# A Mechanism for TCP-Friendly Transport-level Protocol Coordination

David E. Ott and Ketan Mayer-Patel
University of North Carolina at Chapel Hill
{ott,kmp}@cs.unc.edu

*Abstract*—**In this paper, we identify an emerging and important application class comprised of a set of processes on a cluster of devices communicating to a remote set of processes on another cluster of devices across a common intermediary Internet path. We call these applications *cluster-to-cluster applications*, or *C-to-C applications*. The networking requirements of C-to-C applications present unique challenges. Because the application involves communication between clusters of devices, very few streams will share a complete end-to-end path. At the same time, network performance needs to be measured globally across all streams for the application to employ interstream adaptation strategies. These strategies are important for the application to achieve its global objectives while at the same time realizing an aggregate flow behavior that is congestion controlled and responsive. We propose a mechanism called the *Coordination Protocol* (CP) to provide this ability. In particular, CP makes fine-grained measurements of current network conditions across all associated flows and provides transport-level protocols with aggregate available bandwidth information using an equation-based congestion control algorithm. A prototype of CP is evaluated within a network simulator and is shown to be effective.**

## I. Introduction

Advances in broadband networking, the emergence of information appliances (e.g., TiVo, PDA's, HDTV, etc.), and the now ubiquitous computer provide an environment rife with possibilities for new sophisticated multimedia applications that truly incorporate multiple media streams and interactivity. We believe many of these future Internet applications will increasingly make use of multiple communication and computing devices in a distributed fashion. Examples of these applications include distributed sensor arrays, tele-immersion [13], computer-supported collaborative workspaces (CSCW) [7], ubiquitous computing environments [16], and complex multi-stream, multimedia presentations [17]. In these applications, no one device or computer produces or manages all of the data streams transmitted. Instead, the endpoints of communication are collections of devices. We call applications of this type *cluster-to-cluster applications*, or *C-to-C applications*.

C-to-C applications share three important properties:

- They generate many independent, but semantically related, flows of data.
- While very few flows within the application will share the exact same end-to-end path, all flows will share a common intermediary path between clusters.
- This shared common path is the primary contributor of transmission delay and the source of dynamic network conditions including loss, congestion, and jitter.

Traditional multimedia applications like streaming video generate only a few media streams (e.g., audio and video) which in general originate and terminate at the same devices (e.g., media server to media client). The applications we envision go far beyond this traditional model and include myriad flows of information of many different types communicated between clusters of devices.

Each flow of information may play a different role within the application and thus should be matched with a specific transport-level protocol which provides the appropriate end-to-end networking behavior. Furthermore, these flows will have complex semantic relationships which must be exploited by the application to appropriately adapt to changing network conditions and respond to user interaction.

> *The fundamental problem with current transport-level protocols within the C-to-C application context is their lack of coordination.*

Application streams share a common intermediary path between clusters, and yet operate in isolation from one another. As a result, flows may compete with one another when network resources become limited, instead of cooperating to use available bandwidth in application-controlled ways.

In this paper, we describe and evaluate a mechanism that allows transport-level protocol coordination of separate, but semantically related, flows of data. Our approach is to introduce mechanisms at the first- and last-hop routers which make measurements of current network conditions integrated across all flows associated with a particular C-to-C application. These measurements are then communicated to the transport-level protocols on each endpoint. This enables a coordinated response to congestion across

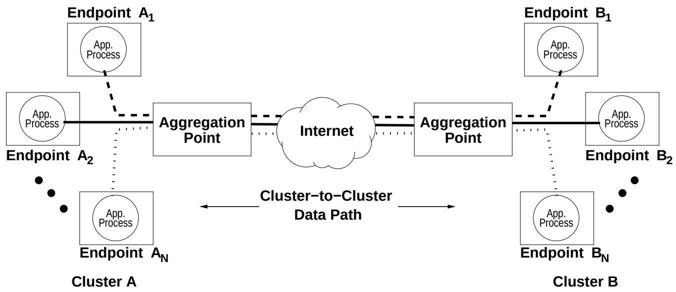


Fig. 1. C-to-C application model.

all flows that reflects application-level goals and priorities. We leverage recent work in equation-based congestion control to ensure that the aggregate bandwidth used by all of the flows is TCP-friendly while allowing the application to allocate available bandwidth to individual flows in whatever manner suits its purposes.

The main contributions of this paper are:

- Identification of the C-to-C class of Internet applications, including a brief motivating example.
- Description of the networking challenges unique to this application type.
- A proposal for a mechanism that provides transport-level protocol coordination in C-to-C applications.
- Evaluation of several aspects of our mechanism using simulation.

The rest of this paper is organized as follows: In Section II, we present the C-to-C application model, describe a motivating example, and discuss networking requirements unique to this class of distributed applications. In Section III, we review related work. We present our solution to the transport-level protocol coordination problem in Section IV, and provide some experimental evaluation in Section V. Section VI mentions future work, and Section VII briefly summarizes the contents of this paper.

## II. Motivation

In this section, we describe in more detail the C-to-C application model, and illustrate it with a specific example. We then discuss the networking challenges associated with this application type, and why there is a need for a protocol coordination mechanism.

### A. C-to-C Application Model

We model a generic C-to-C application as two sets of processes executing on two sets of communication or computing devices. Figure 1 illustrates this model.

A *cluster* is comprised of a set of *endpoints* distributed over a set of *endpoint hosts* (computers or communication devices) and a single *aggregation point*, or *AP*. Each endpoint is a process that sends and/or receives data from another endpoint belonging to a remote cluster. The AP functions as a gateway node traversed by all cluster-to-cluster flows. The common traversal path between aggregation points is known as the *C-to-C data path*.

The AP is typically the first-hop router connecting the cluster to the Internet and the cluster endpoints are typically on the same local area network. This configuration, however, is not strictly required by our model or our proposed mechanism. Our model is intended to capture several important characteristics of C-to-C applications. First, networking resources among endpoints of the same cluster are generally well provisioned for the needs of the application. Second, latency between endpoints of the same cluster is small compared to latency between endpoints on different clusters. Third, there exists a natural point within the network topology through which all cluster-to-cluster communication flows which can act as the AP. Finally, the C-to-C data path between AP's is the main source of dynamic network conditions such as jitter, congestion, and delay. Our overall objective is to coordinate endpoint flows across the C-to-C data path.

