A Control-Theoretic Approach to In-Network Congestion Management

Ning Wu, Yingjie Bi, Nithin Michael, Ao Tang, John C. Doyle and Nikolai Matni

***Abstract*—WANs are often over-provisioned in order to accommodate worst-case operating conditions, with many links typically running at only around 30% capacity. In this paper, we show that *in-network congestion management* can play an important role in increasing network utilization. To mitigate the effects of in-network congestion caused by rapid variations in traffic demand, we propose using High Frequency Traffic Control (HFTraC) algorithms that exchange real-time flow rate and buffer occupancy information between routers to dynamically coordinate their link-service rates. We show that the design of such dynamic link-service rate policies can be cast as a distributed optimal control problem that allows us to systematically explore an enlarged design space of in-network congestion management algorithms. This also provides a means of quantitatively comparing different controller architectures: we show, perhaps surprisingly, that centralized control is not always better. We implement and evaluate HFTraC in the face of rapidly varying UDP and TCP flows, and in combination with AQM algorithms. Using a custom experimental testbed, a Mininet emulator, and a production WAN we show that HFTraC leads to up to 66% decreases in packet loss rates at high link utilizations as compared to FIFO policies.**

I. Introduction

Wide area network operators are faced with a daunting task, as packet losses must be kept at very low levels [1] while network traffic is known to cycle through periods of both high and low demand [2]. In order to meet these and other service level agreements (SLAs) (e.g., restrictions on latency, jitter, etc.) in the face of variable and unpredictable traffic demand, WANs are often over-provisioned in order to accommodate a wide range of possibilities that include both normal and worst-case conditions. Naturally, this conservative approach results in low average utilization, with many network links typically running at around 30% capacity. As network operators strive to lower cost and stay competitive, more responsive approaches that can reduce this over-provisioning become increasingly appealing.

While successfully addressing these issues will ultimately require rethinking how control is done across the protocol stack, this paper focusses on the challenge of controlling, avoiding and managing congestion in the network layer. As such, we assume that a Traffic Engineering (TE) problem has already been solved based on nominal estimates of traffic demands, leading to a fixed routing topology. When traffic demands are inelastic, fluctuations in demand around the nominal values used to solve the TE problem lead to in-network congestion. For elastic demands, existing solutions use combinations of congestion control and congestion avoidance protocols wherein end-hosts use feedback from the network – provided in the form of packet drops (e.g., [3], [4]), ECNs (e.g., [5], [6], [7]), RTTs (e.g., [8]), and queueing delay estimates (e.g., [9]) – to adjust their transmission rates so as to respect the capacity constraints of the network. All of these algorithms are reactive, adjusting their transmission rate based on some form of congestion signal from within the network. Therefore, in both the inelastic and elastic setting, *in-network congestion is unavoidable*.

With the introduction of SDN, solutions [1], [2], [10] have emerged that successfully increase utilization while avoiding congestion by carefully combining rapid and clever TE updates with end-host pacing. The results are impressive, but at this time, their applicability is limited to tightly controlled environments, such as DCNs or enterprise inter-datacenter WANs. Outside of these settings in-network congestion is simply unavoidable when aiming to increase network utilization. Despite the wide body of literature aimed at controlling and avoiding congestion, little attention has been paid to how to mitigate the effects of congestion that has *already entered* the network on performance. The challenge is that congestion management naturally benefits from network-scale coordination, but traffic demand fluctuations are a fast timescale phenomena. It is therefore worth asking: can network-scale coordination and control be effectively applied to manage RTT timescale traffic demand fluctuations?

This paper provides an affirmative answer. By allowing routers to exchange flow rate and buffer occupancy information in real-time, so that they can dynamically coordinate their link-service rates to suitably "spread" congestion throughout the network, we show that the resulting control policies explicitly use *buffers as resources* to manage and mitigate the effects of in-network congestion. Our approach is partially inspired by the success of static in-network pacing, which has been shown to provide benefits in the "tiny-buffer" setting [11], and of feedback rule based end-host pacing in the DCNs [12] and optical packet switched networks [13], [14]. Our work expands upon these ideas by allowing switches to simultaneously react quickly to local congestion events and to coordinate with each other to optimize system wide utilization and packet loss rates.

This combination of *dynamic control* and *network-scale coordination* results in a family of algorithms that we call High Frequency Traffic Control (HFTraC), and is what allows HFTraC to achieve higher network utilization and lower packet loss rates than existing static, end-host only or link-by-link rate-limiting policies. Further, that HFTraC works

N. Wu, Y. Bi, and A. Tang ({nw276,yb236}@cornell.edu, atang@ece.cornell.edu) are with the department of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853, USA. N. Michael (nm373@cornell.edu) is with Waltz Networks, San Francisco, CA 94134, USA. J. C. Doyle and N. Matni ({nmatni,doyle}@caltech.edu) are with the department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA.

This research was in part supported by the NSF and AFOSR, and by gifts from Google and Huawei.

by only rate-limiting the routers' interfaces and does not change queuing management or routing strategies makes it compatible with and complementary to standard congestion control, AQM and TE approaches. We also demonstrate that at the rapid timescales that we consider, the responsiveness of the link-rate controllers plays an important role in performance – to that end, we provide a systematic study of HFTraC implementation architectures, varying from completely centralized to completely decentralized. We show that the degree of decentralization of an architecture loosely determines its latency, describe the associated latency/performance tradeoff and use it to quantitatively compare different architectures, thus providing a fresh perspective on the often philosophical debate pertaining to network architectures.

