A need for Componentized Transport Protocols

Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Sean Rhea, Timothy Roscoe
U.C. Berkeley and Intel Research Berkeley

1. EXTENDED ABSTRACT

There has been a steady stream of research over the years into *componentized network protocols*: protocol implementations assembled from a variety of building blocks. A promise of such frameworks has generally been flexibility: a protocol stack tailored for a particular application can be easily assembled, usually without writing any new code, by binding protocol objects together.

Despite its conceptual elegance, protocol implementations based on this approach have never caught on, particularly at the transport level. Most applications today make use of a kernel-provided IP stack, and usually TCP for transport, to perform network communication. The consensus is that for both bulk-transfer of data and RPC-like call semantics, TCP appears to be perfectly adequate, and it is not worth inventing something new.

In the last few years, considerable research has been devoted to both structured and unstructured *overlay* and *peer-to-peer* applications. As distributed systems, these applications generally include their own techniques for routing messages on an overlay. Many such deployed systems, including Bamboo [11], MIT Chord [12], and P2 [8], use custom transport protocols that provide TCP-friendly congestion control behavior, but over UDP.

In this work, we are evaluating this design shift by examining features of P2P applications and overlays that motivate their designers to adopt custom transport protocols, and the way in which these applications differ from traditional network-based applications. Our initial examination focuses on four aspects of overlay network applications that motivate customized transport protocols: application-level routing freedom, next hop flexibility, application-level buffer management, and alternative congestion control algorithms.

Application-level routing freedom: Widely-distributed applications have many choices about where to forward a message. Unlike traditional client-server applications, there may be several equivalent end-points for a message (e.g., to retrieve a replica of some object). Moreover, P2P systems usually incorporate some kind of overlay network, even if it is not explicit in the design (e.g., the structured overlay of a DHT, or the link-state overlay of an enterprise network of Microsoft Exchange servers). Overlay networks provide options not only for the destination of a message, but also the overlay path taken to get there.

Designers exploit this new-found freedom to achieve high performance (latency, throughput, reliability, etc.) by implementing sophisticated adaptive policies for forwarding data in the system. For example, a node in the Bamboo DHT [11] constantly measures minimum round-trip times to nodes in its routing table, sets aggressive timeouts, and rapidly resends messages to alternate neighbors if these timeouts are exceeded. This performs dramatically better under churn, since Bamboo can rapidly route around failures and transient load spikes [11].

In terms of the implementation, this inverts a traditional ordering of functionality in a transport stack: destination selection (e.g., the lookup in Bamboo's routing table) now takes place downstream of retries, since successive retries for a message can be sent to different destinations.

Next hop flexibility: In addition to having flexibility in the choice of destination, some P2P applications have the additional property of choosing among a very large set of such destinations—a set whose size and contents are typically not known in advance. A good example is the iterative routing employed by MIT Chord [3] and the Kademlia [9] variants used in eDonkey [2] and trackerless BitTorrent [1].

A problem thus arises in maintaining congestion windows for a large and unpredictable number of destinations, many of which are only needed for a single lookup RPC. To address this problem, DHash++ uses a custom transport protocol called STP [3] that maintains aggregate congestion state for all nodes, rather than the per-node state maintained by TCP, DCCP [4], etc. Consequently, all outgoing packets traverse a single congestion-control instance before being sent to a variety of destinations.

This technique represents a different change in the transport stack implementation from the Bamboo example above. Here, congestion control is performed independently of the destination of messages. Indeed, the decision of where to send the message may be deferred until the congestion window allows it to be sent.

WIP SOSP'05, October 23–26, 2005, Brighton, United Kingdom.

Application-level buffer management: The designers of DCCP point out the benefits to applications of "*late data choice*, where the application commits to sending a particular piece of data very late in the sending process" [7] and suggest using familiar ring-buffer techniques for queuing packets rather than the traditional Unix API. This change allows latency-sensitive applications to revise or replace outgoing packets up until the time when the protocol implementation can send them.

