Celerity: A Low-Delay Multi-Party Conferencing Solution *

Xiangwen Chen
Dept. of Information Engineering
The Chinese University of Hong Kong

Minghua Chen
Dept. of Information Engineering
The Chinese University of Hong Kong

Baochun Li
Dept. of Electrical and Computer Engineering
The University of Toronto

Yao Zhao
Alcatel-Lucent

Yunnan Wu
Facebook Inc.

Jin Li
Microsoft Research

ABSTRACT

In this paper, we attempt to revisit the problem of multi-party conferencing from a practical perspective, and to rethink the design space involved in this problem. We believe that an emphasis on low end-to-end delays between any two parties in the conference is a must, and the source sending rate in a session should adapt to bandwidth availability and congestion. We present *Celerity*, a multi-party conferencing solution specifically designed to achieve our objectives. It is entirely Peer-to-Peer (P2P), and as such eliminating the cost of maintaining centrally administered servers. It is designed to deliver video with low end-to-end delays, at quality levels commensurate with available network resources over arbitrary network topologies where *bottlenecks can be anywhere in the network*. This is in contrast to commonly assumed P2P scenarios where bandwidth bottlenecks reside only at the edge of the network. The highlight in our design is a distributed and adaptive rate control protocol, that can discover and adapt to arbitrary topologies and network conditions quickly, converging to efficient link rate allocations allowed by the underlying network. In accordance with adaptive link rate control, source video encoding rates are also dynamically controlled to optimize video quality. We have implemented *Celerity* in a prototype system, and demonstrate its superior performance over existing solutions in a local experimental testbed and over the Internet.

Categories and Subject Descriptors

C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed System—*Distributed applications*; H.4.3 [**INFORMATION SYSTEMS APPLICATIONS**]: Communications Applications—*video conferencing*

General Terms: Algorithms, Design, Experimentation, Performance

Keywords: Peer-to-peer, Multi-party video conferencing, Low delay, Utility maximization, Arbitrary network topology

1. INTRODUCTION

With the availability of front-facing cameras in high-end smartphone devices (such as the Samsung Galaxy S and the iPhone 4), notebook computers, and HDTVs, *multi-party* video conferencing, which involves more than two participants in a live conferencing session, has attracted a significant amount of interest from the industry. Skype, for example, has recently launched a monthly-paid service supporting multi-party video conferencing in its latest version (Skype 5) [1]. Skype video conferencing has also been recently supported in a range of new Skype-enabled televisions, such as the Panasonic VIERA series, so that full-screen high-definition video conferencing can be enjoyed in one's living room. We argue that these new conferencing solutions have the potential to provide an immersive human-to-human communication experience among remote participants. Such an argument has been corroborated by many industry leaders: Cisco predicts that video conferencing and tele-presence traffic will increase ten-fold between 2008-2013 [2].

While traffic flows in a live multi-party conferencing session are fundamentally represented by a multi-way communication process, today's design of multi-party video conferencing systems are engineered in practice by composing communication primitives (*e.g.,* transport protocols) over uni-directional feed-forward links, with primitive feedback mechanisms such as various forms of acknowledgments in TCP variants or custom UDP-based protocols. We believe that a high-quality protocol design must harness the full potential of the multi-way communication paradigm, and must guarantee the stringent requirements of low end-to-end delays, with the highest possible source coding rates that can be supported by dynamic network conditions over the Internet.

From the industry perspective, known designs of commercially available multi-party conferencing solutions are either largely server-based, e.g., Microsoft Office Communicator, or are separated into multiple point-to-point sessions (this approach is called Simulcast), e.g., Apple iChat. Server-based solutions are susceptible to central resource bottlenecks, and as such scalability becomes a main concern when multiple conferences are to be supported concurrently. In the Simulcast approach, each user splits its uplink bandwidth equally among all receivers and streams to each receiver separately.

*The area chair coordinating the review of this manuscript and approving it for publication was Prof. Wei Tsang Ooi.
This work was partially supported by the General Research Fund grants (Project No. 411008, 411209, 411010, and 411011) and an Area of Excellence Grant (Project No. AoE/E-02/08), all established under the University Grant Committee of the Hong Kong SAR, China, as well as two gift grants from Microsoft and Cisco.

MM'11, November 28–December 1, 2011, Scottsdale, Arizona, USA.

Though simple to implement, Simulcast suffers from poor quality of service. Specifically, peers with low upload capacity are forced to use a low video rate that degrades the overall experience of the other peers.

In the academic literature, there are recently several studies on peer-to-peer (P2P) video conferencing from a utility maximization perspective [3–8]. Among them, Li *et al.* [3] and Chen *et al.* [4] may be the most related ones to this work (we call their unified approach Mutualcast). They have tried to support content distribution and multi-party video conferencing in multicast sessions, by maximizing aggregate application-specific utility and the utilization of node uplink bandwidth in P2P networks. Specific depth-1 and depth-2 tree topologies have been constructed using tree packing, and rate control was performed in each of the tree-based one-to-many sessions. However, they only considered the limited scenario where bandwidth bottlenecks reside at the edge of the network, while in practice bandwidth bottlenecks can easily reside in the core of the network [9, 10]. Further, all existing industrial and academic solutions, including Mutualcast, did not explicitly consider bounded delay in designs, and can lead to unsatisfied interactive conferencing experience.

1.1 Contribution

In this paper, we reconsider the design space in multi-party video conferencing solutions, and present *Celerity*, a new multi-party conferencing solution specifically designed to maintain low end-to-end delays while maximizing source coding rates in a session. *Celerity* has the following salient features:

- It operates in a pure P2P manner, and as such eliminating the cost of maintaining centrally administered servers.
- It can deliver video at quality levels commensurate with available network resources over *arbitrary network topologies*, while maintaining *bounded end-to-end delays*.
- It can automatically adapt to unpredictable network dynamics, such as cross traffic and abrupt link failures, allowing smooth conferencing experience.

Enabling the above features for multi-party conferencing is challenging. First, it requires a new formulation that allows systematic solution design over arbitrary network capacity constraints. In contrast, existing P2P system design works with performance guarantee commonly assume bandwidth bottlenecks reside at the edge of the network. Second, maximizing session rates subject to bounded delay is known to be NP-Complete and hard to solve approximately [11]. We take a practical approach in this paper that explores all 2-hop delay-bounded overlay trees with polynomial complexity. Third, detecting and reacting to network dynamics without *a priori* knowledge of the network conditions are non-trivial. We use both delay and loss as congestion measures and adapt the session rates with respect to both of them, allowing early detection and fast response to unpredictable network dynamics.

The highlight in our design is a distributed rate control protocol, that can discover and adapt to arbitrary topologies and network conditions quickly, converging to efficient link rate allocations allowed by the underlying network. In accordance with the adaptive link rate control, source video encoding rates are also dynamically controlled to optimize video quality in arbitrary and unpredictable underlay network conditions.

The design of *Celerity* is largely inspired by our new formulation that specifically takes into account arbitrary network capacity constraints and allows us to explore design space beyond those in existing solutions. Our formulation is overlay-link based and has a number of variables linear in the number of overlay links. This is a significant reduction as compared to the number of variables exponential in the number of overlay links in an otherwise tree-based formulation. We believe our approach is applicable to other P2P system problems, to allow solution design beyond the common assumption in P2P scenarios that the bandwidth bottlenecks reside only at the edge of the network.

We have implemented a prototype *Celerity* system using C++. By extensive experiments in a local experimental testbed and on the Internet, we demonstrate the superior performance of *Celerity* over state-of-the-art solutions Simulcast and Mutualcast.

Due to space limitation, all proofs and pseudo-codes are in our technical report [12].

Notation	Definition
$\mathcal{L}$	Set of all physical links
V	Set of conference participating nodes
E	Set of directed overlay links
C_l	Capacity of the physical link l
$a_{l,e}$	Whether overlay link e passes physical link l
$\boldsymbol{c}_m$	Overlay link rates of stream m, $\boldsymbol{c}_m = [c_{m,e}, e \in E]$
$\boldsymbol{y}$	Total overlay link traffics, $\boldsymbol{y} = \sum_{m=1}^{M} \boldsymbol{c}_m$
D	Delay bound
$R_m(\boldsymbol{c}_m, D)$	Session m's rate within the delay bound D
$q_l(z)$	Price function of violating link l's capacity constraint
p_l	Lagrange multiplier of link l's capacity constraint
$\mathcal{G}(\boldsymbol{c}, \boldsymbol{p})$	Lagrange function of variables $\boldsymbol{c}$ and $\boldsymbol{p}$

Note: we use bold symbols to denote vectors, e.g., $\boldsymbol{c} = [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_M]$.