Our key findings (all in §V) are that:

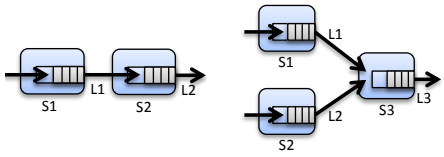
1) HFTraC can *simultaneously* achieve an up to 80% decrease in packet loss rate relative to FIFO and up to 50% decrease in average queue length relative to static smoothing (Fig. 6a).
2) HFTraC leads to improvements in packet loss % and network utilization when used with TCP/AQM (Figs. 13-15).
3) The coordinated implementation of HFTraC leads to up to a 30% decrease in packet loss rate as compared to the centralized implementation (Figs. 6a-7).

We further make the following contributions:[1] (i) we show that the dynamic service rate control task can be cast as a distributed optimal control problem (§III); (ii) we implement practical HFTraC controllers in the context of WANs using OpenFlow enabled switches (§IV); and (iii) we provide a systematic evaluation of HFTraC via a hardware testbed, Mininet emulation, and a production WAN.

II. Using Buffers as Resources

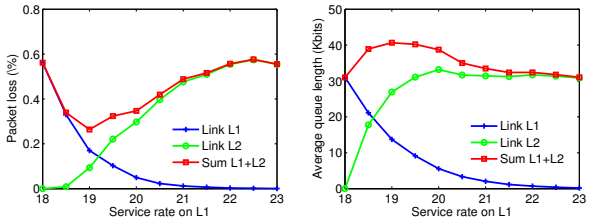
Through the use of a series of toy examples built around the systems shown in Figs. 1a & 1b, we demonstrate how dynamic pacing policies based on buffer occupancy and link-rate information can increase network utilization and robustness to traffic demand variability.[2] We start with familiar static pacing policies, and gradually increase the level of dynamic coordination between switches. We note that in the interest of clarity, we restrict ourselves to end-host only pacing, but that the results of §V all employ dynamic end-host and in-network pacing.

Static end-host pacing: A common approach to traffic pacing at end-hosts is to set a static rate limit – as such we use the performance achieved by such static approaches as a baseline upon which we seek to improve. The system shown in Fig. 1a has link capacities of $C_1 = 23\,\text{Mbps}$, $C_2 = 18\,\text{Mbps}$, both buffer capacities are $200\,\text{Kbits}$, and traffic demands arriving in $S1$ are i.i.d. normally distributed with mean $16\,\text{Mbps}$ and standard deviation $2.5\,\text{Mbps}$. For this system, we set a static rate limit on the upstream link: the upstream buffer therefore absorbs and dampens the variations in link



(a) (b)

Fig. 1. (a) Two tandem switches. (b)Two SD pairs share a common link.



(a) (b)

Fig. 2. Static end-host pacing reduces packet loss %.

rates seen downstream. We assume that $S2$ is implementing a FIFO strategy, and is thus trying to process packets as quickly as possible by setting the service rate for link $L2$ to be equal to its capacity C_2, while $S1$ will select a constant service rate for $L1$ from within $[C_2, C_1]$. Fig. 2 shows that smoothing traffic to $19\,\text{Mbps}$ (lower than C_1) leads to minimal packet loss at the expense of slightly longer average queue lengths.

Buffer Length Coordination: We next explore the benefits of dynamic smoothing policies that vary as a function of downstream buffer occupancy. Such a coordinated approach spreads congestion in **space and time** by making upstream buffers slow their line-service rates when downstream buffers are congested, and conversely, if downstream buffers are (nearly) empty, then upstream buffers are more aggressive in alleviating local congestion. Using the same example, we show how a dynamic rate control policy implemented at $S1$ can lead to superior performance. Figs. 3a-b show how such a buffer occupancy based smoothing policy (computed as described in §III) outperforms the optimal static policy described in the previous example in terms of packet loss % at the expense of slightly longer queue lengths – Figs. 3c-d show similar results for the maximum utilization at which total loss rate is 0.1%. As we demonstrate empirically in §V and prove in §VI, our approach allows for a systematic exploration of this tradeoff between packet loss % and average queue length.

Buffer Length and Link Rate Coordination: In order to illustrate that further benefits can be achieved by coordinating based on both buffer length and link rates, consider the system in Fig. 1b in which two source-destination (SD) pairs must share link $L3$. The links all have capacity $36\,\text{Mbps}$, all buffer capacities are $800\,\text{Kbits}$, and the traffic demands arriving at both $S1$ and $S2$ are i.i.d. normally distributed with mean $17.5\,\text{Mbps}$ and standard deviation $4\,\text{Mbps}$. In this case the resulting control policies at switches $S1$ and $S2$ (again obtained using the optimal control based methods described in §III) depend on the buffer lengths at $S1$ and $S2$ and the link service rates at $L1$ and $L2$. Fig. 4 shows that a dynamic control policy dramatically outperforms a static FIFO policy in terms

[1] An earlier extended abstract [15] suggested HFTraC as an in-network congestion management scheme, but provided only proof-of-concept experimental support of its benefits.

[2] See §III and §V for a discussion of the i.i.d. normally distributed traffic variations used in these examples.

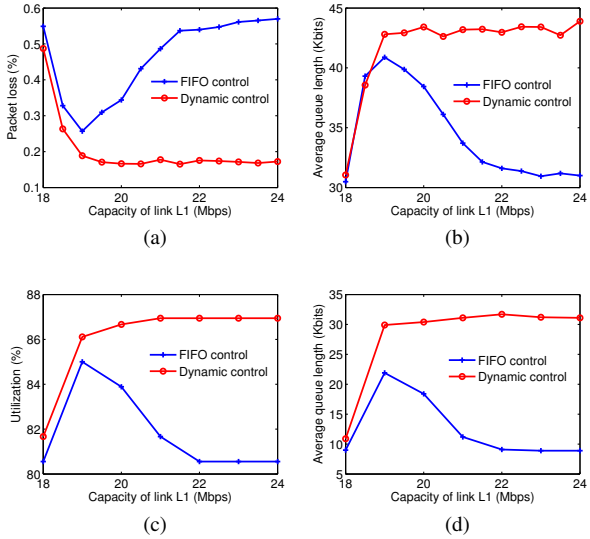


Fig. 3. Dynamic buffer-based pacing outperforms FIFO in terms of packet loss and utilization.

