Content- and Cache-aware TCP
("Poor Man's Information-Centric Networking")

2012-06-14

Pasi Sarolahti, Jörg Ott, Karthik Budiger, Arseny Kurnikov
Aalto University

Colin Perkins
University of Glasgow

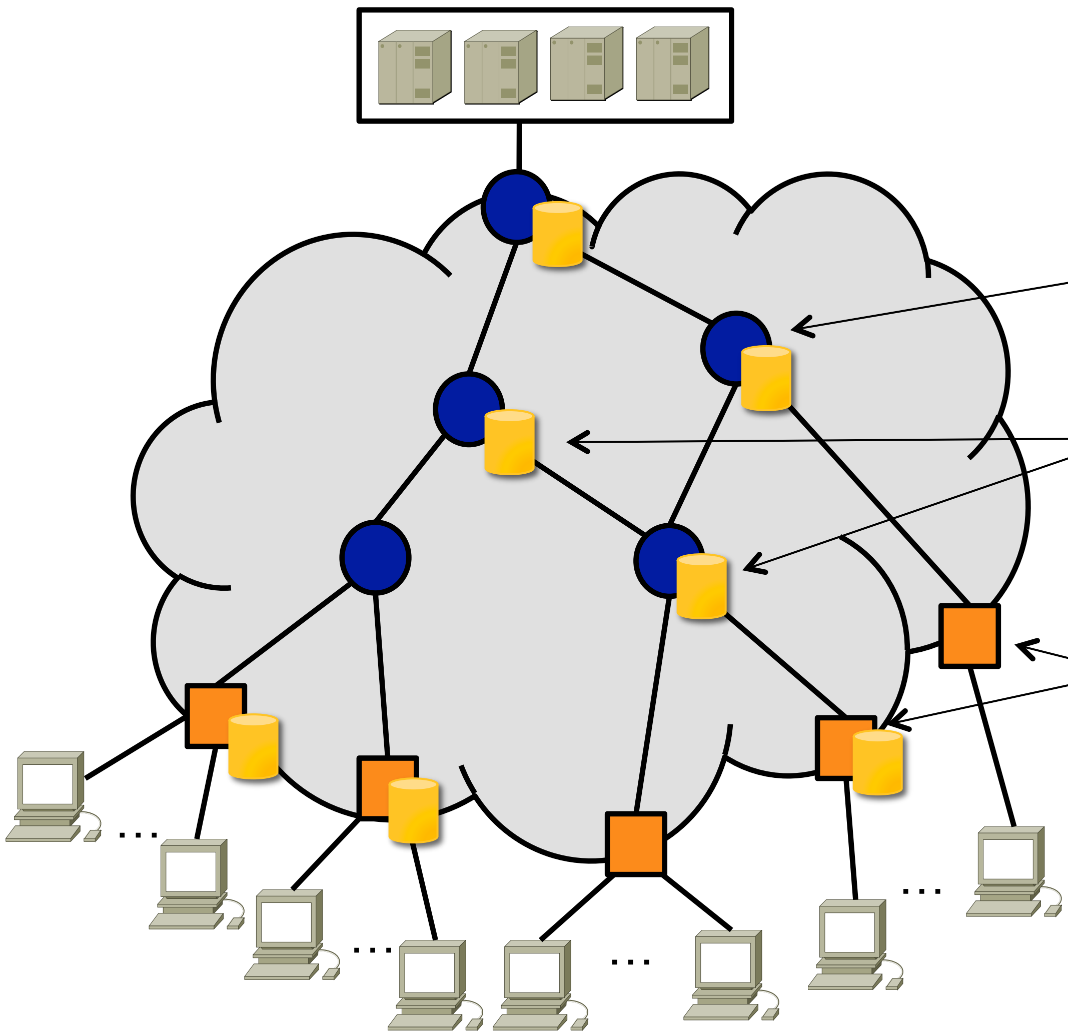
Background

- Scalable content dissemination for non-CDN providers
- Independent of the application protocol
  - Beyond web caching
- Support for
  - Quasi-synchronous fan-out to many receivers (e.g., live streaming)
  - Short-term caching (e.g., flash crowds)
  - (Packet-level retransmissions)
- Mix between redundancy elimination & multicast transport
- Target: deployable content-aware networking