A good motivating example is the use of in-network aggregation techniques for distributed query processors such as PIER [6] and SDIMS [13]. Data is sent up an aggregation tree to the root, and aggregation computation is performed at any intermediate node holding more than one datum at a time. Ideally, each node would send data up the tree eagerly (whenever congestion control allowed it), but otherwise aggregate it with any new data arriving from below.

In practice, traditional protocol implementations (such as Unix TCP) thwart this, since outgoing data may be held at a node in a buffer (before being sent, or for retry purposes), without being available to the query processor for further aggregation. This limitation results in situations where stale results are sent even though a fresher one is available.

We therefore embrace DCCP's notion of late data choice, but extend it further: in addition to being able to revise outgoing packets, widely distributed applications such as distributed query processors benefit from late creation of the packets themselves; an API which provides an upcall to request the next packet to send allows intelligent just-in-time creation of packets, containing an up to date computation at all times.

Furthermore, our approach integrates well with systems that exploit routing freedom to dynamically vary message destinations, as in our first example: a query processor may have several potential "parents" to which it can send partial aggregates [10].

Alternative congestion control algorithms: Finally, TCP's window-based, sender-driven congestion control algorithm may not be the most appropriate for all applications. Floyd et al. [5] propose TFRC: a rate-based, receiver-driven "TCP-friendly" congestion control algorithm (as opposed to window halving congestion control employed by sender-driven algorithms) for flows that benefit from slower changes in sending rate, such as some multimedia traffic. DCCP allows for selection of several different congestion control algorithms, of which TFRC is one. Our own experience with overlay network implementations has shown that TFRC-like approaches have significant advantages, particularly in latency-sensitive overlays that exhibit high loss or unpredictable message delays.

Selection of particular congestion control algorithms can, of course, be achieved via a parameter to the kernel protocol stack, but when combined with the application routing behavior described above, it becomes hard to build a monolithic protocol implementation that can accommodate different congestion control algorithms, themselves occupying different positions in the data path. A more natural construction factors out congestion control into a replaceable module that can be judiciously positioned and configured to application and network characteristics.

Discussion: Taken as an ensemble, the issues above suggest that the solution space for overlay networks is much wider than that for client-server applications. This is in part simply because they are distributed, and hence must interact with and adapt to the network as a whole rather than to a single path through it, blurring the boundary between the application and protocol implementation.

Component based transport protocols provide a natural replacement of black box protocol implementations, with small processing units that can be arranged to form the desired semantics. We are exploring the space of componentized transport protocols in overlay networks using the transport protocol portion of P2, a declarative overlay processor we have built. P2 allows custom transport protocols to be assembled from reusable dataflow building blocks. A variety of diverse but important application behaviors can be achieved naturally within P2's framework, in ways that are hard or impossible to achieve with monolithic kernel implementations of transport protocols such as TCP, RTP, SCTP or DCCP.

2. REFERENCES

[1] Bittorrent goes trackerless: Publishing with bittorrent gets easier! `http://www.bittorrent.com/trackerless.html`.

[2] eDonkey2000 – Overnet. `http://www.edonkey2000.com/`.

[3] F. Dabek, J. Li, E. Sit, F. Kaashoek, R. Morris, and C. Blake. Designing a DHT for low latency and high throughput. In *Proc. NSDI*, March 2004.

[4] S. Floyd, M. Handley, and E. Kohler. Problem Statement for DCCP, June 2005.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM SIGCOMM*, August 2000.

[6] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of PIER: an Internet-scale query processor. In *Proc. CIDR*, January 2005.

[7] J. Lai and E. Kohler. A Congestion-Controlled Unreliable Datagram API, March 2005.

[8] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. In *Proc. ACM SOSP*, October 2005.

[9] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. IPTPS*, March 2002.

[10] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor network streams. In *Proc. ACM SenSys*. November 2004.

[11] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. USENIX Technical Conference*, June 2004.

[12] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.