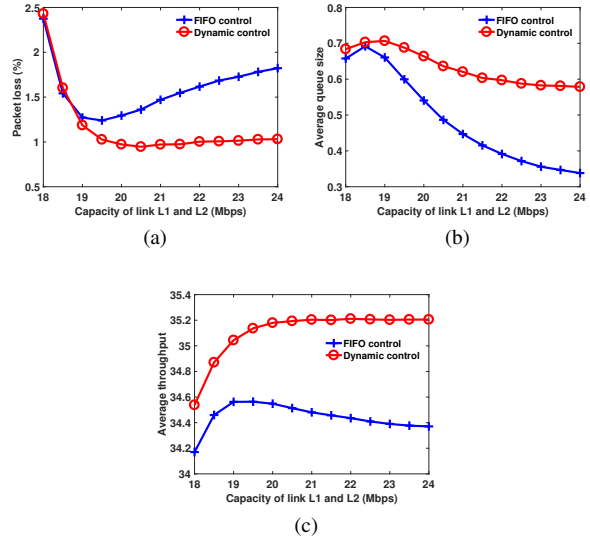


Fig. 4. Dynamic buffer and rate-based control improve packet loss % and average throughput.

of packet loss % and average throughput (as measured by the average transmission rate on link $L3$) at the expense of slightly larger average queue size. This is another simple illustration of how our approach allows for the systematic exploration of such tradeoffs.

III. Optimal Rate Controllers

In the previous simple examples, the correct way for buffers to coordinate is somewhat intuitive, but it can be difficult to identify such opportunities in large-scale networks. We propose using optimal control to synthesize dynamic pacing policies, as these methods automatically recognize and exploit such opportunities for coordination. We later show experimentally that by suitably varying certain parameters at the design stage, this approach allows network operators to explore a larger design space of congestion management algorithms and optimally tradeoff between packet % loss (or maximum utilization) and average queue length.

There are four components that need to be specified to formulate the optimal rate control problem: (i) a dynamic model of traffic demands entering the network; (ii) a dynamic model describing the evolution of these fluctuations through the network; (iii) an objective function that can be used to explore performance tradeoffs; and (iv) the information sharing constraints, as defined by the implementation architecture, imposed on the optimal pacing control feedback policy.

We adopt a fluid network model, and begin by introducing a static model of the network operating under nominal conditions, as is commonly used to solve TE problems. We then introduce stochastic fluctuations in traffic demand, leading to a dynamic model well suited to analyzing and controlling undesirable transient behavior in network state. From this model, we define the optimization objective as the weighted sum of flow rate deviations (from the nominal rates specified by the solution to a TE problem) and buffer occupancy, and argue that it is a natural means of exploring tradeoffs between packet loss % and average queue size.

Model and Notation: We consider a network consisting of a set of switches V and a set of directed links L . We let $\mathcal{L}_v^{in}$ and $\mathcal{L}_v^{out}$ denote the set of incoming and outgoing links respectively for a switch $v \in V$. The network is shared by a set I of source-destination (SD) pairs and the traffic demand is denoted as d_i for SD pair $i \in I$. Let x_l^i be the arrival rate into the egress buffer associated with link l and let f_l^i be the transmission rate on link l due to SD pair i. The total arrival rate and transmission rate on link l are defined as $x_l = \sum_{i \in I} x_l^i$ and $f_l = \sum_{i \in I} f_l^i$ respectively. Here we assume a switch-based routing scheme, in which each switch v splits flows along outgoing links according to fixed split ratios $\alpha_{l,v}^i$ (as specified by the solution to a TE problem) satisfying $\sum_{l \in \mathcal{L}_v^{out}} \alpha_{l,v}^i = 1$ for each SD pair i that utilizes link l. Analogous formulations exist for path-based or label-based forwarding.

From static to dynamic traffic demands: An idealized static flow model, as is typically used when solving a TE problem, assumes that the SD traffic demands are static. For each SD pair i, we denote this nominal static traffic demand by d_i^*, and assume that a TE solution provides a corresponding set of split ratios $\alpha_{l,v}^i$ such that (i) the traffic demands d_i^* are met for each SD pair i, and (ii) the arrival rate $x_l^* := x_l(t) = \sum_{i \in I} (x_l^i)^*$ on each link l satisfies $0 \leq x_l^* \leq C_l$, where C_l denotes the capacity of link l, and $(x_l^i)^* = \alpha_{l,v}^i \sum_{k \in \mathcal{L}_v^{in}} (f_k^i)^*$. Because the arrival rate does not exceed the link capacity at any time, we have that $f_l^* = x_l^*$. Finally, we define

$$\beta_l^i := \frac{(f_l^i)^*}{\sum_{k \in I} (f_l^k)^*} \tag{1}$$

to be the fraction of packets due to SD pair i on link l.

In contrast to this idealized model, it is known [16] that Internet traffic demands vary across several timescales ranging from months to years (capturing constantly growing long term trends in demand) to seconds to milliseconds (capturing random fluctuations in traffic demand). Our focus is on those fluctuations that occur on timescales large enough to lead to

packet losses, but on timescales short enough that traditional TE and/or congestion control algorithms cannot prevent congestion from entering the network — in particular, we address fluctuations that occur on the scale of 10 to 100 milliseconds. For more about traffic characteristics, see [17], [18], [19], [20] and the references therein.

