WHERE PROPOSALS LAND

	Data per header	Checkpoints	Rotating window
Optimistic concurrency	QPACK		QCRAM
Never blocking		QMIN	

WHERE PROPOSALS LAND

	Data per header	Checkpoints	Rotating window
Optimistic concurrency	QPACK		QCRAM
Never blocking		QMIN	

WHERE PROPOSALS LAND

	Checkpoints	Rotating window
Optimistic concurrency	QPACK	QCRAM
Never blocking	QMIN	

ACHILLES HEELS

HPACK →

- Requires full ordering

QCRAM ⬐

- Risks deadlock without major changes to how HTTP cancels requests

QMIN ⇢

- Blocking avoidance reduces efficiency when it matters most

QPACK ⤮

- Parallel control streams are complex, of unproven usefulness

MOVING FORWARD

- Need more data to explore latency versus efficiency trade-off
- Simulation/implementation updates in progress
  - Alan implementing QPACK-07
  - Buck implementing QCRAM-03
- Input from working group: Rule anything else in/out?
  - Blocking?
  - Configurable pieces?
  - Delayed reading from transport?