Basic Idea

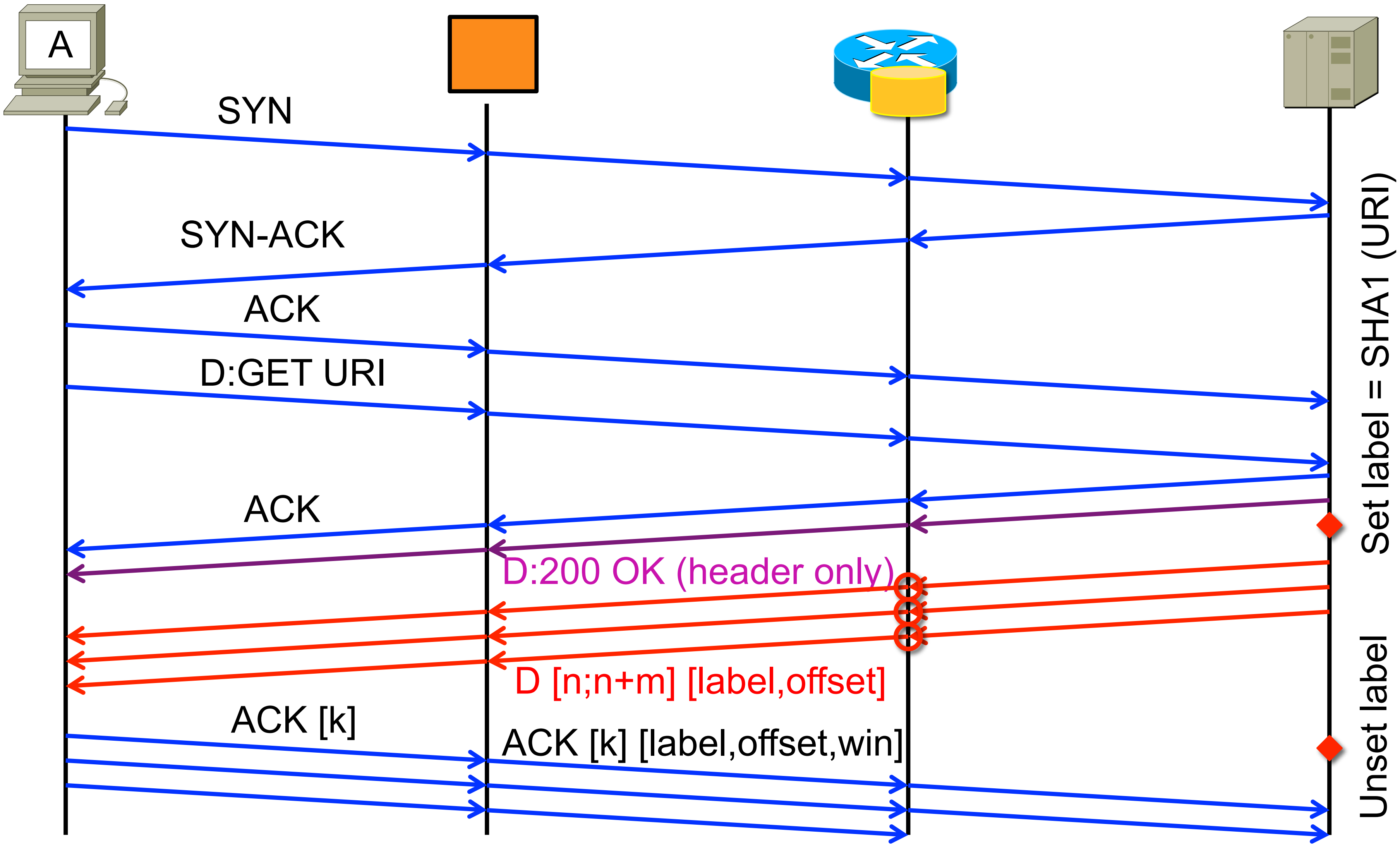
- Operate at the transport layer: TCP
  - Units are sections of a byte stream
  - Carried as TCP segments (but segment boundaries don't matter)
- Label pieces of reusable content at the sender
  - (label, offset) – identifiable independent of their TCP flow
- Store labeled pieces in stateless segment caches
- Re-use these pieces across TCP streams
  - Map (label, offset) to flow-specific TCP sequence numbers
- Controllers maintain state to perform this mapping
  - At an access/edge router or in the receiver

Sample Scenario

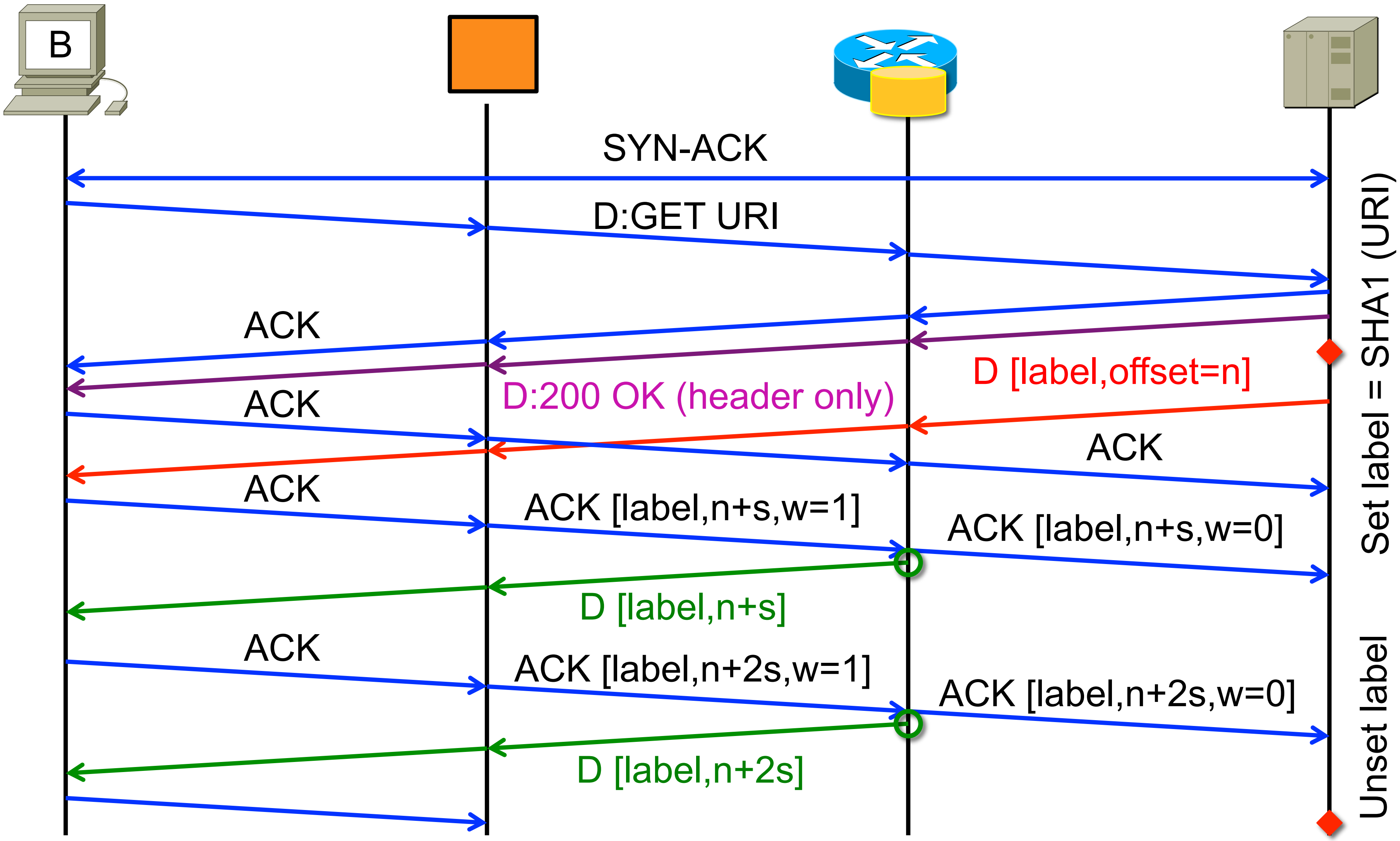


Senders	label content
Routers	work as before
Segment caches	store and retrieve labeled content
Controlling node	map between TCP seq# and content offsets
Receivers	aren't touched (but could be)