We therefore relax the assumption of static traffic demands d_i^* and model the now time-varying demand $d_i(t)$ of SD pair i at time t as

$$d_i(t) = d_i^* + \Delta d_i(t), \tag{2}$$

where each demand fluctuation $\Delta d_i(t)$ is an independent and identically distributed (i.i.d.) Gaussian random variable with zero mean and standard deviation σ_i.[3]

To capture the effect of these fluctuations in the network, we now write the arrival rates $x_l^i(t)$ and transmission rates $f_l^i(t)$ as a linear superposition of their nominal values $(x_l^i)^*$ and $(f_l^i)^*$ and perturbations $\Delta x_l^i(t)$ and $\Delta f_l^i(t)$ induced by the fluctuations in traffic demand, i.e.,

$$\begin{array}{rcl} x_l^i(t) & = & (x_l^i)^* + \Delta x_l^i(t), \\ f_l^i(t) & = & (f_l^i)^* + \Delta f_l^i(t) \end{array} \tag{3}$$

We next derive a dynamic model that tracks the propagation of these fluctuations through the network.

A dynamic view of the network: A consequence of this dynamic model is that it is now possible for the arrival rate $x_l(t)$ at time t into link l to exceed the transmission rate $f_l(t)$. Assuming that each link l is equipped with an egress buffer of capacity $b_l^{\max}$, packet drops occur when the buffer length reaches the buffer limit. It follows that the evolution of buffer occupancy at time t at link l, denoted by $b_l(t)$, is given by[4]

$$\dot{b}_l(t) = \begin{cases} \Delta x_l(t) - \Delta f_l(t) & \text{if } 0 \le b_l(t) < b_l^{max}, \\ \min(\Delta x_l(t) - \Delta f_l(t), 0) & \text{if } b_l(t) = b_l^{max}. \end{cases} \tag{4}$$

The following properties then hold:

$$0 \le f_l(t) = \sum_{i \in I} f_l^i(t) \le C_l, \;\; f_l(t) \le x_l(t) \text{ if } b_l(t) = 0,$$

$$\Delta x_l(t) = \sum_{i \in I} \Delta x_l^i(t),$$

$$\Delta x_l^i(t) = \begin{cases} \Delta d_i(t) & \text{if edge link,} \\ \alpha_{l,v}^i \sum_{k \in \mathcal{L}_v^{\text{in}}} \Delta f_k^i(t - \delta_k) & \text{otherwise.} \end{cases}$$

for δ_k the propagation delay on link $k \in \mathcal{L}_v^{\text{in}}$ of switch v.

For the purposes of control, we introduce a controllable buffer egress rate variable Δu, linearize the above model around the nominal rates $(x_l^i)^*$, $(f_l^i)^*$ and empty buffer states $b_l^* = 0$, and sample it at interval τ to obtain the following discrete time model:

$$b_l(n+1) = b_l(n) + \tau(\Delta x_l(n) - \Delta u_l(n)), \tag{5}$$

$$\Delta f_l(n+1) = \Delta u_l(n), \;\; \Delta x_l(n) = \sum_{i \in I} \Delta x_l^i(n), \tag{6}$$

$$\Delta x_l^i(n) = \begin{cases} \Delta d_i(n) & \text{if edge link,} \\ \alpha_{l,v}^i \sum_{k \in \mathcal{L}_v^{\text{in}}} \beta_k^i \Delta f_k(n - n_k) & \text{otherwise.} \end{cases} \tag{7}$$

Here n is the discrete time index, satisfying $\tau n = t$ and $\tau n_k = \delta_k$ for some integers n_k. In Eq. (7) we make the simplifying assumption that β_l^i in Eq. (1) also specifies the ratio of fluctuations due to each SD pair, i.e. $\Delta f_l^i(n) = \Delta f_l(n)\beta_l^i$.

Objective Function: We propose computing dynamic pacing policies that minimize the following objective function:

$$\lim_{n \to \infty} \sum_l [\mathbb{E}[\Delta f_l(n)]^2 + \lambda_l \mathbb{E}[b_l(n)]^2] \tag{8}$$

which is the the weighted sum (specified by the weights $\lambda_l \ge 0$) of the steady state variance in traffic rate fluctuations and buffer occupancy. Although the connection between this objective function and packet loss rate, buffer occupancy and utilization is not immediate, we show in §VI a natural connection between this objective and standard buffer egress policies. In particular, we show that taking the weights λ_l to zero or infinity leads to, respectively, static smoothing or FIFO policies. As we further demonstrate empirically in the §V (see Fig. 6a), assuming sufficiently large buffers, static smoothing policies minimize packet loss % at the expense of longer average queue lengths; in contrast, FIFO policies minimize average queue size at the expense of higher packet loss %. Thus by sweeping the weighting parameters λ_l across a range of values, network operators can trace out a tradeoff curve in the now expanded packet loss % vs. average queue size design space and select an appropriate pacing policy given the constraints of their SLAs.

Information sharing constraints: When computing link-rate control policies, it is important to explicitly model the delays associated with the exchange and collection of network state between routers. To take this latency into account during the design process, for a controller located at link l we define its available information set $\mathcal{I}_l(n)$ at time n to be

$$\mathcal{I}_l(n) := \{(b_k(n - n_{lk}))_{k \in L}, (\Delta f_k(n - n_{lk}))_{k \in L}\}, \tag{9}$$

where n_{lk} is the communication delay associated with exchanging information from link k to link l. This extra level of detail allows us to quantify the effects of architectural decisions at the design stage – in particular, we focus on four different implementation architectures:[5]

1. GOD: The Globally Optimal Delay free (GOD) architecture assumes that a logically and physically centralized controller can instantaneously access global network state information as well as compute and execute control laws Δu_l for each buffer, i.e., it assumes that the communication delays defining the information constraints (9) satisfy $n_{lk} = 0$ for all $l, k \in L$. This architecture is not implementable in practice, but the performance that it achieves is the best possible by a feedback policy [22], and hence represents a benchmark against which all other architectures should be compared.