### B. An Example Application

A concrete example of a C-to-C application may help clarify the types of applications we envision. In the *Office of the Future*, conceived by Fuchs et al. [13], tens of digital light projectors are used to make almost every surface of an office (walls, desktops, etc.) a display surface. Similarly, tens of video cameras are used to capture the office environment from a number of different angles. At real-time rates, the video streams are used as input to stereo correlation algorithms to extract 3D geometry information. Audio is also captured from a set of microphones. The video streams, geometry information, and audio streams are all transmitted to a remote Office of the Future environment. At the remote environment, the video and audio streams are warped using both local and remote geometry information and stereo views are mapped to the light projectors. Audio is spatialized and sent to a set of speakers. Users within each Office of the Future environment wear shutter glasses that are coordinated with the light projectors.

The result is an immersive 3D experience in which the walls of one office environment essentially disappear to reveal the remote environment and provide a tele-immersive collaborative space for the participants. Furthermore, synthetic 3D models may be rendered and incorporated into both display environments as part of the shared, collaborative experience. Figure 2 is an artistic illustration of the application. A prototype of the application is described in [13].

The Office of the Future is a good example of a C-

Fig. 2. The Office of the Future.

to-C application because the endpoints of the application are collections of devices. Two similarly equipped offices must exchange myriad data streams. While few streams (if any) will share a complete end-to-end communication path, all of the data streams will span a common shared path between the local networking environments of each Office of the Future.

The local network environments are not likely to be the source of congestion, loss, or other dynamic network conditions because they can be provisioned to support the Office of the Future application. The shared Internet path between two Office of the Future environments, however, is not under local control and thus will be the source of dynamic network conditions.

The Office of the Future has a number of complex application-level adaptation strategies that we believe are typical of C-to-C applications. One such strategy, for example, is *dynamic interstream prioritization*. Since media types are integrated into a single immersive display environment, user interaction with any given media type may have implications for how other media types are encoded, transmitted, and displayed. The orientation and position of the user's head, for example, indicates a region of interest within the office environment. Media streams that are displayed within that region of interest should receive a larger share of available bandwidth and be displayed at higher resolutions and frame rates than media streams that are outside the region of interest. When congestion occurs, lower priority streams should react more strongly than higher priority streams. In this way, appropriate aggregate behavior is achieved and dynamic, application-level tradeoffs are exploited.

### C. *Networking Requirements of C-to-C Applications*

A useful metaphor for visualizing the networking requirements of C-to-C applications is to view the communication between clusters as a rope with frayed ends. The rope represents the aggregate data flow between clusters. Each strand represents one particular flow between endpoints. At the ends of the rope, each frayed strand represents a separate path between an endpoint and its local AP. The strands come together at the AP's to form a single aggregate object. While each strand is a separate entity, they share a common fate and purpose when braided together.

With this metaphor in mind, we identify several important networking requirements of C-to-C applications:

- **Preserved end-to-end semantics.**
The transport-level protocol (i.e., TCP, UDP, RTP, RAP, etc.) that is used by each flow is specific to the communication requirements of the data within the flow and the role it plays within the application. Thus, each transport-level protocol should maintain the appropriate end-to-end semantics and mechanisms. For example, if a data flow contains control information that requires in-order, reliable delivery, then the transport-level protocol used (e.g., TCP) should provide these services on an end-to-end basis.
- **Global coordinated measurements of throughput, delay, and loss.**
The application is interested in overall performance which may involve complex interstream adaptation strategies in the face of changing network conditions. Throughput, delay, and loss should be measured across all flows associated with the application as an aggregate. Furthermore, the behavior of individual transport-level protocols must reflect both the end-to-end semantics associated with the protocol as well as application-level adaptation strategies. To achieve this, we need to separate the adaptive dynamic behavior of each transport-level protocol from the mechanisms used to measure current network conditions.
- **TCP-friendliness.**
While the C-to-C application is free to prioritize how bandwidth is allocated among its streams, the total bandwidth used needs to be responsive to congestion. The emerging gold-standard for evaluating responsiveness is TCP-friendliness. Intuitively, a flow of data is considered TCP-friendly if it consumes as much bandwidth as a competing TCP flow consumes given the same network conditions. The advantage of using TCP-friendliness as a standard by which to measure the congestion response of a protocol (or in our case, the aggregate behavior of a set of protocols) is that it ensures "fairness" with the large majority of Internet traffic (including HTTP) that uses TCP as an underlying data transport protocol.

- **Information about peer flows.**

Individual streams within the C-to-C application may require knowledge about other streams of the same application. This knowledge can be used to determine the appropriate adaptive behavior given application-level knowledge about interstream relationships. For example, an application may want to establish a relationship between two flows of data such that one flow consumes twice as much bandwidth as the other.

- **Flexibility for the application.**

A C-to-C application should be free to exploit trade-offs without constraint. That is, a coordination mechanism should not preclude dynamic changes in bandwidth usage among flows, or enforce any particular scheme for establishing bandwidth usage relationships between flows. The application should be free to implement whatever adaptation policy is most appropriate in whatever manner is most appropriate.

## III. Related Work

### A. Application-level Framing

The ideas of this paper are firmly grounded in the concept of Application Level Framing (ALF) [5]. The ALF principle states that networking mechanisms should be coordinated with application-level objectives. As explained above, however, C-to-C applications present unique challenges because these objectives involve interstream tradeoffs for flows that do not share a complete end-to-end path. The actions of heterogeneous protocols distributed among a cluster of devices must be coordinated to incorporate application-specific knowledge. In essence, we are extending the ALF concept to the idea of adapting protocol behavior to reflect application-level semantics. This idea is also well expressed in a position paper by Padmanabhan [11].