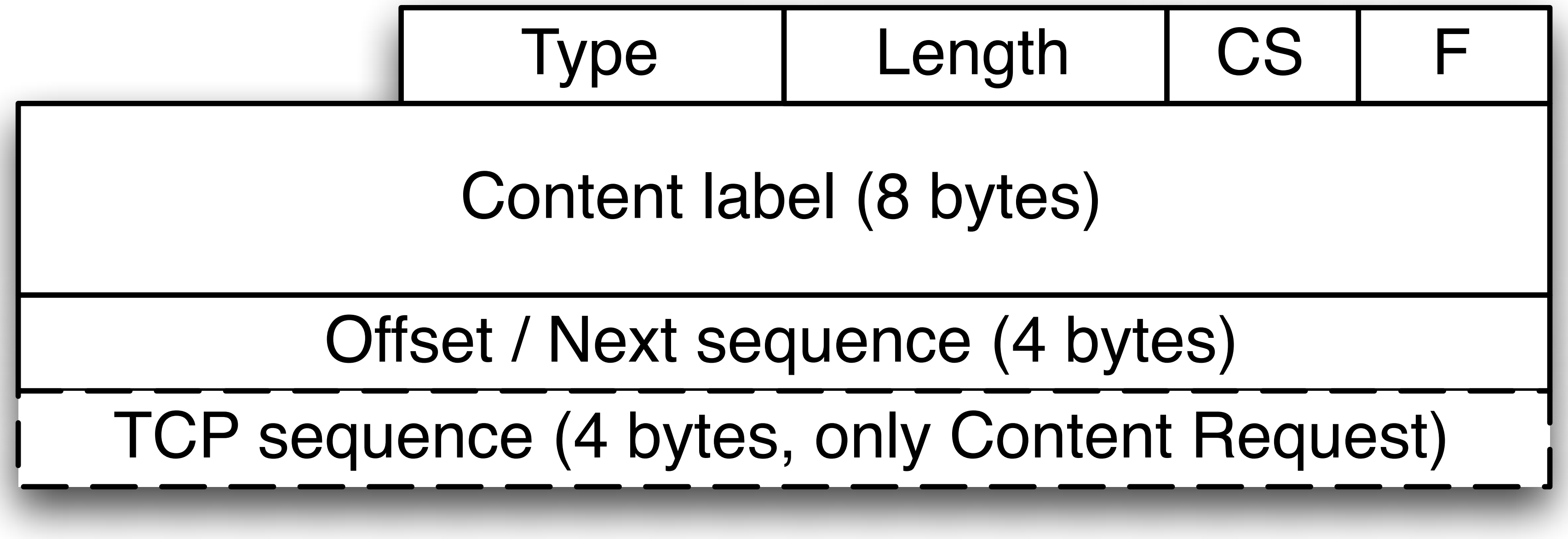
Protocol Phases by Example (1)



Protocol Phases by Example (2)



CA-TCP option



Sender

- Determines content to be cached
  - Assigns labels based upon interaction with receiver
  - SHA1 (URI), BitTorrent chunk id, …
  - Switches between labeled and unlabeled transmission
- Data transmission and ACK processing
  - Sends initial data packets
  - Updates SND.NXT as per ACK offset
  - Runs its congestion control algorithm
    - cwin limited by the can_send field – sends no new data if caches did so
- Tracks receiver interaction
  - Remains aware of requests and their completion
  - Seeds content segments when needed
  - Performs retransmissions
- Operates as regular TCP if there is no controller (no ACK labels)

Sender API

- BSD sockets

- Current implementation: an extension to `send()`
  - Defines the label and offset to be used starting with the next `write()` or `send()` calls.

```
struct catcp_fields {
  uint8_t  content_label[8];
  uint32_t offset;        // network byte order
} catcp_cmd;

send (int sd, &catcp_cmd, sizeof (catcp_cmd), 0xff);
```

Segment cache

- Stores labeled content segments
  - Implements admission and replacement policy

- Matches incoming ACKs with (label, offset) pairs against stored segments p:
  - ACK.can_send > 0
  - ACK.label = p.label
  - ACK.offset ≥ p.offset && ACK.offset < p.offset + p.len

  ➔a cached packet yields fresh data for the resource and does not leave any gap and there is room to send more data

- Create packet towards the receiver flow from the stored one
  - Use the sequence # from the CA-TCP option
  - Update: ACK.can_send and ACK.offset
  - Forward ACK uptream when can_send=0 or no more matches

Controlling node

- Stateful per flow that contains labels
  - Not suitable for the core ➔ edge/access routers or endpoints
- Only acts for flows without a controlling node downstream
- Establishes the binding between flow-specific TCP sequence number and resource offset
- Runs a per-flow congestion control algorithm
  - Simplified version of TCP congestion avoidance)
  - Indicates the # packets per ACK in the can_send field
- Delays ACKs to desynchronize simultaneous flows
  - So that a cache has a chance to receive a packet first from the sender

Receiver

- Operates as usual
- Legacy: ignores CA-TCP options
- CA-TCP: acts as controller for the flow