2. Centralized: A physically centralized controller makes control decisions. For simplicity, we assume that these decisions

[3]Our choice of an i.i.d. Gaussian model for traffic demand fluctuations is supported in the literature [16], and as we show in §V is further supported by CAIDA traces [21]. Modeling traffic demand fluctuations is in itself a very challenging problem, and we do not claim our model to be exact, but rather a reasonable first order approximation. To that end, we discuss and evaluate the robustness of HFTraC algorithms to errors in the statistical model of the traffic demand fluctuation process in §V.

[4]Here we use that $(x_l^i)^* = (f_l^i)^*$ and hence $x_l - f_l = \Delta x_l - \Delta f_l$.

[5]See Fig. 5 for a specific instantiation of these architectures in our experimental testbed.

are based on synchronized global information. The latency of the control loop is determined by $n_{\max} = \max_{l \in L} n_{lk}$, the longest round-trip-time from any router to the centralized controller: it takes $\frac{1}{2} n_{\max}$ for the controller to collect the link-rate and buffer state of the network and another $\frac{1}{2} n_{\max}$ for all routers to receive the control decisions.

3. Coordinated: The coordinated controller is logically and physically distributed, with a local controller located at each router. In this case, n_{lk} is specified by the delay of collecting network state from the router associated with link k by the router associated with link l. Local actions are then taken immediately by local controllers, which compute their line-rate using both timely local and delayed shared information.

4. Myopic: This is a completely decentralized architecture, in which local controllers compute their control laws Δu_l using local information only, i.e., $n_{ll} = 0$, $n_{lk} = \infty$.

Putting it all together: Given the aforementioned dynamic model of traffic fluctuations (2) and network dynamics (5) – (7), the specified objective function (8) and the information sharing constraints (9), we pose the optimal pacing control problem as

$$\begin{aligned} \min_{\Delta u_l(n)} \quad & \lim_{n\to\infty} \textstyle\sum_l [\mathbb{E}[\Delta f_l(n)]^2 + \lambda_l \mathbb{E}[b_l(n)]^2] \\ \text{s.t.} \quad & \text{network dynamics (5) – (7)}, \\ & \Delta u_l(n) = \gamma_l(\mathcal{I}_l(n)) \text{ for all } l \in L, \end{aligned} \tag{10}$$

which is a distributed optimal control problem. The final constraint ensures that the service rates are constrained to be a function γ_l of the available information $\mathcal{I}_l(n)$, as specified by the implementation architecture. We will discuss about solving the problem with respect to the information exchange constraints imposed by different architectures in §IV-A.

Why distributed optimal control? Posing the dynamic pacing synthesis task as the distributed optimal control problem (10) has many advantages. It makes clear that there is a broader design space than existing FIFO and pacing solutions, and provides a systematic way of exploring it. Posing this problem as a *distributed* control problem lets us quantify the effects on performance of different implementation architectures. Further, distributed optimal control naturally handles both the stochastic nature of uncertain traffic demands, and the information sharing delays inherent to spatially distributed WANs. Finally, a key feature of this approach is that it is *feedback* based. The benefits of feedback in providing robustness and performance guarantees in the face of environmental and modeling uncertainty are well-studied and substantial [22]. As we show in §V, in the context of the problem at hand, using feedback provides us with robustness to errors in estimated nominal traffic demands, traffic fluctuation models, and delay jitter, as well as to the other simplifying assumptions that we made in generating our model, as is made clear by the success of our approach on a production WAN. Feedback therefore allows us to use simplified models for analysis and design and be confident that the resulting guarantees carry through to real-world systems.

IV. Design and Implementation

A. Design

The distributed optimal control problem (10) can be solved exactly when the information exchange constraints imposed on the controller are given by the GOD, centralized or coordinated implementation architectures using the methods described in [22], [23]. For the myopic architecture, the resulting problem is in fact NP-hard, and we therefore resort to nonlinear optimization and brute-force search to obtain a reasonable candidate controller to be used in comparison. We highlight that in all cases, pacing control actions can be computed in the controller by multiplying a precomputed gain matrix K (for centralized and GOD architectures) or gain matrices K_l (for coordinated and myopic architectures) with a finite history of link-rate and buffer occupancy states. The details of pacing control actions computation can be found in Appendix A.

Input:
$(x_l^i)^*, (f_l^i)^*$: TE specified nominal flow rates;
λ_l: weight parameters defined in problem (10);
τ: time interval for measuring state variables;
Routing topology and resulting dynamics (5)-(7);
Initialization:
Compute gain matrices K_l by soving problem (10) using methods from [22], [23];
$n \leftarrow 0$;
Output: $(\Delta u_l(n))$: rate control action for all links $l \in L$
while *input variables are not changed* **do**
  foreach *link* $l \in L$ **do**
    Measure $\Delta f_l(n)$ and $b_l(n)$;
    Update $\mathcal{I}_l(n)$ using collected information;
    Compute $\Delta u_l(n)$ using K_l and $\mathcal{I}_l(n)$;
    Send $\Delta f_l(n)$, $b_l(n)$ and $\mathcal{I}_l(n)$ to neighbors;
    Transmit at rate $f_l^* + \Delta u_l(n)$;
    Wait τ sec;
  end
  $n \leftarrow n + 1$;
end

Algorithm 1: Coordinated HFTraC pseudocode.

B. Implementation

Our practical implementation of HFTraC consists of four components: (i) the measurement of link rates and buffer lengths; (ii) applying control actions via rate limiting; (iii) the exchange of state updates and control decisions, as specified by the implementation architecture; and (iv) the computation of control actions within the controller(s). We now present an instantiation of the different HFTraC architectures on our three node experimental testbed.