Table 1: Key notations.

2. PROBLEM FORMULATION AND CELERITY OVERVIEW

One way to design a multi-party conferencing system is to formulate its fundamental design problem, explore powerful theoretical techniques to solve the problem, and use the obtained insights to guide practical system designs. In this way, we can also be clear about potential and limitation of the designs, allowing easy system tuning and further systematic improvements. Table 1 lists the key notations used in this paper.

2.1 Settings

Consider a network modeled as a directed graph $G = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of all physical nodes, including conference participating nodes and other intermediate nodes such as routers, and $\mathcal{L}$ is the set of all physical links. Each link $l \in \mathcal{L}$ has a nonnegative capacity C_l and a nonnegative propagation delay d_l.

Consider a multi-party conferencing system over G. We use $V \subseteq \mathcal{N}$ to denote the set of all conference participating nodes. Every node in V is a source and at the same time a receiver for every other nodes. Thus there are totally $M \triangleq |V|$ sessions of (audio/video) streams. Each stream is generated at a source node, say v, and needs to be delivered to all the rest nodes in $V - \{v\}$, by using overlay links between any two nodes in V.

An overlay link (u, v) means u can send data to v by setting up a TCP/UDP connection, along an underlay path from u to v preassigned by routing protocols. Let E be the set of all directed overlay links. For all $e \in E$ and $l \in \mathcal{L}$, we define

$$a_{l,e} = \begin{cases} 1, & \text{if overlay link } e \text{ passes physical link } l; \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The physical link capacity constraints are then expressed as

$$\boldsymbol{a}_l^T \boldsymbol{y} = \sum_{e \in E} a_{l,e} \sum_{m=1}^{M} c_{m,e} \leq C_l, \quad \forall l \in \mathcal{L},$$

where $c_{m,e}$ denotes the rate allocated to session m on overlay link e and $\boldsymbol{a}_l^T \boldsymbol{y}$ describes the total overlay traffic passing through physical link l.

Remark: In our model, the capacity bottleneck can be anywhere in the network, not necessarily at the edges. This is in contrast to a common assumption made in existing P2P works that the access links of participating nodes are the only capacity bottlenecks.

2.2 Problem Formulation

In a multi-party conferencing system, each session source broadcasts its stream to all receivers over a complete overlay graph on which every link e has a rate $c_{m,e}$ and a delay $\sum_{l \in \mathcal{L}} a_{l,e} d_l$. For smooth conferencing experience, the total delay of delivering a packet from the source to any receiver, traversing one or multiple overlay links, cannot exceed a delay bound D.

A fundamental design problem is to maximize the overall conferencing experience, by properly allocating the overlay link rates to the streams subject to physical link capacity constraints. We formulate the problem as a network utility maximization problem:

$$\textbf{MP}: \quad \max_{\boldsymbol{c} \geq 0} \quad \sum_{m=1}^{M} U_m(R_m(\boldsymbol{c}_m, D)) \tag{2}$$

$$\text{s.t.} \quad \boldsymbol{a}_l^T \boldsymbol{y} \leq C_l, \quad \forall l \in \mathcal{L}. \tag{3}$$

The optimization variables are $\boldsymbol{c}$ and the constraints in (3) are the physical link capacity constraints.

$R_m(\boldsymbol{c}_m, D)$ denotes session m's rate that we obtain by using resource $\boldsymbol{c}_m$ *within the delay bound* D, and is a concave function of $\boldsymbol{c}_m$ as we will show in Corollary 1 in the next section.

The objective is to maximize the aggregate system utility. $U_m(R_m)$ is an increasing and strictly concave function that maps the stream rate to an application-specific utility. For example, a commonly used video quality measure Peak Signal-to-Noise Ratio (PSNR) can be modeled by using a logarithmic function as the utility [4] [1]. With these settings and observations, $U_m(R_m)$ is concave in $\boldsymbol{c}$ and the problem **MP** is a concave optimization problem.

Remarks: (i) The formulation of **MP** is an overlay link based formulation in which the number of variables per session is $|E|$ and thus at most $|V|^2$. One can write an equivalent tree-based formulation for **MP** but the number of variables per session will be *exponential* in $|E|$ and $|V|$. (ii) Existing solutions, such as Simulcast and Mutualcast, can be thought as algorithms solving special cases of the problem **MP**. For example, Simulcast can be thought as solving the problem **MP** by using only the 1-hop tree to broadcast content within a session. Mutualcast can be thought as solving a special case of the problem **MP** (with the uplinks of participating nodes being the only capacity bottleneck) by packing certain depth-1 and depth-2 trees within a session.

2.3 Celerity Overview

Celerity builds upon two main modules to maximize the system utility: (1) a *delay-bounded video delivery* module to distribute video at high rate given overlay link rates (i.e., how to compute and achieve $R_m(\boldsymbol{c}_m, D)$); (2) a *link rate control* module to determine $\boldsymbol{c}_m$.

[1]Using logarithmic functions also guarantees (weighted) proportional fairness among sessions and thus no session will starve at the optimal solution [13].

Video delivery under known link constraints: This problem is similar to the classic multicast problem, and packing spanning (or Steiner) trees at the multicast source is a popular solution. However, the unique "delay-bounded" requirement in multi-party conferencing makes the problem more challenging. We introduce a delay-bounded tree packing algorithm to tackle this problem (detailed in Section 3).

Link rate control: In principle, one can first infer the network constraints and then solve the problem **MP** centrally. However, directly inferring the constraints potentially requires knowing the entire network topology and is highly challenging. In *Celerity*, we resort to design adaptive and iterative algorithms for solving the problem **MP** in a distributed manner, without *a priori* knowledge of the network conditions (detailed in Section 4).

We high-levelly explain how *Celerity* works in a 4-party conferencing example in Fig. 1. We focus on session A, in which source A distributes its stream to receivers B, C, and D, by packing delay-bounded trees over a complete overlay graph shown in the figure. We focus on source A and one overlay link (B, C), which represents a UDP connection over an underlay path B to E to F to C. Other overlay links and other sessions are similar.

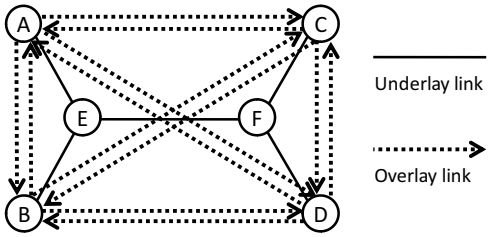


Figure 1: An illustrating example of 4-party (A, B, C, and D) conferencing over a dumbbell underlay topology. E and F are two routers. Solid lines represent underlay physical links. To make the graph easy to read, we use one solid line to represent a pair of directed physical links. Dash lines represent overlay links.

We first describe the control plane operations. For the overlay link (B, C), the head node B works with the tail node C to *periodically* adjust the session rate $c_{A,(B,C)}$ according to *Celerity*'s link rate control algorithm. Such adjustment utilizes control-plane information that source A piggybacks with data packets, and loss and delay statistics experienced by packets traveling from B to C. We show such local adjustments at every overlay link result in globally optimal session rates.

The head node B also *periodically* reports to source A the session rate $c_{A,(B,C)}$ and the end-to-end delay from B to C. Based on these reports from all overlay links, source A *periodically* packs delay-bounded trees using *Celerity*'s tree-packing algorithm, calculates necessary control-plane information, and delivers data and the control-plane information along the trees.

The data plane operations are simple. *Celerity* uses delay-bounded trees to distribute data in a session. Nodes on every tree forward packets from its upstream parent to its downstream children, following the "next-children" tree-routing information embedded in the packet header. *Celerity*'s tree-packing algorithm guarantees that (i) packets arrive at all receivers within the delay bound, and (ii) the total rate of a session m passing through an overlay link e does not exceed the allocated rate $c_{m,e}$.

In the following two sections, we first present the designs of the two main modules in *Celerity*. We then describe how they are implemented in physical peers in Section 5.