Features

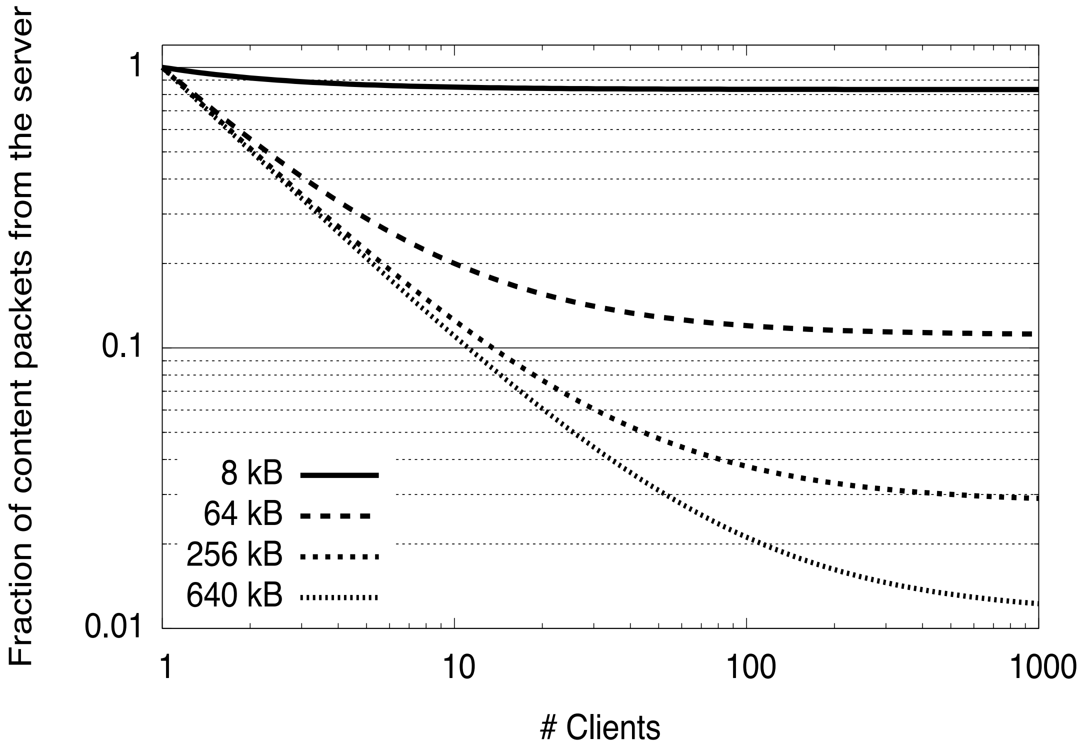
- Supports bidirectional operation
  - Each direction treated independently
  - Caveat: limited TCP option space
    - Won't do bidir with timestamps or SACK due to space limitations
- Works with any application layer protocol that allows a sender to differentiate between cachable and other data
  - Allows any client-server negotiation
  - Server can count requests
- Does not require segment boundary alignment

Implementations

- Linux kernel 2.6.26 and 3.0.9 for the sender side
  - TCP extensions + extended socket API
  - Used for simulations with the ns-3 cradle and for experiments
- Four servers that add CA-TCP content labels
  - highttpd 1.4.18 for web resources
    - Uses 8 bytes of MD5 hashes over of the URI as label
  - BitTorrent extensions to the TCP-based peer-to-peer protocol (PWP)
    - Uses 8 bytes of SHA1 identifiers
  - Simple live streaming server (tcpst)
    - Label, data rate, and stream duration encoded in URI
    - Syncs up clients to a common offset when they join
- Controlling nodes and segment caches
  - Click-based implementation for ns-3 simulation and experiments
  - catcp-bridge (L2 bridge) for the experiments (2400 lines C code)