Experimental testbed implementation: Fig. 5 shows the centralized, coordinated and myopic distributed network architectures as implemented on our experimental testbed. Each node runs Open vSwitch (OVS), with physical links connecting the machines. We implement the different HFTraC architectures as a modules in POX.

Communication: We enable communication between the data and control planes by either sharing the same tunnels with normal traffic, or by implementing dedicated control tunnels. In the centralized architecture (cf. Fig. 5a), a single controller runs on one of the machines and remotely connects to the switches in the other machines. Although the GOD architecture is not implementable in practice, it can be approximated if the control tunnel propagation delays are negligible

compared to the delays of the data tunnels. In the coordinated architecture (cf. Fig. 5b), each controller and its local switch runs on the same machine, with additional communication between controllers to exchange local state information. In the myopic distributed architecture (cf. Fig. 5c), local controllers compute control actions using only local state information – hence no communication tunnels between controllers are needed.

Measurement: HFTraC uses the following state variables: the link rate deviation Δf_l and the egress queue length b_l on link l. Ingress/egress packet amounts and queue lengths can be measured using Linux Traffic Control (TC) tools. By querying the network statistics periodically via TC, controllers gain access to timely measurements of their local link rates and buffer lengths.

Rate Limiting: Rate limiting is needed in order to implement the control actions specified by HFTraC. We use the Token Bucket Filter (TBF) disciplines in TC to provide this functionality: in the TBF queuing discipline, each outgoing traffic byte is serviced by a single token. The most important parameter of the bucket is its size (the number of tokens that it can store). In our implementation, each router, regardless of the system architecture, must be able to perform rate-limiting control in order to implement the HFTraC specified control action. The frequency of rate control depends on the sampling rate. In order to ensure the accuracy of the rate control functionality, the bucket size should be set to be as small as possible, close to Maximum Transmission Unit (MTU). This is because on creation, the TBF initializes with a full bucket, which allows this amount of traffic to burst from the TBF at its first release. Therefore if the rate limit is changed rapidly, the average rate may be skewed by this initial burst of traffic – in practice the real outgoing rate may be slightly larger than the desired rate, unless the bucket size is set small enough to disallow such bursts. In our experiments, we set the bucket size to be $1514\,\mathrm{Bytes}$, equal to the MTU.

Computation: Local or global control actions (depending on the implementation architecture) are computed within the POX controller using local and/or shared measurements and the pre-computed gain matrix K or matrices K_l.

Pseudocode implementation: Algorithm 1 provides a pseudocode implementation of the coordinated HFTraC algorithm operating under a fixed routing topology.[6]

V. Evaluation

Using our experimental testbed, Mininet emulations, and a production WAN we evaluate the ability of HFTraC to improve network utilization in spite of rapidly varying traffic demands. We show that:

- The modeling choices made in §III are consistent with experimentally observed behavior (§V-A).
- HFTraC simultaneously decreases packet loss rate by 80% relative to FIFO average queue length 50% relative to static smoothing (Fig 6a).
- HFTraC achieves an up to 8% increase in link utilization (Fig 11) and 50% decrease in packet loss rate (Fig 12) on our WAN and Backbone Network emulation, respectively.
- HFTraC complements TCP and AQM schemes, yielding up to 50% decreases in packet loss % when used in conjunction with TCP Cubic, TCP Reno with PI or RED, and TCP Vegas with CoDel or PIE (§V-D).
- The coordinated implementation of HFTraC decreases packet loss rates by up to 30% as compared to the centralized HFTraC implementation (Figs. 6a-7).

A. Validating our model

We begin by performing a series of tests using our experimental testbed to validate the modeling decisions made in §III. Namely, we provide empirical support for (i) the validity of the objective function (8) and its ability to interpolate between FIFO (low buffer occupancy) and smoothing (low packet loss) congestion management policies, (ii) HFTraC's ability to increase maximum link utilization in the face of variable demand, (iii) the Gaussian i.i.d. model used for traffic demand fluctuations, and (iv) that the feedback based implementation of HFTraC provides robustness to errors in nominal traffic demand estimates and statistical models of the traffic demand fluctuations.

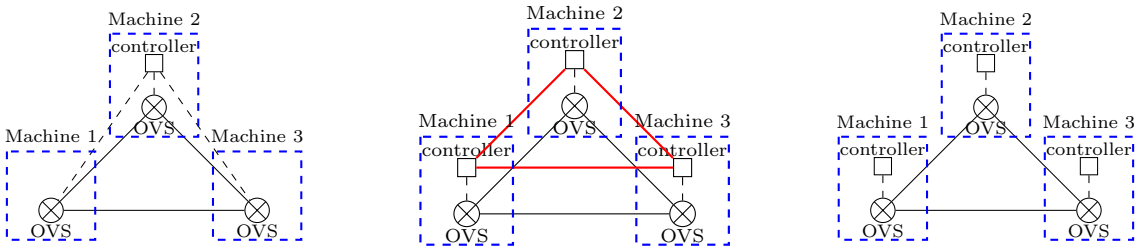
Experimental setup: Our lab testbed, shown in Fig. 5, consists of three switches and three hosts: we overlay it with the routing topology shown in Fig. 7a. Links between switches have a capacity of $30\,\mathrm{Mbps}$ and $0.2\,\mathrm{Mbits}$ egress buffers. Edge links that are connected with hosts have a larger capacity of $42\,\mathrm{Mbps}$ and $0.3\,\mathrm{Mbits}$ egress buffers. Unless otherwise stated, all buffers use a Drop Tail scheme. We choose a sampling time of $\tau = 10\,\mathrm{ms}$ for the control law update frequency. Fig. 5 also shows the different implementation architectures: most are self-explanatory, but we note that we approximately implement the GOD architecture by adding dedicated control tunnels between the centralized controller and switches. The propagation delays are $0.2\,\mathrm{ms}$, which are negligible compared to workload traffic delays.