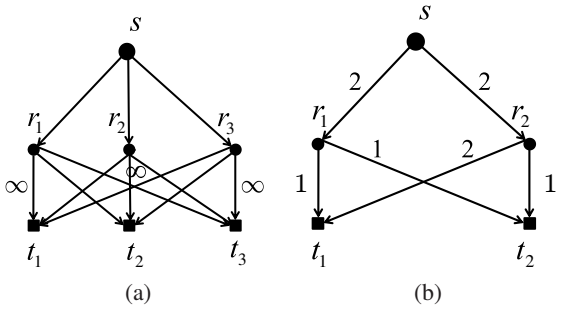


Figure 2: (a) Illustration of the directed acyclic sub-graph over which we pack delay-bounded 2-hop trees. (b) Critical cut example. Source s and its two receivers t_1, t_2 are connected over a directed graph. The number associated with a link represents its capacity.

3. PACKING DELAY-BOUNDED TREES

Given overlay link rate vector $\boldsymbol{c}_m$ and delay for every overlay link e (i.e., $\sum_{l\in\mathcal{L}} a_{l,e}d_l$), achieving the maximum broadcast/multicast stream rate under a delay bound D is a challenging problem. A general way to explore the broadcast/multicast rate under delay bounds is to pack delay-bounded Steiner trees. However, such problem is *NP*-hard [14]. Moreover, the number of delay-bounded Steiner trees to consider is in general exponential in the network size.

In this paper, we pack 2-hop delay-bounded trees in an overlay graph of session m, denoted by $\mathcal{D}_m$, to achieve a good stream rate under a delay bound. Note by graph theory notations, a 2-hop tree has a depth at most 2. Packing 2-hop trees is easy to implement. It also explores all overlay links between source and receiver and between receivers, thus trying to utilize resource efficiently. In fact, it is shown in [3, 4] that packing 2-hop multicast trees suffices to achieve the maximum multicast rate for certain P2P topologies. We elaborate our tree-packing scheme in the following.

We first define the overlay graph $\mathcal{D}_m$. Graph $\mathcal{D}_m$ is a directed acyclic graph with two layers; one example of such graph is illustrated in Fig. 2a. In this example, consider a session with a source s, three receivers $1, 2, 3$. For each receiver i, we draw two nodes, r_i and t_i, in the graph $\mathcal{D}_m$; t_i models the receiving functionality of node i and r_i models the relaying functionality of node i.

Suppose that the prescribed link bit rates are given by the vector $\boldsymbol{c}_m$, with the capacity for link (i, j) being $c_{m,(i,j)}$. Then in $\mathcal{D}_m$, the link from s to r_i has capacity $c_{m,(s,i)}$, the link from r_i to t_j (with $i \neq j$) has capacity $c_{m,(i,j)}$, and the link from r_i to t_i has infinite capacity. If the propagation delay of an edge (i, j) exceeds the delay bound, we do not include it in the graph. If the propagation delay of a two-hop path $s \to i \to j$ exceeds the delay bound, we omit the edge from r_i to t_j from the graph. As a result, every path from s to any receiver t_i in the graph has a path propagation delay within the delay bound.

Over such 2-layer sub-graph $\mathcal{D}_m$, we pack 2-hop trees connecting the source and every receiver using the greedy algorithm proposed in [15]. Below we simply describe the algorithm and more details can be found in [15].

Assuming all edges have unit-capacity and allowing multiple edges for each ordered node pair. The algorithm packs unit-capacity trees one by one. Each unit-capacity tree is constructed by greedily constructing a tree edge by edge starting from the source and augmenting towards all receivers. It is similar to the greedy tree-packing algorithm based on Prim's algorithm. The distinction lies in the rule of selecting the edge among all potential edges. The edge whose removal leads to least reduction in the multicast capacity of the residual graph is chosen in the greedy algorithm. We show an example to illustrate how the tree packing algorithm works in details in our technical report [12, Sec. 3].

The above greedy algorithm is very simple to implement and its practical implementation details are further discussed in Section 5. Utilizing the special structure of the graph $\mathcal{D}_m$, we obtain performance guarantee of the algorithm as follows.

Theorem 1. *The tree-packing algorithm in [15] achieves the minimum of the min-cuts separating the source and receivers in $\mathcal{D}_m$ and is expressed as*

$$R_m(\boldsymbol{c}_m, D) = \min_j \sum_i \min\{c_{m,(s,i)}, c_{m,(i,j)}\}. \tag{4}$$

Furthermore, the algorithm has a running time of $O(|V||E|^2)$.

Hence, our tree-packing algorithm achieves the maximum delay-bounded multicast rate over the 2-layer subgraph $\mathcal{D}_m$. The achieved rate $R_m(\boldsymbol{c}_m, D)$ is a concave function of $\boldsymbol{c}_m$ as summarized below.

Corollary 1. *The delay-bounded multicast rate $R_m(\boldsymbol{c}_m, D)$ obtained by our tree-packing algorithm is a concave function of the overlay link rates $\boldsymbol{c}_m$.*

4. OVERLAY LINK RATE CONTROL

4.1 Considering Both Delay and Loss

We revise original formulation to design our link rate control algorithm with both queuing delay and loss rate taken into account. Adapting link rates to both delay and loss allows early detection and fast response to network dynamics.

Consider the following formulation with a penalty term added into the objective function of the problem **MP**:

$$\mathbf{MP-EQ}: \max_{\boldsymbol{c}\geq 0} \quad \mathcal{U}(\boldsymbol{c}) \triangleq \sum_{m=1}^{M} U_m\left(R_m(\boldsymbol{c}_m, D)\right) - \sum_{l\in\mathcal{L}} \int_0^{\boldsymbol{a}_l^T\boldsymbol{y}} q_l(z)\,dz \tag{5}$$

$$\text{s.t.} \qquad \boldsymbol{a}_l^T\boldsymbol{y} \leq C_l, \quad \forall l \in \mathcal{L}, \tag{6}$$

where $\int_0^{\boldsymbol{a}_l^T\boldsymbol{y}} q_l(z)\,dz$ is the penalty associated with violating the capacity constraint of physical link $l \in \mathcal{L}$, and we choose

$$q_l(z) \triangleq \frac{(z - C_l)^+}{z}, \tag{7}$$

where $(a)^+ = \max\{a, 0\}$. If all the constraints are satisfied, then the second term in (5) vanishes; if instead some constraints are violated, then we charge some penalty for doing so.

Remark: (i) The problem **MP-EQ** is equivalent to the original problem **MP**. Because any feasible solution $\boldsymbol{c}$ of these two problems must satisfy $\boldsymbol{a}_l^T\boldsymbol{y} \leq C_l$, and consequently the penalty term in the problem **MP-EQ** vanishes. Therefore, any optimal solution of the original problem **MP** must be an optimal solution of the problem **MP-EQ** and vice versa. (ii) It can be verified that $-\sum_{l\in\mathcal{L}} \int_0^{\boldsymbol{a}_l^T\boldsymbol{y}} q_l(z)\,dz$ is a concave function in $\boldsymbol{c}$; hence, $\mathcal{U}(\boldsymbol{c})$ is a linear combination of concave functions and is concave. However, because $R_m(\boldsymbol{c}_m, D)$ is the minimum min-cut of the overlay graph $\mathcal{D}_m$ with link rates being $\boldsymbol{c}_m$, $\mathcal{U}(\boldsymbol{c})$ is not a differentiable function [16].

We apply Lagrange dual approach to design distributed algorithms for the problem **MP-EQ**. The advantage of adopting distributed rate control algorithms in our system is that it allows robust adaption upon unpredictable network dynamics.

The Lagrange function of the problem is given by:

$$\mathcal{G}(\boldsymbol{c},\boldsymbol{p}) \triangleq \sum_{m=1}^{M} U_m\left(R_m(\boldsymbol{c}_m,D)\right) - \sum_{l\in\mathcal{L}} \int_0^{\boldsymbol{a}_l^T\boldsymbol{y}} q_l(z)\,dz - \sum_{l\in\mathcal{L}} p_l\left(\boldsymbol{a}_l^T\boldsymbol{y} - C_l\right), \quad (8)$$

where $p_l \geq 0$ is the Lagrange multiplier associated with the capacity constraint in (6) of physical link l. p_l can be interpreted as the price of using link l. Since the problem **MP-EQ** is a concave optimization problem with linear constraints, strong duality holds and there is no duality gap. Any optimal solution of the problem and one of its corresponding Lagrangian multiplier is a saddle point of $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$ and vice versa. Thus to solve the problem **MP-EQ**, it suffices to design algorithms to pursue saddle points of $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$.