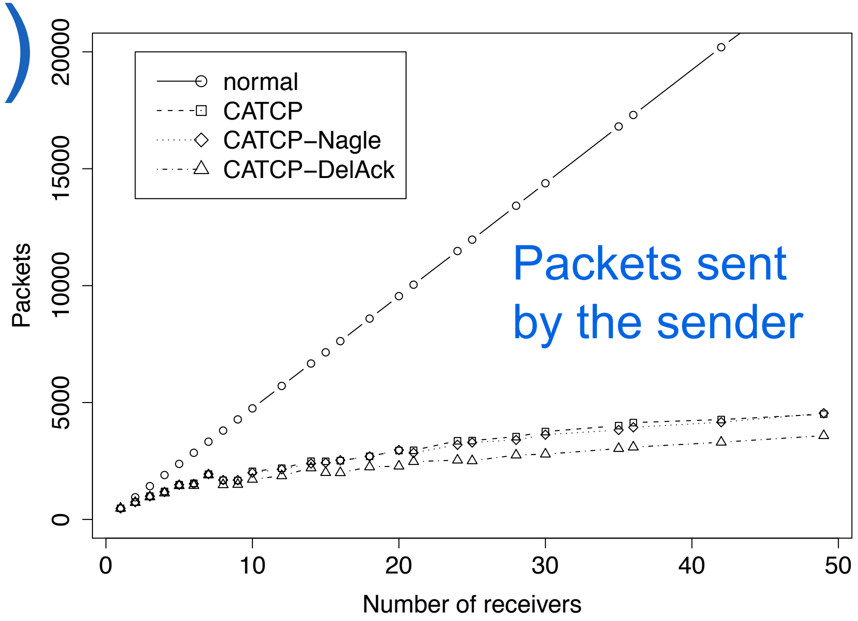
Performance limits

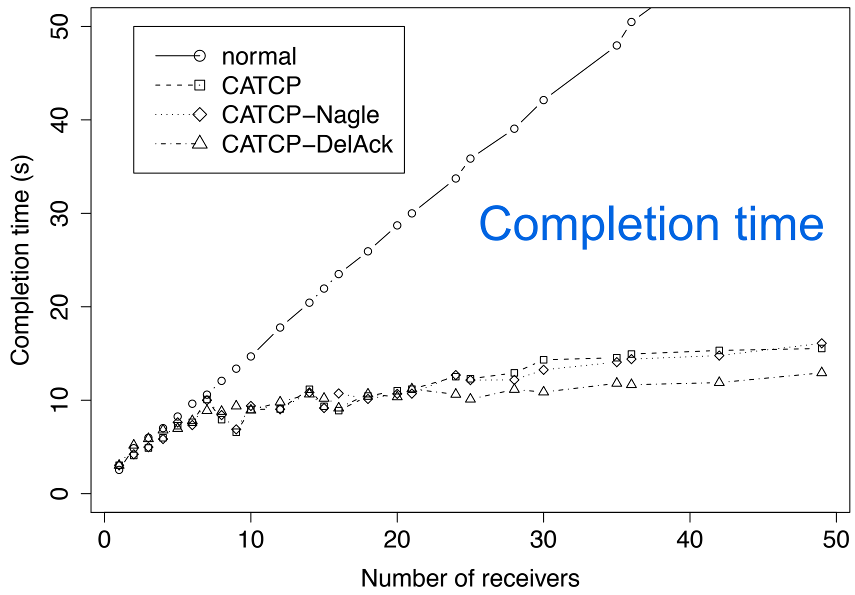
- Minimal number of packets always seen by the sender
  - Control traffic for every connection
    - SYN-ACK + FIN-ACK handshakes
    - Request/response header packets + ACKs
  - Initial cwin data packets
  - Continuous flow of ACKs
- No gain for resources less than 8 KB
  - Don't label them

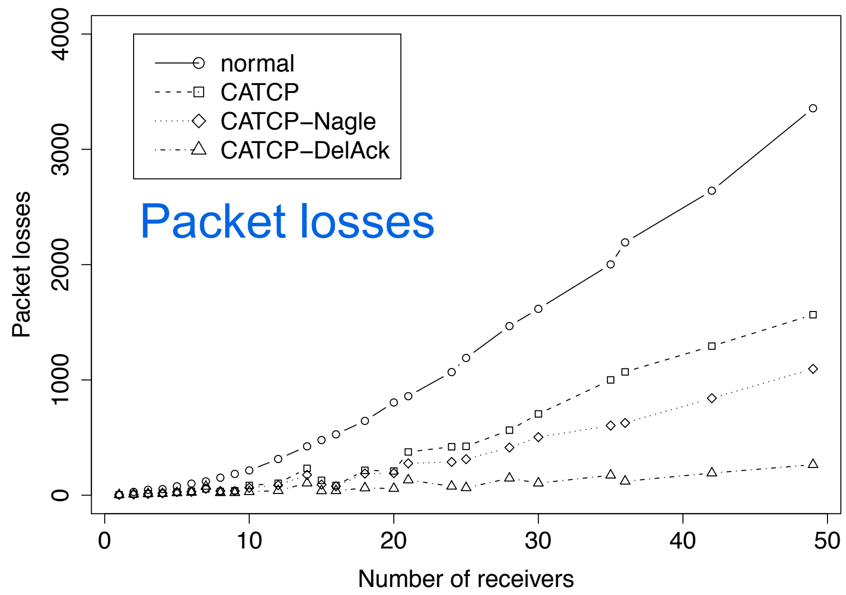


Simulation results (1)

- Setup
  - Single sender
  - 7 network segments
  - up to 7 receivers each (1…49)
  - 625 KB download