### B. Protocol Coordination

The coordination problem presented by C-to-C applications is addressed most directly by Balakrishnan et al. in their work on the Congestion Manager (CM) [3], [1], [2]. CM provides a framework for different transport-level protocols to share information on network conditions, specifically congestion, thus allowing substantial performance improvements. We note, however, that CM flows share the same end-to-end path, while C-to-C flows share only a common intermediary path. The fact that C-to-C senders do not reside on the same host significantly limits the extensibility of the CM architecture to our problem context. CM offers applications sharing the same macroflow a system API and callback mechanisms for coordinating send events. Implementing this scheme using message passing between hosts is at best problematic.

Furthermore, CM makes use of a scheduler to apportion bandwidth among flows. In [3], this is implemented using a Hierarchical Round Robin (HRR) algorithm. We might extend this scheme to the C-to-C context by placing the scheduler at the AP. Doing so, however, results in several problems. First, packet buffering mechanisms are required which, along with scheduling, add complexity to the AP and hurt forwarding performance. Second, packet buffering at the AP lessens endpoint control over send events since endpoint packets can be queued for an indeterminate amount of time. Balakrishnan et al. deliberately avoid buffering for exactly this reason, choosing instead to implement a scheduled callback event. Finally, scheduler configuration is problematic since C-to-C applications are complex and may continually change the manner in which aggregate bandwidth is apprortioned among flow endpoints.

In [9], Kung and Wang propose a scheme for aggregating traffic between two points within a backbone network, and applying the TCP congestion control algorithm to the whole bundle. The mechanism is transparent to applications and does not provide a way for a particular application to make interstream tradeoffs.

Pradhan et al. propose a way of aggregating TCP connections sharing the same traversal path in order to share congestion control information [12]. Their scheme takes a TCP connection and divides it into two separate ("implicit") TCP connections: a "local subconnection" and a "remote subconnection." This scheme, however, breaks the end-to-end semantics of the transport protocol.

[14] describes a scheme for sharing congestion information across TCP flows from different hosts. This work is similar to ours in that a mechanism is introduced within the network itself to coordinate congestion response across a number of different flows which may not share a complete end-to-end path. Their mechanism does not provide the application with information about flows as an aggregate, however, and focuses on optimizing TCP performance by avoiding slow-start and detecting congestion as early as possible.

Finally, Seshan et al. propose the use of *performance servers* that act as a repository for end-to-end performance information [15]. This information may be reported by individual clients or collected by *packet capture hosts*, and then made available to client applications using a query mechanism. The time granularity of performance information is coarse compared to CP, however, since it is intended to enable smart application decisions on connection type and destination, and not ongoing congestion responsive-

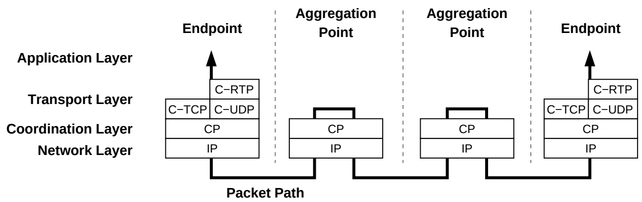


Fig. 3. CP network architecture.

ness. In addition, their work does not associate heterogeneous flows belonging to the same application, or consider the performance of flow aggregates.

### C. Equation-based Congestion Control

TCP-friendly equation-based congestion control has recently matured as a technique for emulating TCP behavior without replicating TCP mechanics. In [6], [10], an analytical model for TCP behavior is derived that can be used to estimate the appropriate TCP-friendly rate given estimates of various channel properties. A number of important recommendations for using their TCP-friendly equation-based congestion control have been documented in [8].

## IV. Coordination Protocol (CP)

In this section we describe our solution to the problem of transport-level protocol coordination in C-to-C applications.

### A. The Coordination Protocol (CP)

We propose the use of a new protocol which operates between the network layer (IP) and transport layer (TCP, UDP, etc.) that addresses the need for transport-level coordination. We call this protocol the *Coordination Protocol (CP)*. The coordination function provided by CP is transport protocol independent. At the same time, CP is distinct from network-layer protocols like IP that play a more fundamental role in routing a packet to its destination.

CP works by attaching probe information to packets transmitted from one cluster to another. As additional probe information is returned along the reverse cluster-to-cluster data path, a picture of current network conditions is formed by the AP and shared among endpoints within the local cluster. A consistent view of network conditions across flows follows from the fact that the same information is shared among all endpoints.

Figure 3 shows our proposed network architecture from a stack implementation point of view. CP exists on each endpoint device participating in the C-to-C application, as well as on the two aggregation points (APs) on either end of the cluster-to-cluster data path. Routers on the data path between APs need *not* be CP-enabled since they examine only the IP header of each incoming packet in order to route the packet in their customary manner.

The decision to insert CP between the network and transport layer rather than handling coordination at the application level requires some justification. Of primary importance to us is the preservation of end-to-end semantics. An alternative would be for each endpoint to send to a multiplexing agent who would send the data, along with probe information, to a demultiplexing agent on the remote cluster. By breaking the communication path into three stages, however, the end-to-end semantics of individual transport-level protocols have been severed. Such a scheme would also mandate that application-level control is centralized and integrated into the multiplexing agent.

Furthermore, we note that CP logically belongs between the network and transport layer. While the network layer handles the next-hop forwarding of individual packets and the transport layer handles the end-to-end semantics of individual streams, CP is concerned with streams that share a significant number of hops along the forwarding path but do not share the same end-to-end path. This relaxed notion of a stream bundle logically falls between the strict end-to-end notion of the transport-level and the independent packet notion of the network-level.

Finally, placement of CP between the network and transport layer allows for greater efficiency. In an application-level implementation of CP, information on network conditions (e.g., round trip time between APs) must pass up through an endpoint's protocol stack to the application layer. The information must then be passed back down to the transport layer where sending rate adjustments can be made in response to the information. In contrast, a distinct coordination layer allows for the information to be received and passed directly to the transport layer in a single pass as the incoming packet is processed by each layer of its endpoint's network stack.

While we acknowledge that implementing CP mechanisms at the application layer is indeed possible, we believe there are distinct advantages to the approach we have chosen. We emphasize, however, that the relative merits or drawbacks of our scheme are merely implementation issues that should not obscure the fundamental problem of C-to-C flow coordination described in this paper.