4.2 A Loss-Delay Based Primal-Subgradient-Dual Algorithm

There are two issues to address in designing algorithms for pursuing saddle points of $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$. First, the utility function $\mathcal{U}(\boldsymbol{c})$ (and consequently $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$) is not everywhere differentiable. Second, $\mathcal{U}(\boldsymbol{c})$ (and consequently $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$) is not strictly concave in $\boldsymbol{c}$, thus distributed algorithms may not converge to the desired saddle points under multi-party conferencing settings [4].

To address the first concern, we use subgradient in algorithm design. To address the second concern, we provide a convergence result for our designed algorithm.

To proceed, we first compute subgradients of $\mathcal{U}(\boldsymbol{c})$. The proposition below presents a useful observation.

Proposition 1. *A subgradient of $\mathcal{U}(\boldsymbol{c})$ with respect to $c_{m,e}$ for any $e \in E$ and $m = 1,\ldots M$ is given by*

$$U_m'(R_m)\frac{\partial R_m}{\partial c_{m,e}} - \sum_{l\in\mathcal{L}} a_{l,e}\frac{(\boldsymbol{a}_l^T\boldsymbol{y} - C_l)^+}{\boldsymbol{a}_l^T\boldsymbol{y}}$$

where $\frac{\partial R_m}{\partial c_{m,e}}$ is a subgradient of $R_m(\boldsymbol{c}_m, D)$ with respect to $c_{m,e}$.

Motivated by the pioneering work of Arrow, Hurwicz, and Uzawa [17] and the followup works [18] [19], we propose to use the following *primal-subgradient-dual* algorithm to pursue the saddle point of $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$: $\forall e\epsilon M$, $m = 1,...M$, $\forall l\epsilon\mathcal{L}$,

Primal-Subgradient-Dual Link Rate Control Algorithm:

$$c_{m,e}^{(k+1)} = c_{m,e}^{(k)} + \alpha\left[U_m'\left(R_m^{(k)}\right)\frac{\partial R_m^{(k)}}{\partial c_{m,e}} \sum_{l\in\mathcal{L}} a_{l,e}\frac{(\boldsymbol{a}_l^T\boldsymbol{y}^{(k)} - C_l)^+}{\boldsymbol{a}_l^T\boldsymbol{y}^{(k)}} - \sum_{l\in\mathcal{L}} a_{l,e}p_l^{(k)}\right]_{c_{m,e}^{(k)}}^{+} \quad (9)$$

$$p_l^{(k+1)} = p_l^{(k)} + \frac{1}{C_l}\left[\boldsymbol{a}_l^T\boldsymbol{y}^{(k)} - C_l\right]_{p_l^{(k)}}^{+} \quad (10)$$

where $\alpha > 0$ represents a constant step size for all iterations, and

$$[b]_a^+ = \begin{cases}\max(0,b), & a \leq 0;\\ b, & a > 0.\end{cases}$$

We have the following observations for the algorithm in (9)-(10):

- It is known that $\sum_{l\in\mathcal{L}} a_{l,e}\frac{(\boldsymbol{a}_l^T\boldsymbol{y}-C_l)^+}{\boldsymbol{a}_l^T\boldsymbol{z}}$ can be interpreted as the packet loss rate observed at overlay link e [20]. The intuitive explanation is as follows. The term $(\boldsymbol{a}_l^T\boldsymbol{y} - C_l)^+$ is the excess traffic rate offered to physical link l; thus $\frac{(\boldsymbol{a}_l^T\boldsymbol{y}-C_l)^+}{\boldsymbol{a}_l^T\boldsymbol{y}}$ models the fraction of traffic that is dropped at l. Assuming the packet loss rates are additive (which is a reasonable assumption for low packet loss rates), the total packet loss rates seen by the overlay link e is given by $\sum_{l\in\mathcal{L}} a_{l,e}\frac{(\boldsymbol{a}_l^T\boldsymbol{y}-C_l)^+}{\boldsymbol{a}_l^T\boldsymbol{y}}$.
- It is also known that p_l updating according to (10) can be interpreted as queuing delay at physical link l [21]. Intuitively, if the incoming rate $\boldsymbol{a}_l^T\boldsymbol{y} > C_l$, then it introduces an additional queuing delay of $\frac{\boldsymbol{a}_l^T\boldsymbol{y}-C_l}{C_l}$ at link l. If otherwise $\boldsymbol{a}_l^T\boldsymbol{y} \leq C_l$, then the queueing delay is reduced by an amount of $\frac{C_l-\boldsymbol{a}_l^T\boldsymbol{y}}{C_l}$ unless hitting zero. The total queuing delay observed by the overlay link e is simply the sum $\sum_{l\in\mathcal{L}} a_{l,e}p_l$.
- It turns out that the utility function, the subgradients, packet loss rate and queuing delay are sufficient statistics to update $c_{m,e}$ independently of the updates of other link rates. This way, we can solve the problem **MP-EQ** without knowing the physical network topology and physical link capacities.

The algorithm in (9)-(10) is similar to the standard primal-dual algorithm, but since $\mathcal{U}(\boldsymbol{c})$ is not differentiable everywhere, we use subgradient instead of gradient in updating the overlay link rates $\boldsymbol{c}$. If we fix the dual variables $\boldsymbol{p}$, then the algorithm in (9) corresponds to the standard subgradient method [22]. It maximizes a non-differentiable function in a way similar to gradient methods for differentiable functions — in each step, the variables are updated in the direction of a subgradient. However, such a direction may not be an ascent direction; instead, the subgradient method relies on a different property. If the variable takes a sufficiently small step along the direction of a subgradient, then the new point is closer to the set of optimal solutions.

Establishing convergence of subgradient algorithms for saddle-point optimization is in general challenging [18]. We explore convergence properties for our primal-subgradient-dual algorithm in the following theorem.

Theorem 2. *Let $(\boldsymbol{c}^*,\boldsymbol{p}^*)$ be a saddle point of $\mathcal{G}(\boldsymbol{c},\boldsymbol{p})$, and $\bar{\mathcal{G}}^{(k)}$ be the average function value obtained by the algorithm in (9)-(10) after k iterations:*

$$\bar{\mathcal{G}}^{(k)} \triangleq \frac{1}{k}\sum_{i=0}^{k-1}\mathcal{G}\left(\boldsymbol{c}^{(k)},\boldsymbol{p}^{(k)}\right).$$

Suppose $\left|U_m'(R_m(\boldsymbol{c}_m))\right| \leq \bar{U}$, $\forall m = 1,\ldots,M$, where $\bar{U}$ is a constant, then we have

$$-\frac{B_1}{2\alpha k} - \frac{\Delta^2}{2}\alpha \leq \bar{\mathcal{G}}^{(k)} - \mathcal{G}(\boldsymbol{c}^*,\boldsymbol{p}^*) \leq \frac{B_2}{2k} + \frac{\Delta^2}{2}\max_{l\in\mathcal{L}} C_l^{-1},$$

where $B_1 = \left\|\boldsymbol{c}^{(0)} - \boldsymbol{c}^*\right\|^2$ and $B_2 = \left[\boldsymbol{p}^{(0)} - \boldsymbol{p}^*\right]^T diag\,(C_l, l\in\mathcal{L})\left[\boldsymbol{p}^{(0)} - \boldsymbol{p}^*\right]$ are two positive distances depending on $(\boldsymbol{c}^{(0)},\boldsymbol{p}^{(0)})$, and Δ is a positive constant depending on $\bar{U}$ and $(\boldsymbol{c}^{(0)},\boldsymbol{p}^{(0)})$.