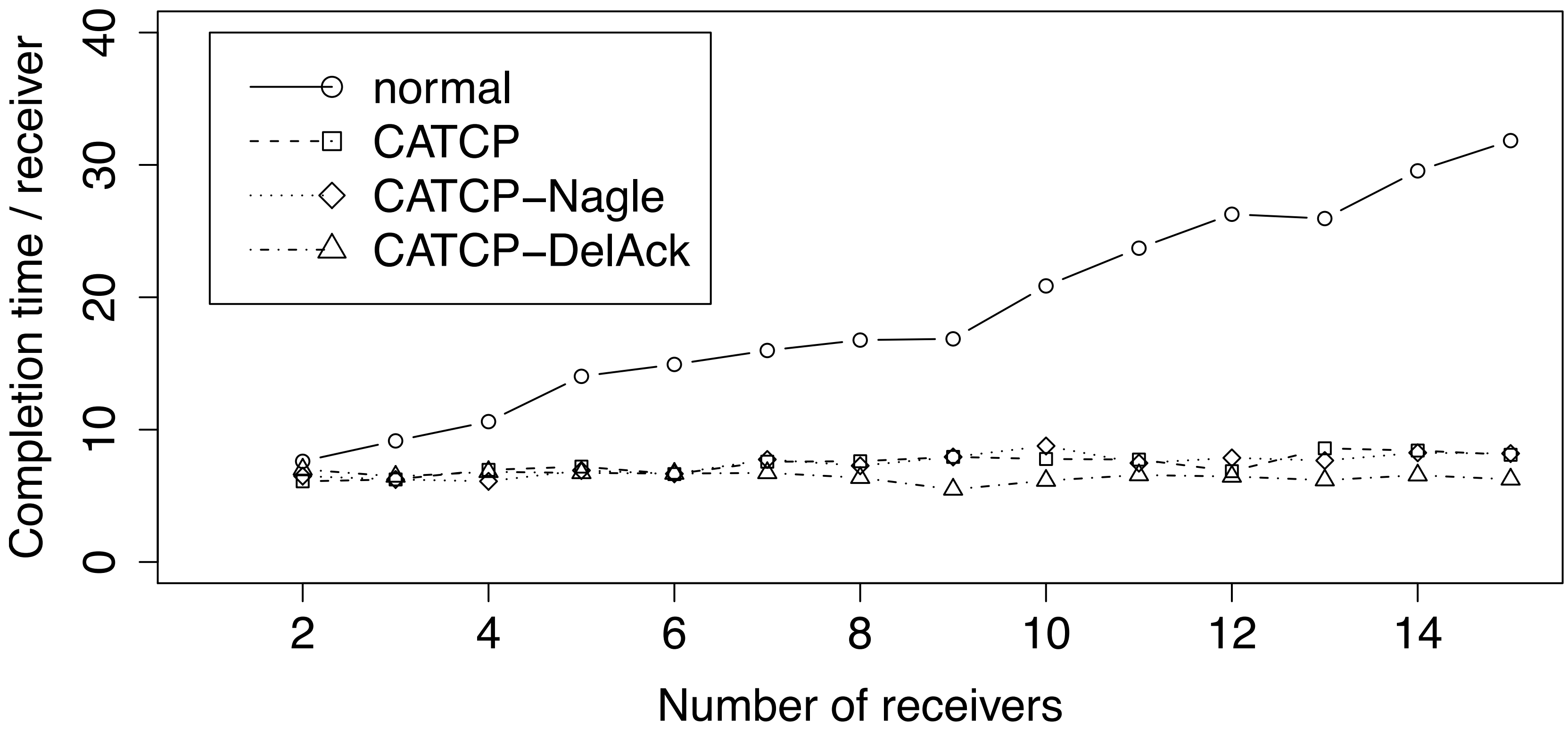






Simulation results (2)

- Evaluation of impact on TCP cross-traffic
  - Three TCP flows share a bottleneck with 1..15 CA-TCP flows



Experiments

- Interop
  - Fixed: MS Windows XP and 7, MacOS 10.[456], Linux
  - Mobile: Linux (Maemo, MeeGo), Android, iOS 4, 5, Symbian S 60

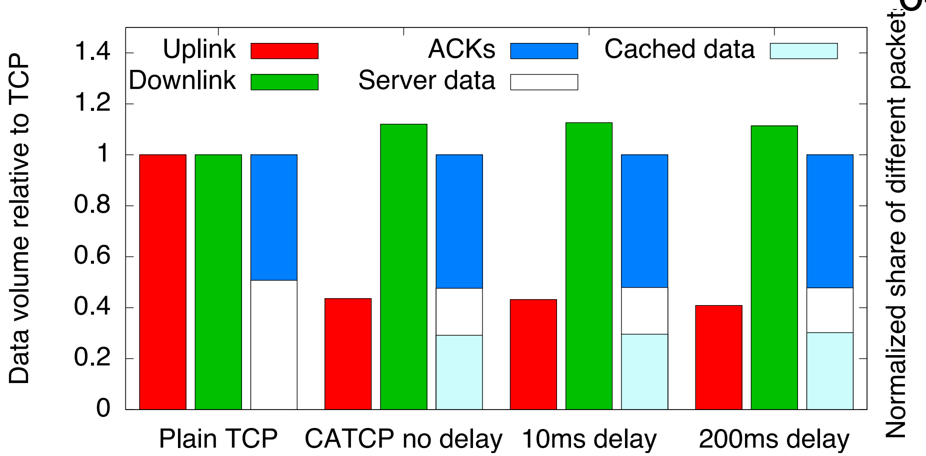
- Lab setup: 4 Linux machines
- Amazon cloud: 4 servers on different continents
  - Ireland, Brazil (Sao Paulo), Singapore, US (Virginia)
- Home server with 24/1 Mbit/s DSL

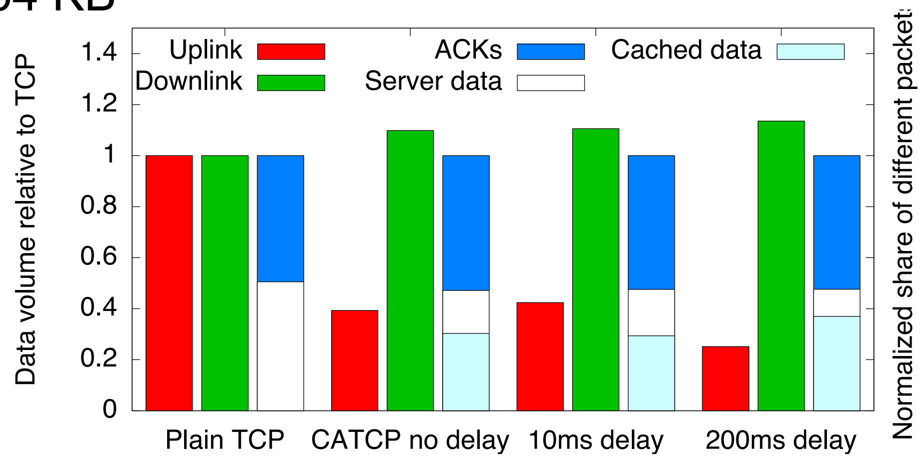
- 1 – 50 receivers, 0 – 1 s intervals, 0 – 200ms ACK delays
  - Web: 64 KB, 256 KB, 1 MB objects
  - Streaming: 100 kbit/s streams

- BitTorrent: 2 – 10 leechers for 64 KB downloads from Amazon

Web experiments (1)