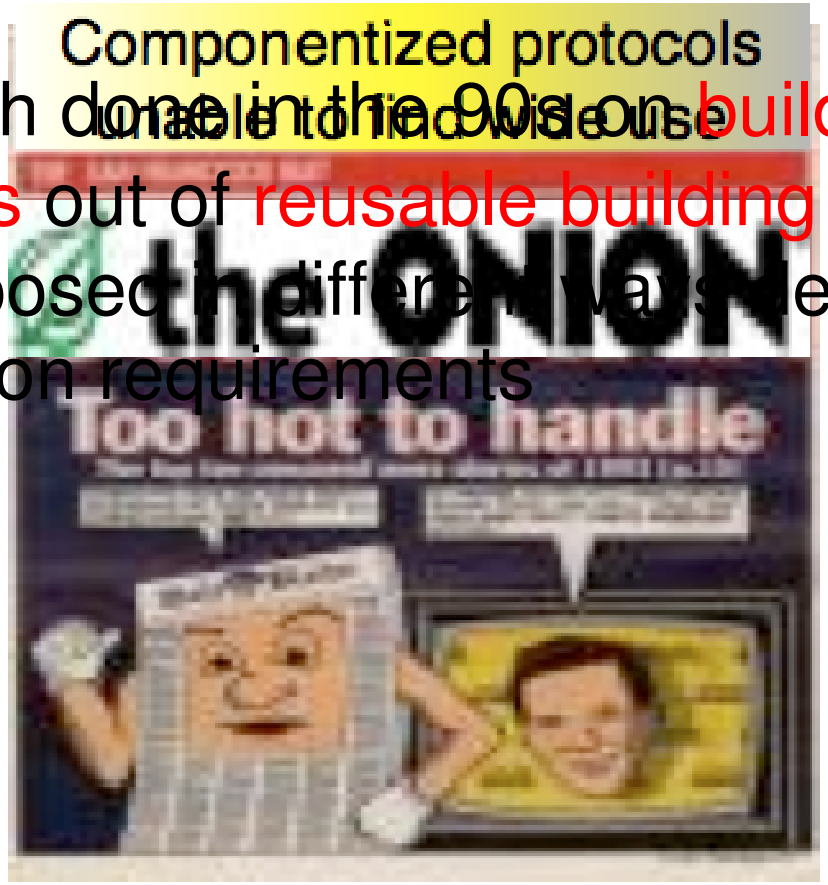
[13] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proc. ACM SIGCOMM*, September 2004.

A need for Componentized Transport Protocols

Tyson Condie
with Joseph M. Hellerstein, Petros Maniatis,
Sean Rhea, Timothy Roscoe
U.C. Berkeley and Intel Research Berkeley