Remarks: (i) The results bound the time-average Lagrange function value obtained by the algorithm to the optimal in terms of distances of the initial iterates $(\boldsymbol{c}^{(0)},\boldsymbol{p}^{(0)})$ to a saddle point. In particular, the averaged function values $\bar{\mathcal{G}}^{(k)}$ converge to the saddle point value $\mathcal{G}(\boldsymbol{c}^*,\boldsymbol{p}^*)$ within a gap of $\max\left(\alpha, \max_{l\in\mathcal{L}} C_l^{-1}\right)\frac{\Delta^2}{2}$, at a rate of $1/k$. (ii) The requirement of the utility function is easy to satisfied; one example is $U_m(z) = \log(z+\epsilon)$ with $\epsilon > 0$. (iii) Our results generalize the one in [18] in the sense that the result in [18] only applies to the case of uniform step size, while we allow different p_l to update with different step size $\frac{1}{C_l}$, which is critical for p_l to be interpreted as queuing delay and thus practically measurable. Our results also have less stringent requirement on the utility function

than the one in [18]. (iv) Although the results may not warranty convergence in the strict sense, our experiments over LAN testbed and on the Internet in Section 6 show the algorithm quickly stabilizes around optimal operating points. Obtaining stronger convergence results that confirm our practical observations are of great interests and is left for future work.

4.3 Computing Subgradients of $R_m(\boldsymbol{c}_m, D)$

A key to implementing the Primal-Subgradient-Dual algorithm is to obtain subgradients of $R_m(\boldsymbol{c}_m, D)$. We first present some preliminaries on subgradients, as well as concepts for computing subgradients for $R_m(\boldsymbol{c}_m, D)$.

Definition 1. *Given a convex function f, a vector ξ is said to be a subgradient of f at $x \in \mathbf{dom} f$ if*

$$f(x') \geq f(x) + \xi^T(x' - x), \forall x' \in \mathbf{dom} f,$$

where $\mathbf{dom} f = \{x \in \mathbf{R}^n | |f(x)| < \infty\}$ is the domain of f.

For a concave function f, $-f$ is a convex function. A vector ξ is said to be a subgradient of f at x if $-\xi$ is a subgradient of $-f$.

Next, we define the notion of a *critical cut*. For session m, let its source be s_m and receiver set be $V_m \subset V - \{s_m\}$. A partition of the vertex set, $V = Z \cup \bar{Z}$ with $s_m \in Z$ and $t \in \bar{Z}$ for some $t \in V_m$, determines an s_m-t-cut. Define

$$\delta(Z) \triangleq \left\{(i, j) \in E | i \in Z, j \in \bar{Z}\right\}$$

be the set of overlay links originating from nodes in set Z and going into nodes in set $\bar{Z}$. Define the capacity of cut $(Z, \bar{Z})$ as the sum capacity of the links in $\delta(Z)$:

$$\rho(Z) \triangleq \sum_{e \in \delta(Z)} c_{m,e}.$$

Definition 2. *For session m, a cut $(Z, \bar{Z})$ is an s_m-V_m critical cut if it separates s_m and any of its receivers and $\rho(Z) = R_m(\boldsymbol{c}_m, D)$.*

We show an example to illustrate the concept of critical cut. In Fig. 2b, s is a source, and t_1, t_2 are its two receivers. The minimum of the min-cuts among the receivers is 2. For the cut $(\{s, r_1, r_2, t_1\}, \{t_2\})$, its $\delta(\{s, r_1, r_2, t_1\})$ contains links (r_1, t_2) and (r_2, t_2), each having capacity one. Thus the cut $(\{s, r_1, r_2, t_1\}, \{t_2\})$ has a capacity of 2 and it is an $s - (t_1, t_2)$ critical cut.

With necessary preliminaries, we turn to compute subgradients of $R_m(\boldsymbol{c}_m, D)$. Since $R_m(\boldsymbol{c}_m, D)$ is the minimum min-cut of s_m and its receivers over the overlay graph $\mathcal{D}_m$, it is known that one of its subgradients can be computed in the following way [16].

- Find an s_m-V_m critical cut for session m, denote it as $(Z, \bar{Z})$. Note there can be multiple s_m-V_m critical cuts in graph $\mathcal{D}_m$, and it is sufficient to find any one of them.

- A subgradient of $R_m(\boldsymbol{c}_m, D)$ with respect to $c_{m,e}$ is given by

$$\frac{\partial R_m(\boldsymbol{c}_m, D)}{\partial c_{m,e}} = \begin{cases} 1, & \text{if } e \in \delta(Z); \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

5. PRACTICAL IMPLEMENTATION

Using the asynchronous networking paradigm supported by the asynchronous I/O library (called `asio`) in the `Boost` C++ library, we have implemented a prototype of *Celerity* with about 17,000 lines of code in C++.

Celerity consists of three main functional components: link rate control module, tree-packing and critical cut calculation module, and data multicast engine. Fig. 3 describes the relationship between these components and where they physically reside.

In the following, we describe the functionality implemented by peers, some critical implementations, and operation overhead.

5.1 Peer Functionality

In our implementation, all peers perform the following functions:

- Peers in every broadcast tree forward packets from its upstream parent to its downstream children. Sufficient information about the children is embedded in the packet header, for a packet to become "self-routing" from the source to all leaf nodes in a tree.

- Every 200 ms, each peer calculates the loss rate and queuing delay of its incoming links and updates the rates of these links based on the link rate control algorithm. The peer then sends the updated rates to the corresponding senders to take effect.

- Every 300 ms, each peer sends the link state (including allocated rate and Round Trip Time) of all its outgoing links for each session to the session source.

Upon receiving link states from all the links, the source of each session uses the received link rates and the delay information to pack a new set of delay-bounded trees, and starts transmitting session packets along these trees. We set the delay bound to be 200 ms when packing delay-bounded trees in our implementation. When a source packs delay-bounded trees, it also calculates one critical cut and the derivative of the utility for its session based on the allocated link rates. The source embeds the information about the critical cut and the derivative of the utility in the header of outgoing packets. When these packets are received, a peer learns the derivative of the utility and whether a link belongs to the critical cut or not; it then adjusts the link rate accordingly.

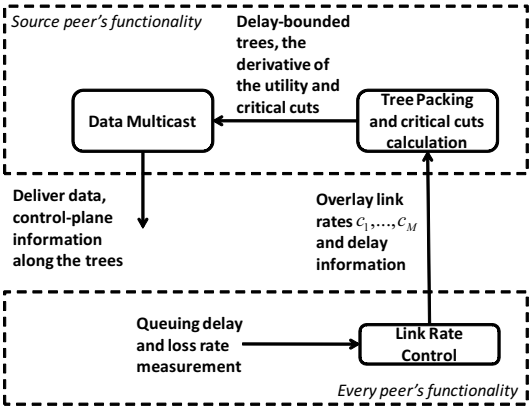


Figure 3: System architecture of *Celerity*.

5.2 Critical Cut Calculation

The calculation of critical cuts, i.e., the subgradient of $R_m(\boldsymbol{c}_m, D)$, is by the source of each session. It is the key to our implementation of the primal-subgradient-dual algorithm. There can be multiple critical cuts in one session, but it is sufficient to find any one of them. After the source collects the allocated session rates from all overlay links, it calculates the min-cut from itself to every receiver, and records the min-cut. Then the source compares the capacities of these min-cuts; the cut with the smallest capacity is a critical cut.

5.3 Utility Function

With respect to the utility function in our implementation, the PSNR (peak signal-to-noise ratio) metric is the de facto standard criterion to provide objective quality evaluation in video processing. The PSNR of a video stream coded at a rate z can be approximated by a logarithmic function $\beta \log(z + \delta)$ [4], in which a higher β represents videos with a larger amount of motion. δ is a small positive constant to ensure the function has a bounded derivative for $z \geq 0$. Due to this observation, we use a logarithmic utility function in our implementation.

5.4 Opportunistic Local Loss Recovery

Providing effective loss recovery in a delay-bounded reliable broadcast scenario, such as multi-party conferencing, is known to be challenging [23]. It is hard for error control coding to work efficiently, since different receivers in a session may experience different loss rates and thus choosing proper coding parameters to avoid unnecessary waste of throughput is non-trivial. If re-broadcasting the lost-packets is in use, it introduces additional delay and may cause packets missing deadlines and become useless.

In our implementation, we use network coding [23] [24] to allow opportunistic local loss recovery. For each overlay link e, if the trees of a session m do not exhaust $c_{m,e}$, the overlay-link rate dedicated for the session, then we send coded packets (i.e., linear combination of received packets of corresponding session) over such link e. As such, receiver of the overlay link e can recover the packets that are lost on link e locally by using the network coded packets. This way, Celerity provides certain flexible local loss recovery capability without incurring delay due to retransmission.