### B. CP Packet Headers

Figure 4 shows a CP data packet. CP encapsulates transport-level packets by prepending a 16-byte header and indicating in the protocol field which transport level protocol is associated with the packet. In turn, IP encapsulates CP packets and indicates in its protocol field that CP is being used.

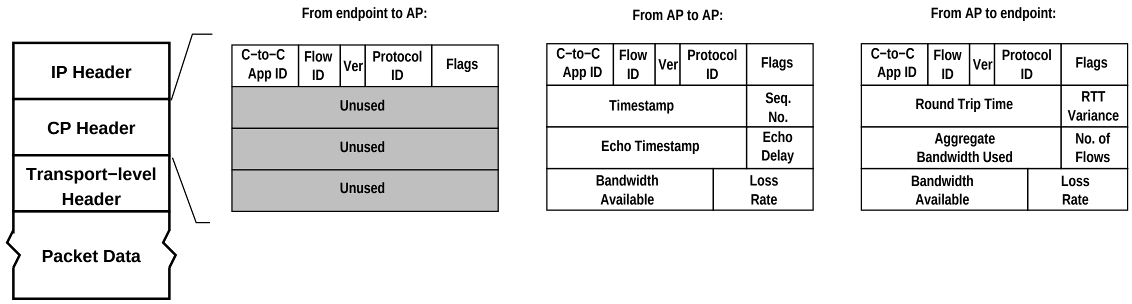


Fig. 4. CP packet structure.

Each CP header contains an application identifier associating the packet with a particular C-to-C application, and a flow identifier indicating which flow from a given endpoint host the packet belongs to. The triple *(application id, IP address, flow id)* uniquely identifies each flow within the C-to-C application, and hence the source of each CP packet. The header also contains a version number and a flags field.

The remaining contents of the CP header vary according to the changing role played by the header as it traverses the network path from source endpoint to destination endpoint. As the packet passes from the source endpoint to its local AP, the header merely identifies the cluster application it is associated with and its sender. As the packet is sent from the source's local AP to the remote AP, the header contains probe information used to measure round trip time, detect packet loss, and communicate current loss rate and bandwidth availability. As the packet is forwarded from the remote AP to its destination endpoint, the header contains information on application bandwidth use, flow membership, round trip time, loss rate, and bandwidth availability.

### C. Basic Operation

The basic operation of CP is as follows.

- **As packets originate from source endpoints.**

The CP header is included in the application packet indicating the source of the packet and the cluster application it is associated with.

- **As packets arrive at the local AP.**

CP will process the identification information arriving in the CP header, and note the packet's size and arrival time. Part of the CP header will then be overwritten, allowing the AP to communicate congestion probe information to the remote AP.

- **As packets arrive at the remote AP.**

The CP header is processed and used to detect network conditions. Again, part of the CP header is overwritten to communicate network condition information, along with information on cluster application size and bandwidth usage, to the remote remote endpoint.

- **As packets arrive at the destination endpoint.**

CP processes network condition information from the CP header and passes it on to the transport-level protocol and the application.

### D. State Maintained by an AP

An AP maintains a table of active cluster applications, each entry of which exists as soft state. When a packet arrives with an unknown cluster identifier in its CP header, a new entry will be created in the table and CP probe mechanisms will become active for that application. Similarly, if no CP packet has been seen for a particular cluster identifier $i$, then the entry will time out and be removed from the application table. Use of soft state in this manner is both flexible and lightweight in that it avoids the need for explicit configuration and ongoing administration.

For each cluster application, the AP monitors the number of participating flows, and the number and size of packets received during a given interval. Weighted averages are calculated to dampen the effect of packet bursts. The information is passed back to local cluster endpoints using the CP header whenever a packet arrives from the remote AP on route to a local endpoint. If no such packet arrives within a specified time period, then a *report packet* is created and "pushed" to each endpoint informing them of cluster application membership and bandwidth usage, as well as current network conditions.

An AP also maintains probe state, including a current packet sequence number, estimated round trip time and mean deviation, a loss history and estimated loss rate, and a bandwidth availability calculation. Use of these mechanisms is described below.

### E. Detecting Network Delay and Loss

A primary function of CP is to measure network delay and detect packet loss along the cluster-to-cluster data path. Figure 5, Table I, and Table II together illustrate how

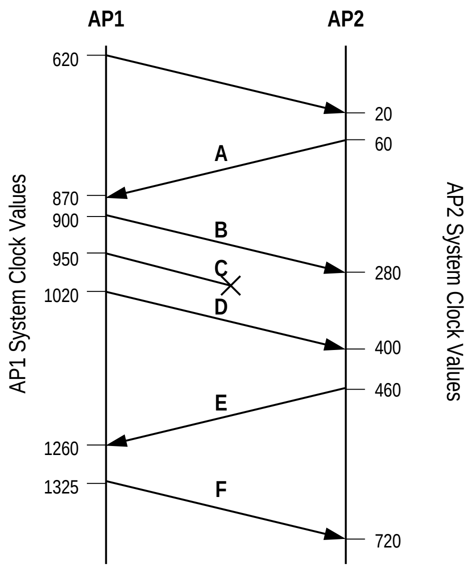


Fig. 5. Timeline of AP packet exchanges.

| Packet | Sequence Number | Time-stamp | Time-stamp Echo | Echo Delay |
|---|---|---|---|---|
| B | 14 | 900 | 60 | 30 |
| C | 15 | 950 | 60 | 80 |
| D | 16 | 1020 | 60 | 150 |
| F | 17 | 1325 | 460 | 65 |

TABLE I
INFORMATION IN CP HEADER FOR PACKETS TRAVELING FROM AP1 TO AP2 IN FIGURE 5.

information in the CP header is used to make these measurements.

Each packet passing from one AP to another has several numbers inserted into its CP header. The first is a sequence number that increases monotonically for every packet sent. A remote AP may use this number to observe gaps (and reorderings) in the aggregate flow of cluster application packets that it receives. In this way, it can detect losses and infer congestion. In our example, AP2 detects the loss of packet C when the sequence number received skips from 14 (packet A) to 16 (packet D).