Componentized Protocols

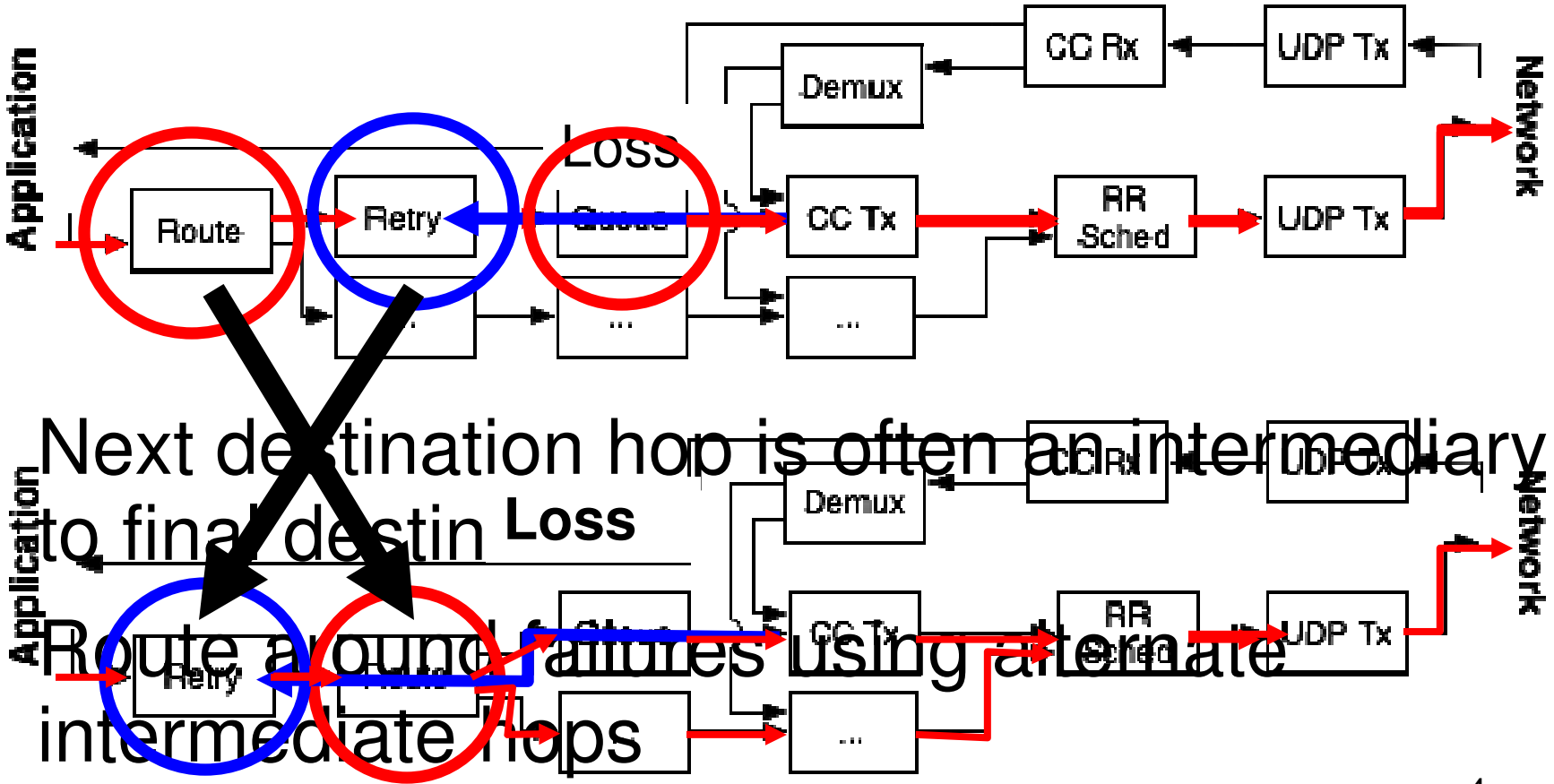
- Research done in the 90s on building transport protocols out of reusable building blocks that can be composed in different ways depending on the application requirements



Why Now?

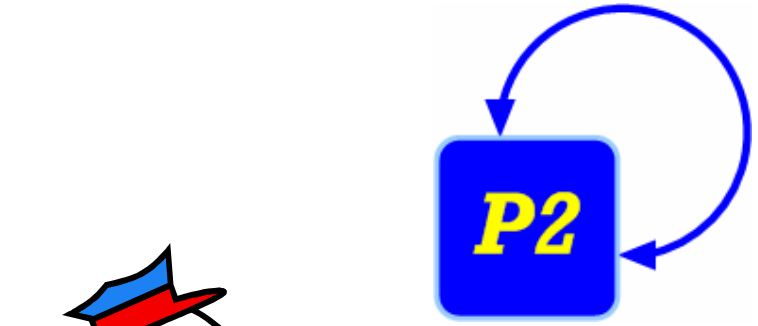
- Applications are becoming more distributed
- Increasing popularity of overlay networks
  - DHT, BitTorrent, Akamai, Narada, Microsoft Exchange
- Overlay network applications demand a highly configurable transport layer
  - Most protocols today are tuned for point-to-point (client/server) communications
  - Forces application programmers to write and tune their own transport layer
- P2 dataflow model extended into network stack
  - Provides highly configurable transport layer