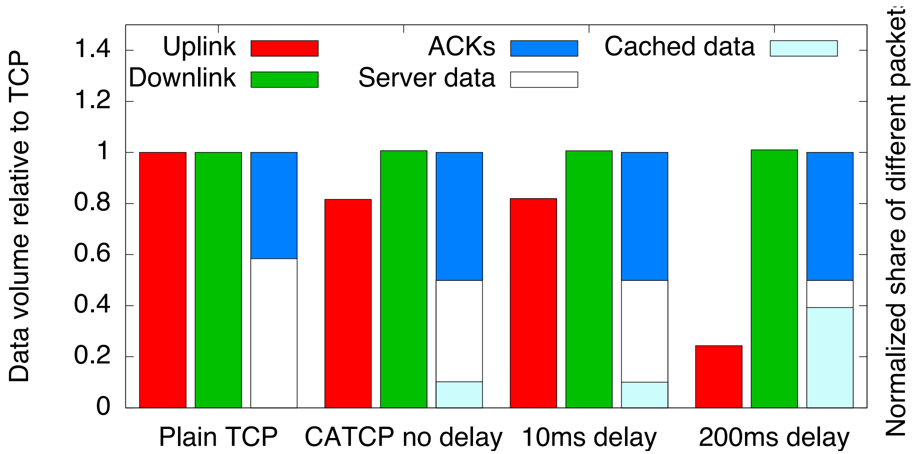
64 KB

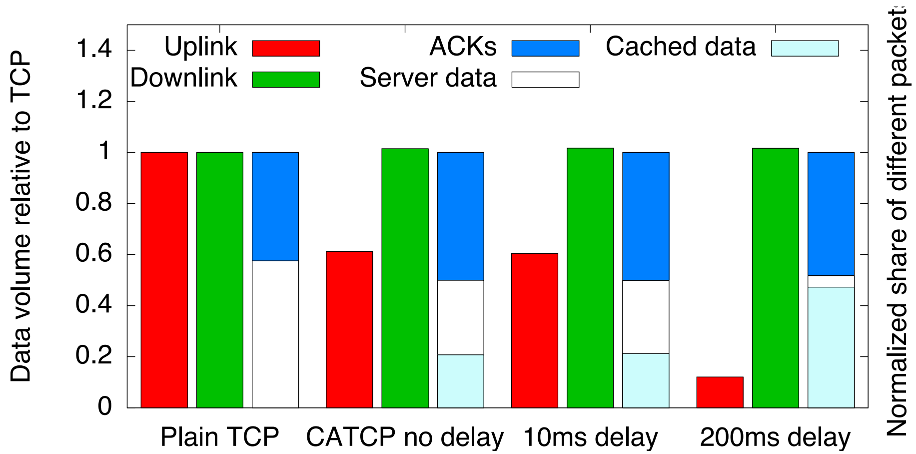




5 receivers

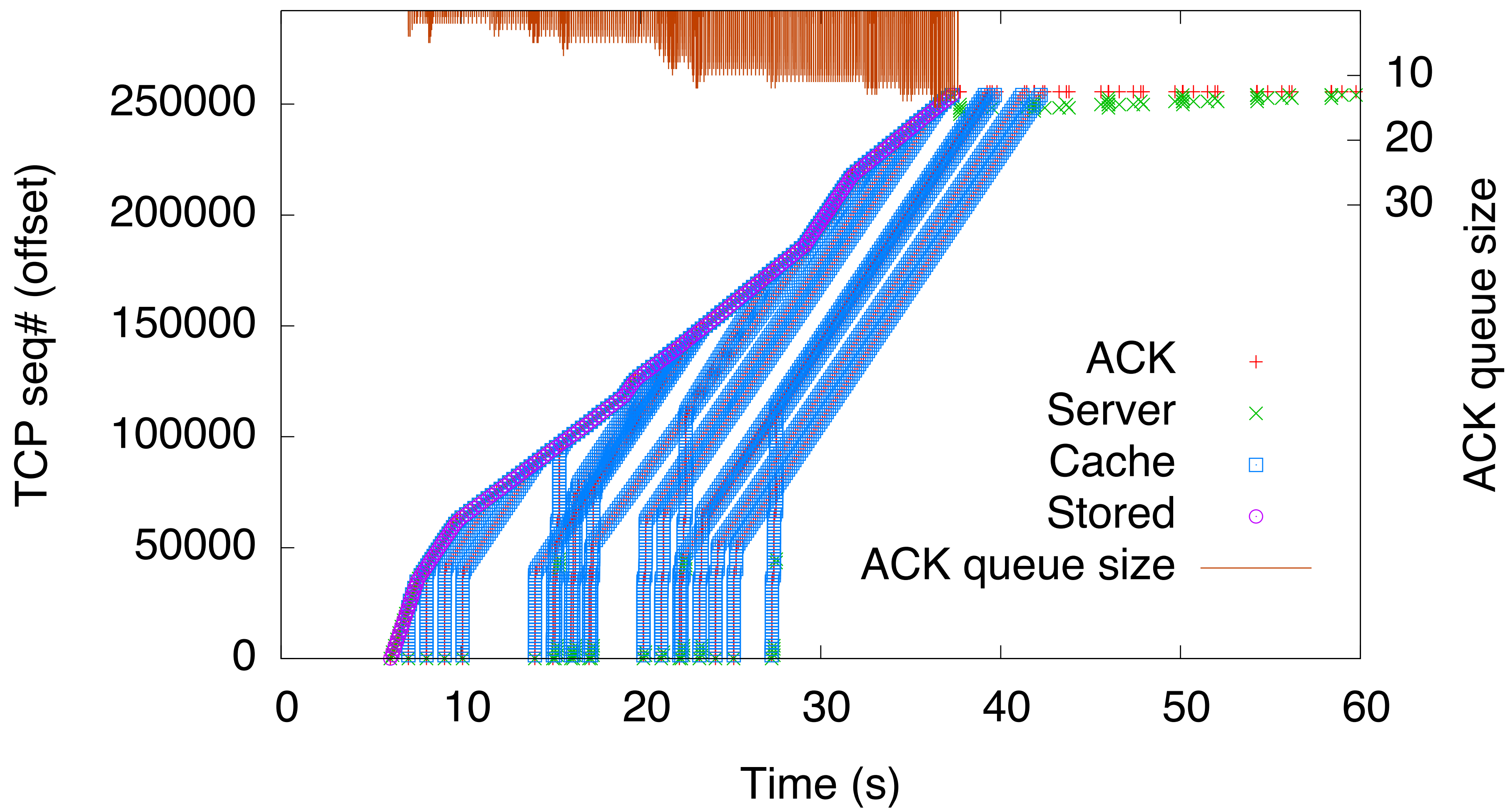
50 receivers



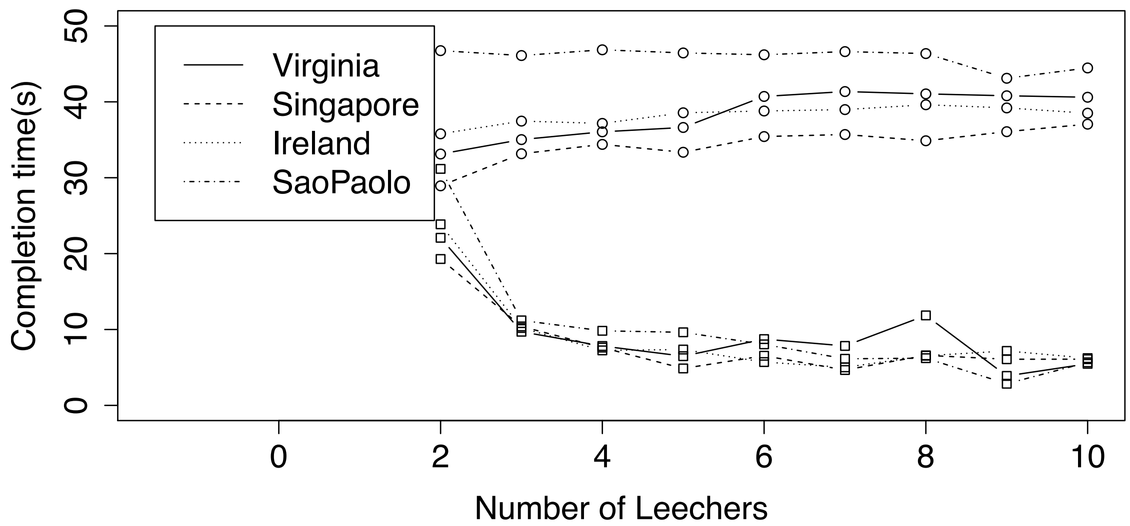


1 MB

Web experiments (2)

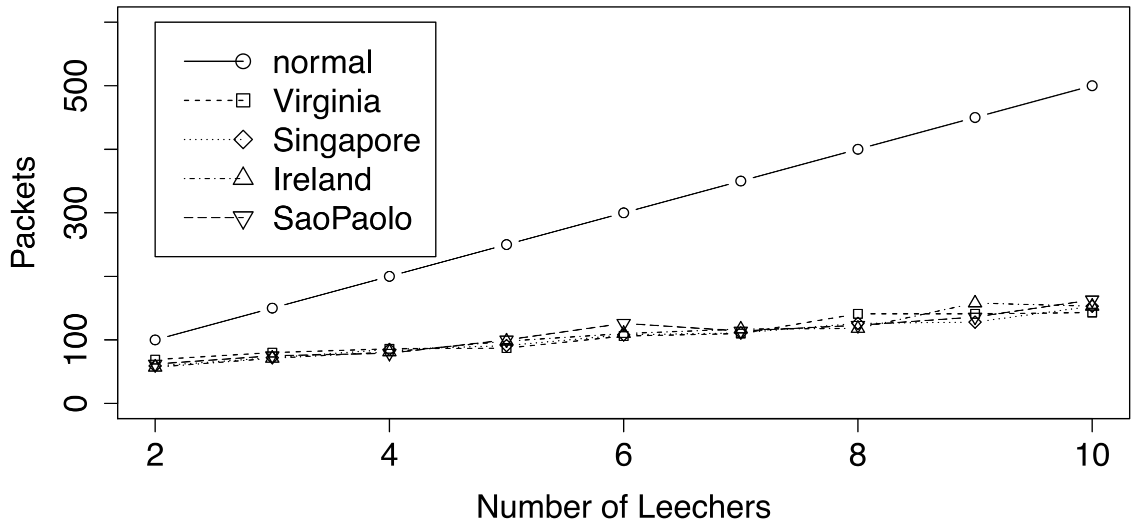


BitTorrent experiments



TCP

CA-TCP



TCP

CA-TCP

Issues (from our 2010 talk)

- How to get congestion control right?
  - Controller can implement this per flow. Conservative default: can_send = 1
- How well do clients deal with unknown options?
  - Works! We tried a dozen different clients (mobile + fixed OSes)
- Uniqueness of resource id
  - We use an optimistic 8 byte hash, could be made longer
  - Including the server IP address would even allow guarantees
- False positives should not be an issue due to router state
  - Core routers without state could be subject to cache poisoning
  - Again, including the server IP address could help to some extent

New Issues

- NATs or firewalls
  - Linux NAT tracks sequence numbers
    - ACKs w/o preceding data packets may not get through
    - May cause the sender to time out and retransmit
  - Sequence number and port rewriting do not matter
  - Re-segmenting does not matter, but may lower efficiency
- Getting TCP options through middleboxes
  - Not an issue in our specific setups
  - Reported problematic in IMC 2011 paper [Honda et al. 2011]
- Asymmetric routing
  - ACKs need to travel the same path as labeled data packets
  - But data packets may come from any TCP other connection
- Route changes are not an issue

Conclusion

- CA-TCP offers an incrementally deployable approach to efficient content distribution
  - For quasi-synchronous access (multicast style)
  - For flash crowds with small intervals between accesses
- Cannot and does not want to compete with web caching
- Segment-level caching supports partial resource caching
- TCP-based operation independent of application protocols
- Incrementally deployable w/o client side changes

Future

- Adaptive AckDelays
- Play with different caching policies
- Understand and exploit dependencies between packets
  - Evicting groups of packets per flow rather than individual ones
- Caching stream sections rather than segments
  - More compatible with the TCP service model
- Build it for a real router