In addition, a timestamp is sent along with the sequence number indicating the time at which the AP sent the packet. The remote AP will then echo the timestamp of the last sequence number received by placing the value in the CP header of the next packet traveling on the reverse path back to the sending AP. Along with this timestamp, a delay value will also be given indicating the length of time between the arrival of the sequence number at the AP and the time the AP transmitted the echo.

| Packet | Sequence Number | Time-stamp | Time-stamp Echo | Echo Delay |
|---|---|---|---|---|
| A | 76 | 60 | 620 | 40 |
| E | 77 | 460 | 1020 | 60 |

TABLE II
INFORMATION IN CP HEADER FOR PACKETS TRAVELING FROM AP2 TO AP1 IN FIGURE 5.

By noting the time when a packet is received ($T_{arrival}$), the AP can calculate the round trip time as $(T_{arrival} - T_{echo}) - T_{delay}$. In our example, AP2 receives packet B at time 280. The CP header contains the timestamp echo 60 and an echo delay value of 30. Thus, the round trip time is calculated as $280 - 60 - 30 = 190$. A weighted average of these round trip time calculations is used to dampen the effects of burstiness.

Note that because sequence numbers in the CP header do not have any transport-level function, CP can use whatever C-to-C application packet is being transmitted next to carry this information. Since the packets of multiple flows are available for this purpose, this mechanism can be used for fine-grained detection of network conditions along the cluster-to-cluster data path.

We also observe that there is no one-to-one correspondence between timestamps sent and timestamps echoed between APs. It may be the case that more than one packet is received by a remote AP before a packet traveling along the opposite path is available to echo the most current timestamp. The AP simply makes use of available packets in a best effort manner. In Figure 5 this can be seen as AP2 receives both packets B and D before packet E is available to send on the return path. Likewise, an AP may echo the same timestamp more than once if no new CP packet arrives with a new timestamp. In our example, this occurs when AP1 sends packets B, C, and D with a timestamp echo value of 60 which it received from packet A.

### *F. Calculating Loss Rate and Bandwidth Availability*

Calculation of loss rate and bandwidth availability make use of equation-based congestion control methods described in Floyd et al. in their work on TCP-friendly rate control (TFRC) [8].

Loss rate, a central input parameter into the bandwidth availability equation, is calculated using a *loss history* and *loss events* rather than individual packet losses. By using a loss event rate rather than a simple lost packet rate, we provide a more stable handling of lost packet bursts. The reader is referred to [6] for more details.

Calculation of available bandwidth makes use of the equation:

$$X = \frac{s}{R\sqrt{\frac{2bp}{3}} + t_{RTO}(3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \quad (1)$$

where $X$ is the transmit rate (bytes/sec), $s$ is the packet size (bytes), $R$ is the round trip time (sec), $p$ is the loss event rate on the interval [0,1.0], $t_{RTO}$ is the TCP retransmission timeout (sec), and $b$ is the number of packets acknowledged by a single TCP acknowledgement.

The resulting quantity, which we refer to as current *bandwidth availability*, is calculated at the remote AP, and then passed using the CP header to each endpoint in the cluster. Similarly, the event loss rate is also passed on to endpoints to inform them of current network conditions.

We emphasize here that the use of the above equation to calculate bandwidth availability for the cluster application makes the *aggregate* data flow from one AP to another TCP-compatible.

### G. Transport-level Protocols

Transport-level protocols at the endpoints are built on top of CP in the same manner that TCP is built on top of IP. CP provides these transport-level protocols with a consistent view of network conditions, including aggregate bandwidth availability, loss rate, and round trip delay measurements. In addition, it informs endpoints of the aggregate bandwidth usage and the current number of flows in the cluster application. A transport-level protocol will in turn use this information, along with various configuration parameters, to determine a data transmission rate and related send characteristics.

In Figure 3, we show several possible transport-level protocols (C-TCP, C-UDP, and C-RTP) which are meant to represent coordinated counterparts to existing protocols. A coordinated version of UDP (C-UDP) simply makes the above information available directly to the application which may modify its sending rate according to an application-specific rule or bandwidth sharing scheme.

A coordinated version of TCP (C-TCP) may consider acknowledgements only as an indicator of successful transfer. The burden of round trip delay determination and congestion detection can be relegated entirely to CP. Send rate adjustments at the transport level are the combined result of configuration information given by the application (e.g., a maximum sending rate), and information on current network conditions as provided by CP.

While C-UDP and C-TCP represent adaptations of familiar transport-level protocols, we believe that other coordinated transport-level protocols are possible. Such protocols will make use of CP information and application semantics to adjust sending rates to meet application-specific objectives.

### H. Application-level Programming Interface

Endpoint implementations of CP provide a modified socket interface to the application layer. With this interface, the application is able to associate its data flow with a particular cluster application and interact more directly with CP-related mechanisms in two ways.

First, the application may use the interface to communicate configuration information to the transport-level. For example, an application may wish to restrict the transport-level sending rate to no more than some maximum value. Or, an application may instruct the transport layer to send at only some fraction of the available bandwidth given various conditions. Such configuration is made possible by a set of system calls which allow applications to pass functions to the transport layer which operate on reported CP values in order to calculate an instantaneous sending rate.

The application may also use the interface to access CP information directly. Thus, a system call is provided which allows the application to query, for example, available bandwidth, round trip time, and the current loss rate. Obtaining this information directly is of particular importance when the application itself controls its own send rate (e.g., C-UDP) rather than relegating such control to the transport-level protocol (e.g., C-TCP).

### I. Endpoint Coordination

While a goal of C-to-C applications is to maintain congestion responsiveness on an aggregate level, how this goal is realized is left entirely to the application. The approach of CP is to avoid the use of traffic shaping or packet scheduling mechanisms at the AP, but instead to provide application endpoints with bandwidth availability "hints" and other information about changing network conditions. An application may then apportion bandwidth among endpoints by configuring them to respond to these hints in ways which meet the objectives of the application as a whole.