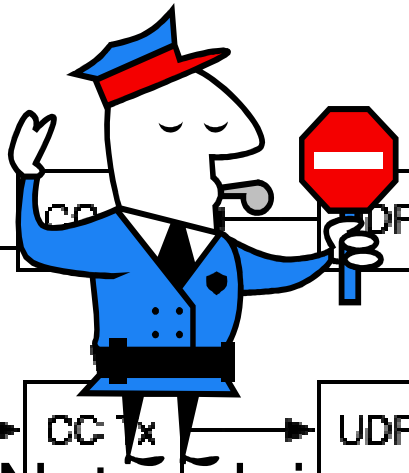
Alternate Path Selection



- Next destination hop is often an intermediary to final destin
- Route around failures using alternate intermediate hops

Application-level Buffer Management



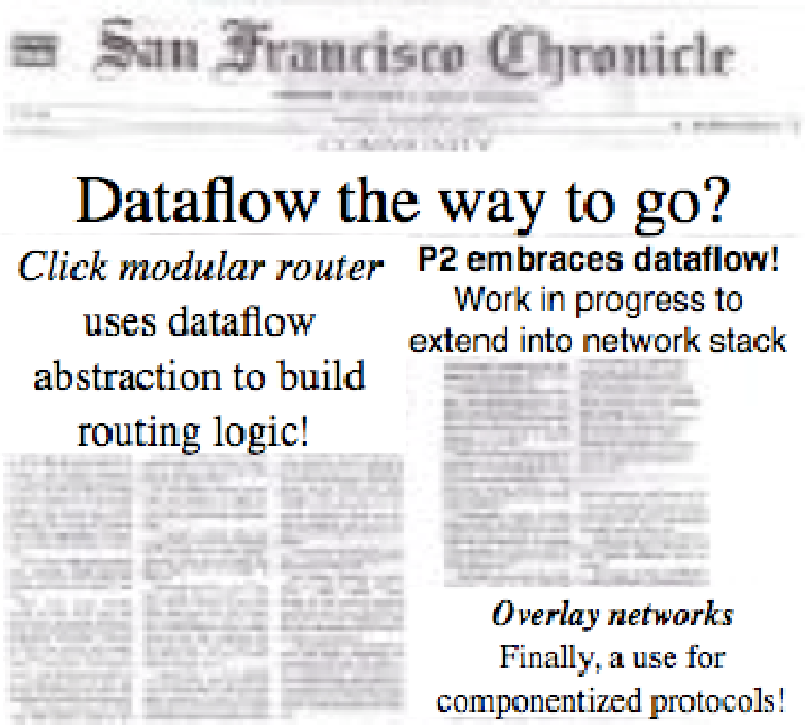


Work in progress

- Automatic static generation of dataflow graphs
  - Graph structure determined by properties of application and network
  - Cost model for choosing among several semantically equivalent dataflows
- Runtime reconfiguration / reoptimization
  - What kinds of modifications and how they are triggered?
  - What kinds of statistics would aid in this effort?
- Declarative language for transport layer
  - Specify high level invariants that are translated into a supporting dataflow

Thank You!

http://p2.cs.berkeley.edu/


San Francisco Chronicle

Dataflow the way to go?

Click modular router uses dataflow abstraction to build routing logic!

P2 embraces dataflow! Work in progress to extend into network stack

Overlay networks Finally, a use for componentized protocols!

T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe. Finally, a use for Componentized Transport Protocols. *HotNets IV*, November 2005.

E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modularrouter.*ACMTrans.Comput.Syst.*,18(3):263–297, 2000.

B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. In *Proc. ACM SOSP*, October 2005.

Componentized Protocols Using Dataflow Abstraction

- Aggregate congestion control
  - Short lived connections
  - Connections that infrequently send messages to a specific destination (next hop)
  - Iterative routing (next hop unknown)