5.5 Fast Bootstrapping

Similar to TCP's Slow Start strategy, we implement a method in *Celerity* called "quick start" to quickly ramp up the session rates during conference initialization stage. The purpose is to bootstrap the system to close-to-optimal operating points when the conference just starts, during which period peers are joining the conference and nothing significant is going on. We achieve this by using large β in the utility functions and large step sizes in link rate adaptation during the first 30 seconds. After the initialization stage, we reset β and the step sizes to proper values and allow our system converge gradually, avoiding unnecessary performance fluctuation.

5.6 Operation Overhead

There are two types of overhead in *Celerity*: (i) *packet overhead*: each application-layer packet has a 46-byte header, containing critical cut information, derivative of the utility, packet sequence number, coding vector, and timestamp. (ii) *link-rate control and link-state report overhead*: every 200 ms, each peer adjusts the rates of its incoming links and sends them to their corresponding upstream senders. In our implementation, such rate-control overhead is 0.2 kbps per link per session. For the link state report overhead, each peer sends the link state of all its outgoing links for each session to the source of the session every 300 ms. In our implementation, for each peer, such link-state report overhead is 0.158 kbps per link per session. In Section 6.2, we report an overall operational overhead of 3.9% in our 4-party Internet experiment.

6. EXPERIMENTS

We evaluate our prototype *Celerity* system over a LAN testbed as well as over the Internet. The LAN experiments allow us to (i) stress-test *Celerity* under various network conditions; (ii) see whether *Celerity* meets the design goal – delivering high delay-bounded throughput and automatically adapting to dynamics in the network; (iii) demonstrate the fundamental performance gains over existing solutions, thus justifying our theory-inspired design.

The Internet experiments allow us to further access *Celerity*'s superior performance over existing solutions in the real world.

6.1 LAN Testbed Experiments

We evaluate *Celerity* over a LAN testbed illustrated in Fig. 4, where four PC nodes (A, B, C, D) communicate over a LAN dumbbell topology. The dumbbell topology represents a popular scenario of multi-party conferencing between branch offices. It is also a "tough" topology – existing approaches, such as Simulcast and Mutualcast, fail to efficiently utilize the bottleneck bandwidth and optimize system performance.

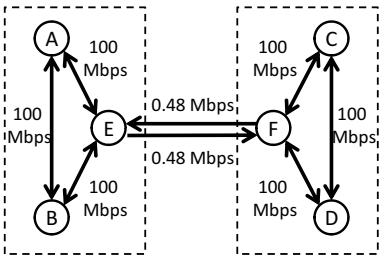


Figure 4: The "tough" dumbbell topology of the experimental testbed. Two conference participating nodes A and B are in one "office" and another twos nodes C and D are in a different "office". The two "offices" are connected by directed links between gateway nodes E and F, each link has a capacity of 0.48 Mbps. Link propagation delays are negligible.

In our experiments, all four nodes run *Celerity*. We run a four-party conference for 1000 seconds and evaluate the system performance. In order to evaluate *Celerity*'s performance in the presence of network dynamics, we reduce cross traffic and introduce link failures during the experiment. In particular, we introduce an 80kpbs cross-traffic from node E to node F between the 300th second and the 500th second, reducing the available bandwidth between E and F from 480 kbps to 400 kbps. Further, starting from the 700th second, we disconnect the physical link between A and E; this corresponds to a practical situation where node A suddenly cannot directly communicate with nodes outside the "office" due to middleware or configuration errors at the gateway E.

Figs. 6a-6d show the sending rate of each session (one session originates from one node to all other three nodes). For comparison, we also show the maximum achievable rates by Simulcast and Mutualcast, as well as the optimal sending rate of each session calculated by solving the problem in (2)-(3) using a central solver. Fig. 6e shows the utility obtained by *Celerity* and its comparison to the optimal. Fig. 6f shows the average end-to-end delay and packet loss rate of session A. Delay and loss performance of other sessions are similar to those of session A.

In the following, we explain the results according to three different experiment stages.

6.1.1 Absence of Network Dynamics

We first look at the first 300 seconds when there is no cross traffic or link failure. In this time period, the experimental settings are symmetric for all participating peers; thus the optimal sending rate for each session is 240 kbps.

As seen in Figs. 6a-6d, *Celerity* demonstrates fast convergence: the sending rate of each session quickly ramps up to 95% to the optimal within 50 seconds. Fig. 6e shows that *Celerity* quickly

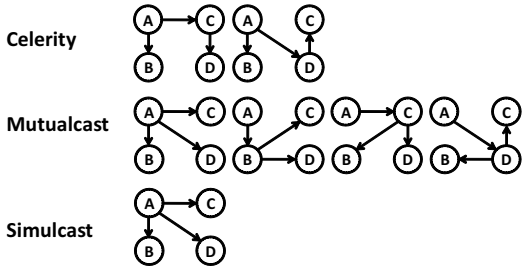


Figure 5: Session A's trees used by *Celerity* (upon convergence), Mutualcast and Simulcast in the dumbbell topology, in the absence of network dynamics.

achieves a close-to-optimal utility. These observations indicate any other solution can at most outperform *Celerity* by a small margin.

As a comparison, we also plot the theoretical maximum rates achievable by Simulcast and Mutualcast in Figs. 6a-6d. We observe that within 20 seconds, our system already outperforms the maximum rates of Simulcast and Mutualcast.

Upon convergence, *Celerity* achieves sending rates that nearly double the maximum rate achievable by Simulcast and Mutualcast. This significant gain is due to that *Celerity* can utilize the bottleneck resource more efficiently, as explained below.

In Fig. 5, we show the trees for session A that are used by *Celerity*, Mutualcast and Simulcast in the dumbbell topology. As seen, Simulcast and Mutualcast only explore 2-hop trees satisfying certain structure, limiting their capability of utilizing network capacity efficiently. In particular, their trees consumes the bottleneck link resource twice, thus to deliver one-bit of information it consumes two-bit of bottleneck link capacity. For instance, the tree used by Simulcast has two branches $A \rightarrow C$ and $A \rightarrow D$ passing through the bottleneck links between E and F, consuming twice the critical resource. Consequently, the maximum achievable rates of Simulcast and Mutualcast are all 120 kbps. In contrast, *Celerity* explores all 2-hop delay-bounded trees, and upon convergence utilizes the trees that only consume bottleneck link bandwidth once, achieving rates that are close to the optimal of 240 kbps.

Fig. 6f shows the average end-to-end delay and packet loss rate of session A. As seen, the loss rate and delay are high initially, but decrease and stabilize to small values afterwards. The high loss rate and delay are due to *Celerity* increasing the rates aggressively during the initialization stage, in order to bootstrap the conference and explore network resource limits. *Celerity* quickly learns and adapts to the network topology, ending up with using cost-effective trees to deliver data. After the initialization stage, *Celerity* adapts and converges gradually, avoiding unnecessary performance fluctuation that deteriorates user experience. By adapting to both delay and loss, we achieve low loss rate upon convergence as compared to the case when only loss is taken into account [25].

6.1.2 Cross Traffic

Between the 300th second and the 500th second, we introduce an 80kpbs cross-traffic from node E to node F. Consequently, the available bottleneck bandwidth between E and F decreases from 480 kbps to 400 kbps. We calculate the optimal sending rates during this time period to be 200 kbps for sessions A and B, and remain 240 kbps for sessions C and D.

Seen in Figs. 6a-6d, *Celerity* quickly adapts to the bottleneck bandwidth reduction. *Celerity*'s adaptation is expected from its design, which infers from loss and delay the available resource and adapt accordingly. From Fig. 6f, we can see a spike in session A's packet loss rate around 300th second, at which time the available bottleneck bandwidth reduces. The link rate control modules in *Celerity* senses this increased loss rate, adjusts, and reports the reduced (overlay) link rates to node A. Upon receiving the reports, the tree-packing module in *Celerity* adjusts the source sending rate accordingly, adapting the system to a new close-to-optimal operating point. At 500th second, the cross traffic is removed and the available bottleneck bandwidth between E and F restores to 480kbps. *Celerity* also quickly learns this change and adapts to operate at the original point, evident in Figs. 6a-6b.