Workloads: Hosts send and receive UDP traffic via iPerf. The traffic demand increments $d_i(n)$, which we allow to vary every $10\,\mathrm{ms}$, are distributed i.i.d. $\mathcal{N}(d_i^*, \sigma^2)$. TC rate limiting is used in the hosts to realize the variation of source traffic in real time. The average source traffic at $Src1$ is half of that at $Src2$, and thus the flows from $Src2$ are split along two paths ($S2 \to S3$ and $S2 \to S1 \to S3$ with split ratio $3:1$), given by the routing solution provided by standard TE methods.

Expanding and exploring the design space: In §III, we claimed that the objective function (8) allows the user to explore tradeoffs that arise between packet loss % and average queue length. Here we provide an experimentally derived example of such a tradeoff curve by evaluating the performance achieved by HFTraC algorithms designed for various values of the weighting parameters λ_l. We begin with the case where the host buffer limits are infinite, meaning that packet loss can only occur at switch buffers. Fig. 6a shows the empirically derived tradeoff curve between loss rate and average queue length for each of the possible implementation architectures. For these experiments, the traffic demands from $Src1$ are distributed according to a $\mathcal{N}(19, 8^2)$ distribution, and those from $Src2$ follow a $\mathcal{N}(38, 8^2)$ distribution.

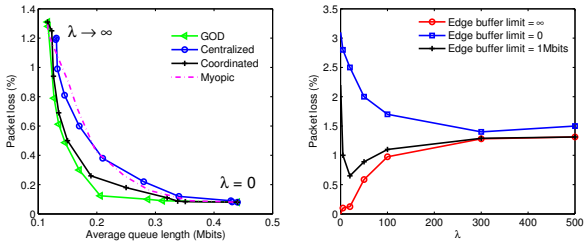
The tradeoff curve demonstrates that there exists a "sweet-spot" in the design space that optimally trades off between

[6]The pseudocode for the other implementation architectures is similar to that shown, differing only in how information is exchanged between routers and controller(s), and hence are omitted due to space constraints.



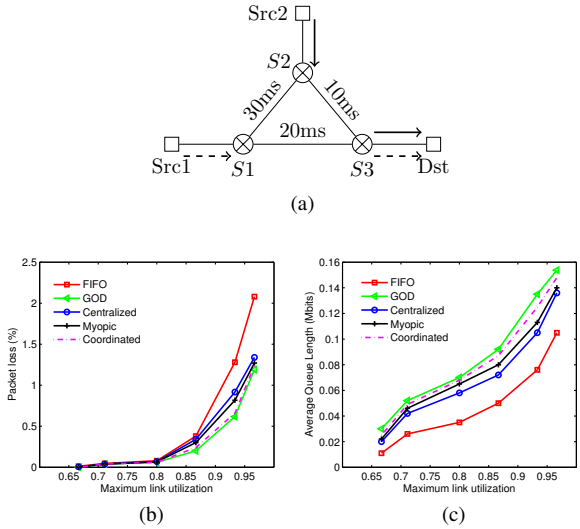
(a) Centralized/GOD (b) Coordinated (c) Myopic distributed

Fig. 5. Architectures implemented on experimental testbed. Lines connecting OVSs are tunnels for normal traffic, connecting OVSs and controllers are communication tunnels between data and control planes, and connecting controllers are tunnels for control message exchange.



(a) (b)

Fig. 6. (a) Tradeoff between loss rate and average queue length for different architectures. (b) Loss rate for varying edge buffer limits under GOD control.



(a) (b) (c)

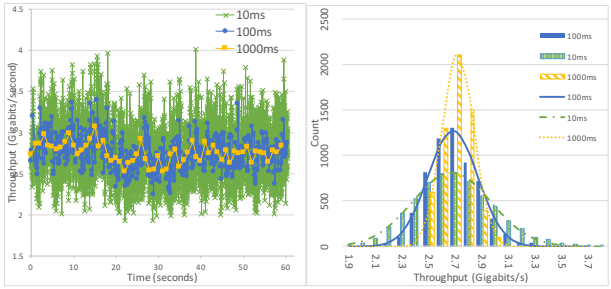
Fig. 7. (a): Experimental testbed setup. (b)/(c): Loss rate/average queue length as utilization is increased.

packet loss % and average queue length: in particular, a well-tuned coordinated implementation of HFTraC can simultaneously achieve an 80% decrease in packet loss rate relative to FIFO and 50% decrease in average queue length relative to static smoothing. It also shows that the combination of responsiveness and coordination of the coordinated architecture provides an advantage over the centralized and myopic implementations, and nearly matches the idealized performance of the GOD architecture. Finally, when the weights λ_l are all set to 0, we recover a static pacing solution that minimizes packet loss % at the expense of larger average queue lengths; conversely, if the weighting parameters tend to infinity, we recover a FIFO solution that minimizes average queue length at the expense of larger packet loss %.

Edge buffer limits matter: The previous experiment was somewhat idealized, in that we assumed infinite buffers at the network edge. We investigate the impact of edge buffer limits, and Fig. 6b shows how the loss rate changes as a function of the weighting parameter λ for three different cases of host buffer limit sizes using the GOD controller. The other architectures will exhibit qualitatively similar trajectories. As expected, the more buffer resources available, the more dramatic the performance improvements of HFTraC.