For example, a C-to-C application may configure secondary streaming endpoints to reduce their sending rate, or stop sending altogether, in response to a drop in available bandwidth below a particular threshold value. At the same time, a primary stream endpoint may continue to send at its original rate, and a control endpoint may increase its sending rate somewhat in order to transmit important commands telling the receive side how to respond to the change. Despite these differences in response be-

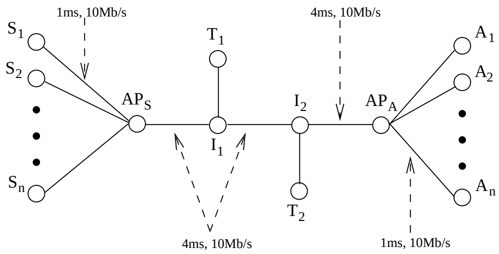


Fig. 6. Simulation testbed in ns2.

havior, the aggregate bandwidth usage drops appropriately to match the bandwidth availability hint given.

CP provides a C-to-C application with the mechanisms needed to make coordinated adaptation decisions which reflect the current state of the network and the application's objectives. We believe it unnecessary to provide additional mechanisms which enforce bandwidth usage among endpoints since each belongs to the same application and thus shares the same objectives. In addition, endpoint configuration may be complex and change dynamically making the implementation of an enforcement scheme inherently problematic.

## V. Evaluation

In this section, we evaluate the behavior of CP using the network simulator *ns-2* [4]. We focus here on our implementation of C-TCP, the coordinated counterpart to TCP.

C-TCP, like TCP, implements reliability through the use of acknowledgement packets, timeouts, and retransmission. Unlike window-based TCP, however, C-TCP is a rate-based implementation which adjusts its instantaneous send rate based on bandwidth availability information supplied by CP, and configuration information supplied by the application. Our implementation of C-TCP draws heavily from TFRC [8], except that loss and send rate calculations are handled by APs communicating over the C-to-C data path, and TCP-compatibility, as defined in [6], is achieved on an aggregate and not per-flow level.

### A. Network Topology

Our simulation topology is pictured in Figure 6. A cluster of sending agents is labeled $S_1$ through $S_n$, with its local aggregation point labeled $AP_S$. A remote cluster of ACK (acknowledgement) agents is labeled $A_1$ through $A_n$, with its aggregation point labeled $AP_A$. $I_1$ and $I_2$ are intermediary nodes used to create a congested link, and $T_1$ and $T_2$ are used for traffic generation.

Propagation delay on links $AP_S$-$I_1$, $I_1$-$I_2$, and $I_2$-$AP_A$ is configured to be 4 msec, while it is only 1 msec on links $S_i$-$AP_S$ and $A_i$-$AP_A$. The link capacity for all links is 10 Mb/s, except for links $T_1$-$I_1$ and $T_2$-$I_2$ where link capacity is 100 Mb/s. This allows traffic generators to increase traffic over link $I_1$-$I_2$ to any desired level.

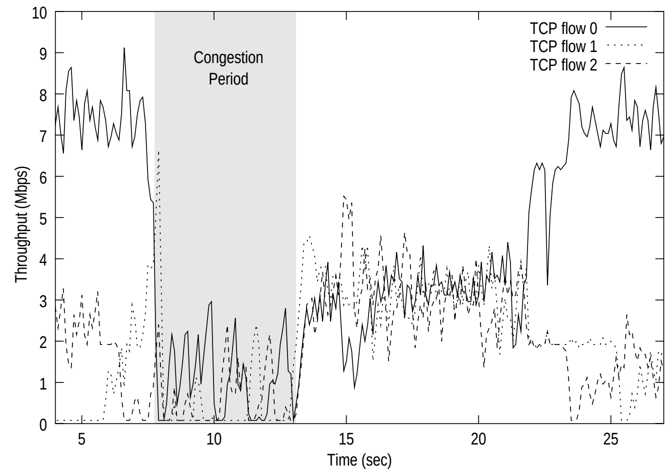


Fig. 7. TCP flows competing for bandwidth during congestion.

Trace data is collected as it is transmitted from $AP_S$ to $I_1$ since this allows us to observe sending rates before additional traffic on the link $I_1$-$I_2$ causes queuing delays, drops, or jitter not reflective of cluster endpoint sending rates.

TCP and C-TCP flows in this section use an infinitely large data source and send at the maximum rate allowed by their respective algorithms. Congestion periods are created by configuring $T_1$ and $T_2$ to generate constant bitrate traffic across the link $I_1$-$I_2$. In particular, a CBR agent sending at a constant 7.5-9.0 Mb/s from $T_1$ to $T_2$ competes with data traffic from $S_1$-$S_n$ over link $I_1$-$I_2$.

### B. Behavior of Uncoordinated TCP Flows

To better see the problem addressed by CP, we first examine how several TCP connections behave without coordination. In Figure 7, we see the throughput plot of three TCP connections as network congestion occurs between time 8.0 and 13.0 seconds. Flow 0 belongs to an application process with higher bandwidth requirements than processes associated with flows 1 and 2. This can be seen clearly at the right and left edges of the plot when flow 0 takes its full share of the bandwidth under congestion-free circumstances.

We note the following observations:

- During the congestion interval, all three flows compete with one another and receive a roughly similar portion of the available bandwidth.
- The flows continue to compete in a similar fashion during the period directly afterward (time 13.0 through 22.0) as each struggles to send accumulated data and regain its requisite level of bandwidth.

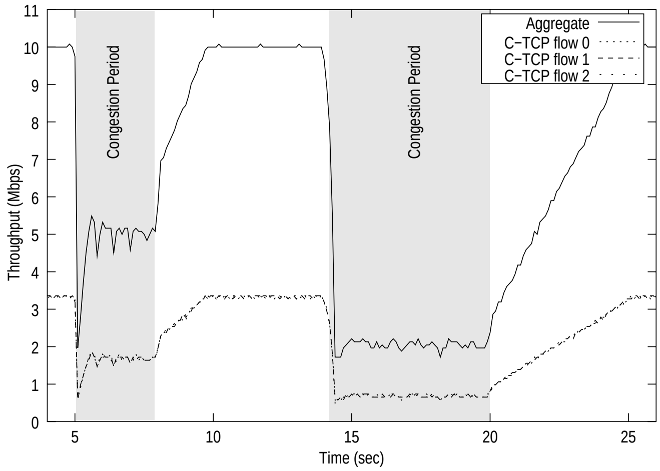


Fig. 8. C-TCP flows sharing bandwidth equally.