6.1.3 Link Failure

Between the 700th second and the 1000th second, we disconnect the physical link between A and E. Consequently, node A cannot use the 2-hop threes with node C (D) being intermediate nodes; similarly node C (D) cannot use the 2-hop threes with node A being intermediate nodes. They can, however, still use the trees with node B as intermediate nodes. We compute the theoretical optimal sending rates during this time period to be 240 kbps for all sessions.

We observe from Fig. 6a that node A's sending rate first drops immediately upon link failure, then quickly adapts to the new operating point of around 120 kbps, only half of the theoretical optimal. This is because *Celerity* only explores 2-hop trees for content delivery, while in this case 3-hop trees (e.g., $A \rightarrow B \rightarrow C \rightarrow D$) are needed to achieve the optimal. It is of great interest to explore beyond this 2-hop tree-packing limitation to further improve the performance without incurring excessive overhead.

In Figs. 6d, we observe session D's rate first drops and then climbs back. This is because session D *happens* to use trees with node A being an intermediate node right before the link failure. The failure breaks these trees, and causes session D's rate to drop dramatically. *Celerity* adapts and switches to use trees with node B as the intermediate node. Consequently, session D's rate gradually restores to around the optimal. This observation shows the excellent adaptability of *Celerity* to abrupt network condition changes.

As a comparison, we observe that Simulcast's maximum achievable rates of session A, C, and D all drop to zero upon the link failure. This is because there is no direct overlay link between A and C (D) after the link failure. Consequently, Simulcast is not able to broadcast the source's content to all the receivers in these sessions, resulting in zero session rates.

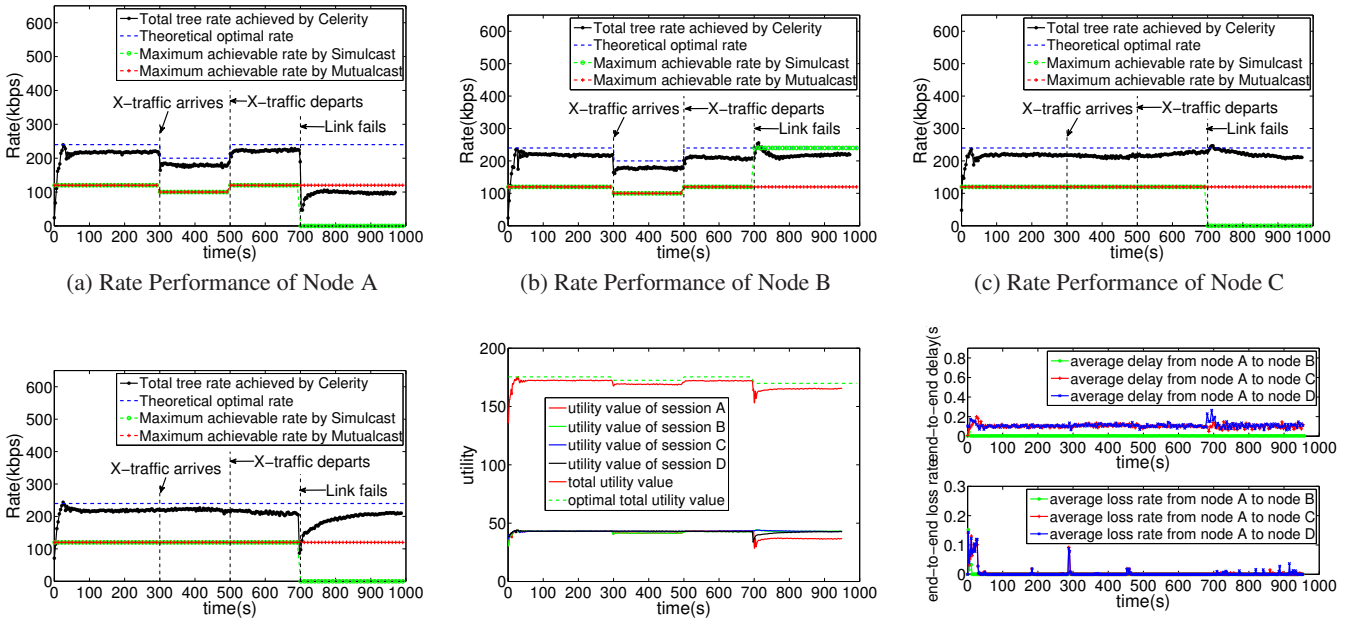
6.2 Internet Experiments

Beside the prototype *Celerity* system, we also implement two prototype systems of Simulcast and Mutualcast, respectively. Both *Celerity* and Mutualcast use the same log utility functions in their rate control modules. We evaluate the performance of these systems in a four-party conferencing scenario over the Internet.

We use four PC nodes that spread two continents and tree countries to form the conferencing scenario. Two of the PC nodes are in Hong Kong, one is in Redmond, Washington, US, and the last one is in Toronto, Canada. This setting represents a common global multi-party conferencing scenario.

We run multiple 15-minute four-party conferences using the prototype systems, in a one-by-one and interleaving manner. We select one representative run for each system, and summarize their performance in Fig. 7.

Figs. 7a-7d show the rate performance of each session. As seen, all the session rates in *Celerity* quickly ramp up to near-stable values within 50 seconds, and outperforms Simulcast within 10 seconds. Upon stabilization, *Celerity* achieves the best throughput performance among the three systems and Simulcast is the worst. For instance, all the session rates in *Celerity* is 2x of those in Simulcast



(a) Rate Performance of Node A

(b) Rate Performance of Node B

(c) Rate Performance of Node C

(d) Rate Performance of Node D

(e) Total utility of all sessions

(f) Average end-to-end delay and loss rate from node A to other nodes

Figure 6: Performance of *Celerity* over a dumbbell LAN testbed. (a)-(d): Sending rates and receiving rates of individual sessions. (e): Utility value achieved compared to the optimum. (f): End-to-end delay and loss rate of session *A*.

and Mutualcast, except in session C where Mutualcast is able to achieve a higher rate than *Celerity*.

We further observe *Celerity*'s superior performance in Fig. 7e, which shows the aggregate session rates, and in Fig. 7f, which shows the total achieved utilities. In both statistics, *Celerity* outperforms the other two systems by a significant margin. Specifically, the aggregate session rate achieved by *Celerity* is 2.5x of that achieved by Simulcast, and is 1.8x of that achieved by Mutualcast.

These results show that our theory-inspired solution *Celerity* can allocate the available network resource to best optimize the system performance. Mutualcast aims at similar objective but only works the best in scenarios where bandwidth bottlenecks reside only at the edge of the network [4].

Figs. 7g-7i show the average end-to-end loss rate and delay from source to receivers for session *A*, session *C* and session *D*. The results for session *B* is very similar to session *A* and is not included here. As seen, the average end-to-end delays of all sessions are within 200 ms, which is our preset delay bound for effective interactive conferencing experience. The average end-to-end loss rate for all sessions are at most 1%-2% upon system stabilization.

The overall operation overhead of *Celerity* in the 4-party Internet experiment is around 3.9%. In particular, the packet overhead accounts for 3.4%, and the link-rate control and link-state report overhead is around 0.5%.

7. CONCLUDING REMARKS

With the proliferation of front-facing cameras on mobile devices, multi-party video conferencing will soon become an utility that both businesses and consumers would find useful. With *Celerity,* we attempt to bridge the long-standing gap between the bit rate of a video source and the highest possible delay-bounded broadcasting rate that can be accommodated by the Internet where *the bandwidth bottlenecks can be anywhere in the network*. This paper reports *Celerity* solution as a first step in making this vision a reality: by combining a polynomial-time tree packing algorithm on the source and an adaptive rate control along each overlay link, we are able to maximize the source rates without any *a priori* knowledge of the underlying physical topology in the Internet. *Celerity* has been implemented in a prototype system, and extensive experimental results in a "tough" dumbbell LAN testbed and on the Internet demonstrate *Celerity*'s superior performance over the state-of-the-art solution Simulcast and Mutualcast.

As future work, we plan to explore source rate control mechanisms beyond the 2-hop tree-packing limitation in *Celerity* to further improve its performance without incurring excessive overhead.

8. REFERENCES

[1] Skype, "http://www.skype.com/intl/en-us/home."

[2] Cisco, "http://newsroom.cisco.com/dlls/2010/prod_111510c.html."

[3] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: an efficient mechanism for content distribution in a P2P network," in *Proc. ACM SIGCOMM Asia Workshop*, Beijing, 2005.