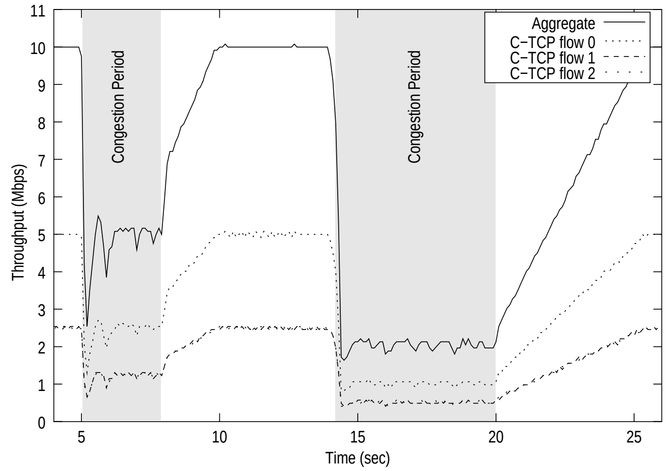


Fig. 9. C-TCP flows sharing bandwidth proportionally.

• The bandwidth used by each flow is characterized by jagged edges, often criss-crossing one another. This makes sense since each flow operates independently, searching the bandwidth space by repeatedly ramping up and backing off.

### *C. Behavior of C-TCP Flows*

We postulate here that use of the Coordination Protocol (CP) should be distinctive in at lease two ways. First, since all flows make use of the same bandwidth availability calculation, round trip time, and loss rate information, bandwidth usage patterns among CP flows should be much smoother. That is, there should be far fewer jagged edges and less criss-crossing of individual flow bandwidths as flows need not search the bandwidth space in isolation for a maximal send rate.

Second, the use of bandwidth by a set of CP flows should reflect the priorities and configuration of the application–including intervals of congestion when network resources become limited.

To test these hypotheses, we implemented three simple bandwidth sharing schemes which reflect different objectives an application may wish to achieve on an aggregate level. We note here that more schemes are possible, and the mixing of schemes in complex, application-specific ways is an open area of research.

Figure 8 shows a simple *equal bandwidth* sharing scheme in which C-TCP flows divide available bandwidth ($B$) equally among themselves. ($R_i = B/N$ where $R_i$ is the send rate for sending endpoint $i$, and $N$ is the number of sending endpoints.) The aggregate plot line shows the total bandwidth used by the multi-flow application at a given time instant. While not plotted on the same graph, this line closely corresponds to bandwidth availability values calculated by APs and communicated to cluster endpoints.

Figure 8 confirms our hypothesis that usage patterns among CP flows should be far smoother, and avoid the jagged criss-crossing effect seen in Figure 7. This is both because flows are not constantly trying to ramp up in search of a maximal sending rate, and because of the use of weighted averages in the bandwidth availability calculation itself. The latter has the effect of dampening jumps in value from one instant to the next.

Figure 9 shows a *proportional bandwidth* sharing scheme among C-TCP flows. In this particular scheme, flow 0 is configured to take .5 of the bandwidth ($R_0 = .5 * B$), while flows 1 and 2 evenly divide the remaining portion for a value of .25 each ($R_1 = R_2 = .25 * B$).

Figure 9 confirms our second hypothesis above by showing sustained proportional sharing throughout the entire time interval. This includes the congestion intervals (times 5.0-8.0 and 14.0-20.0) and post-congestion intervals (times 8.0-10.0, 20.0-25.0) when TCP connections might still contend for bandwidth.

In Figure 10, we see a *constant bandwidth* flow in conjunction with two flows equally sharing the remaining bandwidth. The former is configured to send at a constant rate of 3.5 Mb/s or, if it is not available, at the bandwidth availability value for that given instant. ($R_0 = min(3.5Mb/s, B)$). Flows 1 and 2 split the remaining bandwidth or, if none is available, send at a minimum rate of 1Kb/s. ($R_1 = R_2 = max((B - R_0)/2, 1Kb/s)$)

We observe that flows 1 and 2 back off their sending rate almost entirely whenever flow 0 does not receive its full share of bandwidth. We also note that while flow 0 is configured to send at a constant rate, it never exceeds available bandwidth limitations during time of congestion.

We emphasize once again the impossibility of achieving results like Figure 9 and Figure 10 in an application

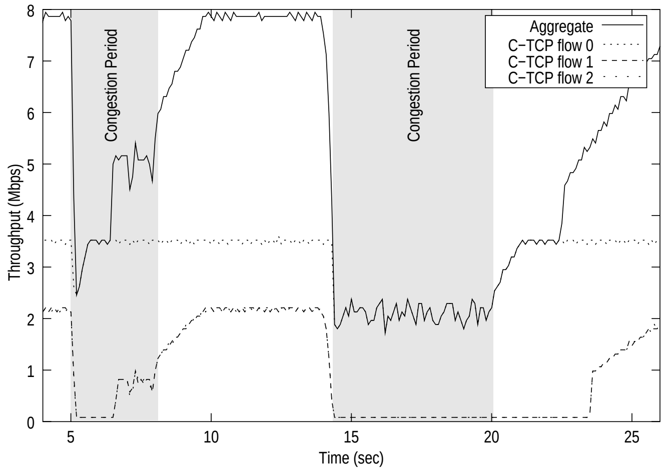


Fig. 10. A constant bandwidth C-TCP flow with two C-TCP flows sharing remaining bandwidth.

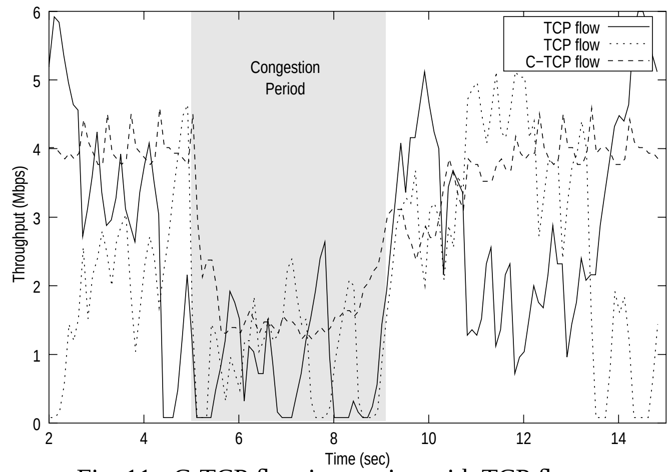


Fig. 11. C-TCP flow interacting with TCP flows.

without the transport-level coordination provided by CP.

### D. TCP-Friendliness

The TCP-friendliness of aggregate CP traffic is established by using the equation-based congestion control method described in [6] and used by TFRC [8].

While equation-based rate control guarantees TCP-compatibility over long time intervals, Figure 11 illustrates informally the behavior of a single C-TCP connection with two TCP connections during a short congested interval (time 5.0 through 9.0). Here we're interested in verifying that the behavior of the C-TCP flow does indeed appear to be compatible with that of the TCP flows.

In general, we see that the C-TCP connection mixes reasonably well with the TCP connections, receiving approximately an equal share of the available bandwidth. In addition, we once again observe the smoothness of its rate adjustments compared to the far more volatile changes in TCP flows.

## VI. Future Work

We believe transport-level protocol coordination in C-to-C applications to be fertile area for future work. In particular, much work remains to be done on new transport protocols better equipped to make use of network condition and cluster flow information. These protocols may provide end-to-end semantics which are more specific to an application's needs than current all-purpose protocols like TCP and UDP.

Flow coordination in a C-to-C application within this paper has meant the sharing of bandwidth from a single bandwidth availability calculation, equivalent to a single TCP-compatible flow. Future work might focus on sharing the equivalent of more than one TCP-compatible flow, just as many applications (eg., Web browsers) open more than one connection to increase throughput by parallelizing end-to-end communication.

The assumption that local networks on each end of a C-to-C application can always be provisioned to minimize network delay and loss may not always be true. For example, wireless devices may introduce delay and loss inherent to the technology itself. How CP can be adapted to accomodate this situation is an area of future work. One idea is to use CP for distinguishing between congestion sources. End-to-end estimates of delay and loss could be compared with those of CP in order to determine whether congestion is local or within the network.

Finally, the impact of CP mechanisms on forwarding performance at the AP is an important issue that deserves further study. We conjecture here that the impact will be modest since per-packet processing largely amounts to simple accounting and checksum computations, and an AP avoids entirely the need for buffering or scheduling mechanisms. An actual implementation is required, however, before any meaningful analysis can be done.

## VII. Summary

In this paper, we have identified a class of distributed applications known as *cluster-to-cluster (C-to-C) applications*. Such applications have semantically related flows that share a common intermediary path, typically between first- and last-hop routers. C-to-C applications require transport-level coordination to better put the application in control over bandwidth usage, especially during periods when network resources become limited by congestion. Without coordination, high-priority flows may contend equally with low-priority flows for bandwidth, or receive no bandwidth at all, thus preventing the application from meeting its objectives entirely.

We have proposed the Coordination Protocol (CP) as a way of coordinating semantically related flows in application-controlled ways. CP operates between the network (IP) and transport (TCP, UDP) layers, offering C-to-C flows fine-grained information about network conditions along the cluster-to-cluster data path, as well as information about application flows as an aggregate. In particular, CP makes use of equation-based rate control methods to calculate bandwidth availability for the entire C-to-C application. This results in aggregate flow rates that are highly adaptive to changing network conditions and TCP-compatible.

## REFERENCES


[1] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System Support for Bandwidth Management and Content Adaptation in Internet Applications. *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI)*, pages 213–226, October 2000.

[2] H. Balakrishnan and S. Seshan. *RFC 3124: The Congestion Manager*, June 2001.

[3] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An Integrated Congestion Management Architecture for Internet Hosts. *Proceedings of ACM SIGCOMM*, September 1999.

[4] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in Network Simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[5] D.D. Clark and D.L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Proc. ACM SIGCOMM 1990, Computer Communication Review*, 20(4):200–208, September 1990.

[6] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. *Proceedings of ACM SIGCOMM*, pages 43–56, 2000.

[7] J. Grudin. Computer-Supported Cooperative Work: Its History and Participation. *Computer*, 27(4):19–26, 1994.

[8] M. Handley, J. Padhye, S. Floyd, and J. Widmer. *TCP Friendly Rate Control (TFRC): Protocol Specification*. IETF, May 2001. Internet Draft, work in progress.

[9] H.T. Kung and S.Y. Wang. TCP Trunking: Design, Implementation and Performance. *Proc. of ICNP '99*, November 1999.

[10] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. *Proceedings of ACM SIGCOMM*, 1998.

[11] V.N. Padmanabhan. Coordinated Congestion Management and Bandwidth Sharing for Heterogeneous Data Streams. *Proceedings of the 9th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 187–190, 1999.

[12] P. Pradhan, T. Chiueh, and A. Neogi. Aggregate TCP Congestion Control Using Multiple Network Probing. *Proc. of IEEE ICDCS 2000*, 2000.

[13] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. *Proceedings of ACM SIGRAPH 98*, 1998.

[14] S. Savage, N. Cardwell, and T. Anderson. The Case for Informed Transport Protocols. *Proceedings of HotOS VII*, March 1999.

[15] Srinivasan Seshan, Mark Stemm, and Randy H. Katz. SPAND: Shared Passive Network Performance Discovery. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[16] M. Weiser. Some Computer Science Problems in Ubiquitous Computing. *Communications of the ACM*, 36(7):75–84, July 1993.

[17] T.-P. Yu, D. Wu, K. Mayer-Patel, and L.A. Rowe. DC: A Live Webcast Control System. *Proc. of SPIE Multimedia Computing and Networking*, 2001.