[4] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *Proc. ACM SIGMETRICS*, Annapolis, MD, 2008.

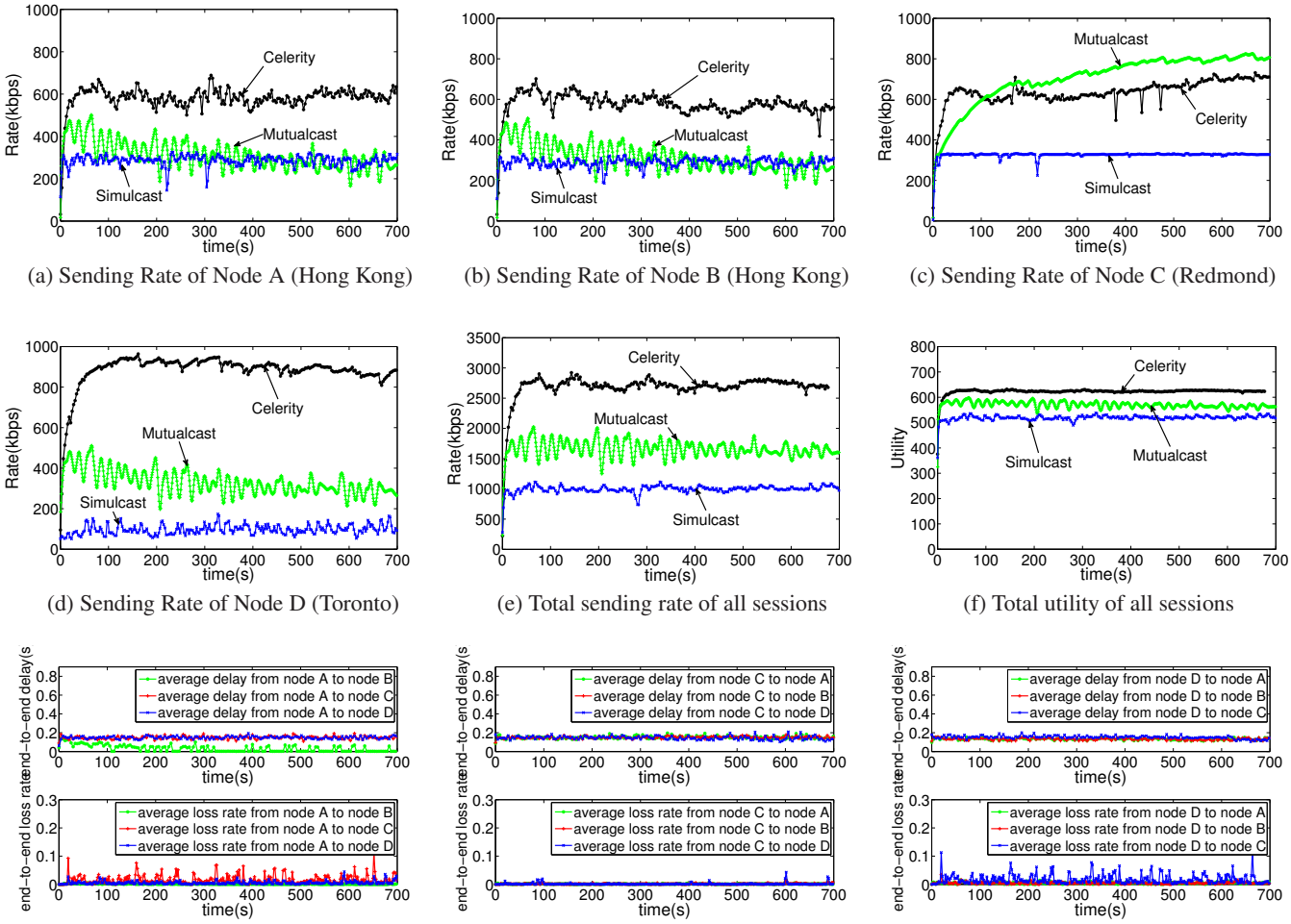
[5] İ. E. Akkuş, Ö. Özkasap, and M. Civanlar, "Peer-to-peer multipoint video conferencing with layered video," *Journal of Network and Computer Applications*, vol. 34, no. 1, pp. 137–150, 2011.

[6] M. Ponec, S. Sengupta, M. Chen, J. Li, and P. Chou, "Multi-rate peer-to-peer video conferencing: A distributed approach using scalable coding," in *IEEE International Conference on Multimedia and Expo*, New York, 2009.

[7] ——, "Optimizing Multi-rate Peer-to-Peer Video Conferencing Applications," *IEEE Trans. on Multimedia*, 2011.

[8] C. Liang, M. Zhao, and Y. Liu, "Optimal Resource Allocation in Multi-Source Multi-Swarm P2P Video Conferencing Swarms," *accepted for publication in IEEE/ACM Trans. on Networking*, 2011.

[9] A. Akella, S. Seshan, and A. Shaikh, "An empirical evaluation of



(a) Sending Rate of Node A (Hong Kong)

(b) Sending Rate of Node B (Hong Kong)

(c) Sending Rate of Node C (Redmond)

(d) Sending Rate of Node D (Toronto)

(e) Total sending rate of all sessions

(f) Total utility of all sessions

(g) Average end-to-end delay and loss rate from node A to other nodes

(h) Average end-to-end delay and loss rate from node C to other nodes

(i) Average end-to-end delay and loss rate from node D to other nodes

Figure 7: Performance of four-party conferences over the Internet, running prototype systems of *Celerity*, Simulcast, and the scheme in [4]. (a)-(d): Throughput of individual sessions. (e): Total throughput of all sessions. (f): Utility achieved by different systems. (g)-(h): End-to-end delay and loss rate of session *A*, *C*, and *D* for the *Celerity* system.

wide-area internet bottlenecks," in *Proc. of the 3rd Internet Measurement Conference*, 2003.

[10] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating internet bottlenecks: Algorithms, measurements, and implications," in *Proc. of ACM SIGCOMM*, 2004.

[11] V. Vazirani, *Approximation algorithms*. Springer Verlag, 2001.

[12] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: A low-delay multi-party conferencing solution," The Chinese University of Hong Kong, Hong Kong, Tech. Rep., 2011. [Online]. Available: http://arxiv.org/abs/1107.1138

[13] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, no. 5, pp. 556 – 567, Oct. 2001.

[14] L. Guo and I. Matta, "QDMR: An efficient QoS dependent multicast routing algorithm," in *Proc. IEEE Real-Time Technology and Applications Symposium*, Canada, 1999.

[15] L. Lovasz, "On two minimax theorems in graph theory," *Journal of Combinatorial Theory, Series B*, vol. 21, no. 2, pp. 96–103, 1976.

[16] Y. Wu, M. Chiang, and S. Kung, "Distributed utility maximization for network coding based multicasting: A critical cut approach," in *Proc. IEEE NetCod 2006*, 2006.

[17] K. Arrow, L. Hurwicz, H. Uzawa, and H. Chenery, *Studies in linear and non-linear programming*. Stanford university press, 1958.

[18] A. Nedić and A. Ozdaglar, "Subgradient methods for saddle-point problems," *Journal of optimization theory and applications*, vol. 142, no. 1, pp. 205–228, 2009.

[19] R. Bruck, "On the weak convergence of an ergodic iteration for the solution of variational inequalities for monotone operators in Hilbert space," *Journal of Mathematical Analysis and Applications*, vol. 61, no. 1, pp. 159–164, 1977.

[20] F. Kelly, "Fairness and stability of end-to-end congestion control," *European Journal of Control*, vol. 9, no. 2-3, pp. 159–176, 2003.

[21] S. H. Low, L. Peterson, and L. Wang, "Understanding vegas: A duality model," *Journal of ACM*, vol. 49, no. 2, pp. 207–235, 2002.

[22] D. P. Bertsekas, *Nonlinear programming*. Athena Scientific Belmont, MA, 1999.

[23] J. Park, M. Gerla, D. Lun, Y. Yi, and M. Medard, "Codecast: a network-coding-based ad hoc multicast protocol," *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 76–81, 2006.

[24] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Trans. on Information Theory*, vol. 46, no. 4, pp. 1204–1216, 2000.

[25] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: Towards low-delay multi-party conferencing over arbitrary network topologies," in *ACM NOSSDAV*, 2011.