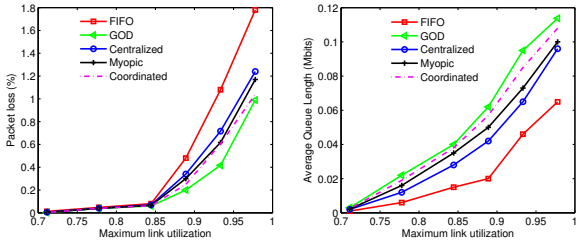
Network-scale coordination increases utilization: We now show that an appropriately tuned HFTraC algorithm can allow for higher link utilization than standard FIFO approaches. For this experiment, we set traffic fluctuations to have a standard deviation of $\sigma = 8$, and gradually increase the nominal values d_i^* to explore the behavior of HFTraC and FIFO schemes as maximum link utilization increases. Fig. 7 shows that HFTraC algorithms are able to reduce packet loss by suitably spreading congestion due to traffic demand fluctuations across buffers – not surprisingly, this in general leads to slightly larger average queue lengths compared to FIFO policies. Thus we see that HFTraC, regardless of architecture, effectively reduces the packet loss %, especially at high maximum link utilizations. The combined responsiveness and coordination of the coordinated HFTraC implementation achieves very similar performance to that of the idealized GOD architecture, both of which provide up to 50% improvements in packet loss % relative to FIFO.

Traffic demand fluctuations can be modeled as i.i.d. Gaussians: In the following set of experiments, we use internet traces extracted from the CAIDA anonymized dataset [21], which are recorded with nanosecond scale timestamps. Fig. 8a shows the averaged link-rates measured across different timescales. The histograms in Fig. 8b show that the link-rate values are all well described by Gaussian curves of the same mean and with variances that increase with the sampling period. It can further be verified that data points are approximately uncorrelated in time, with empirical cross-correlations between neighboring time-points being two orders



(a) (b)

Fig. 8. (a)CAIDA trace sampled at three timescales: 10ms, 100ms, and 1000ms. (b) Histogram and Gaussian curve fitting of link rates at these timescales.



(a) Loss rate (b) Average queue length

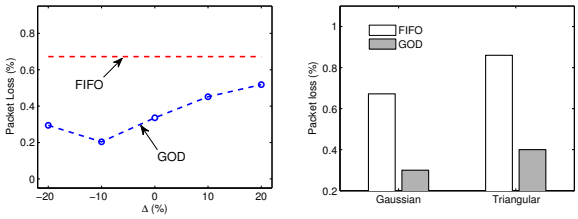
Fig. 9. Loss rate and average queue length for data-driven experiment with CAIDA traces.

of magnitude smaller than the mean link-rate.[7] The CAIDA traces capture the behavior of aggregate flows, which is the level of granularity at which TE solutions typically operate (as in, e.g., [1], [2]) – therefore although individual TCP or UDP flows may not exhibit the i.i.d. Gaussian behavior used in our model, the CAIDA traces suggest it to be a reasonable first order approximation for the behavior of aggregate flows. We repeated the experiments of the previous section, shown in Fig. 7a, with these CAIDA traces driving the traffic demand. We appropriately scaled the rates to be meangingful for our testbed, resulting in mean throughputs of $27.5\,\text{Mbps}$ and $13.75\,\text{Mbps}$ for $Src1$ and $Src2$, respectively. Fig. 9 shows similar curves to the results from our previous experiment. We show next that the feedback based implementation of HFTraC provides robustness to, among other things, deviations from this statistical model.

HFTraC is robust to modeling errors: Our model is built under the assumption that nominal traffic demand rates are exactly known, and that traffic demand fluctuations follow an i.i.d. zero mean Gaussian distribution. We claimed at the end of §III that an advantage of the feedback based nature of HFTraC is that it provides robustness to modeling errors: we demonstrate this now using our experimental testbed.

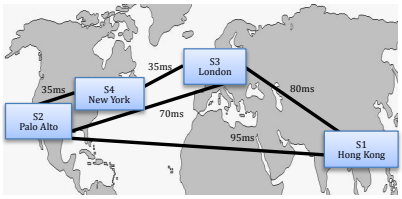
For simplicity, we restrict ourselves to the GOD architecture, but later validate the claim for all architectures in §V-B, by showing that HFTraC provides substantial benefits when implemented on a production WAN that is subject to delay jitter, control packet losses and other non-ideal behavior. In

[7]In particular, we compute the sample cross-correlation between neighboring data-points for sampling times of 10 ms and 100 ms, and find it to be 1.3% and .75% of the mean link rate, respectively.

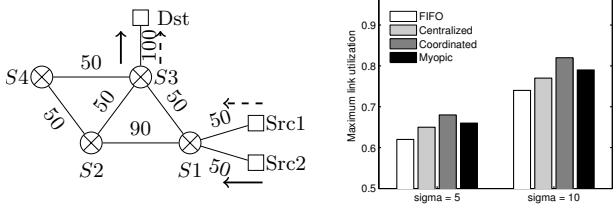


(a) (b)

Fig. 10. HFTraC is robust to modeling errors. (a) Packet loss % as a function of nominal traffic demand estimation error. (b) Packet loss % improvements for Gaussian and Triangular fluctuation distributions.



(a) Network topology



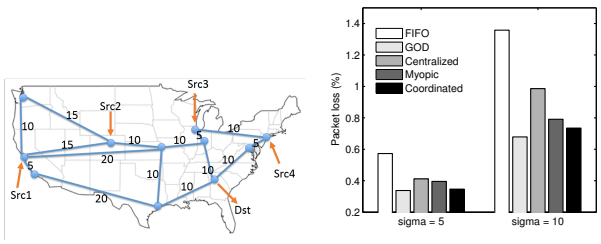
(b) Experimental setup (c) Maximum link utilization

Fig. 11. WAN testbed experiments. HFTraC leads to up to an 8% increase in link utilization.

Fig. 10a, we show the robustness of HFTraC to deviations of up to 20% from the nominal traffic demand used to compute the buffer egress line-rates. This shows that even when the actual rate is 20% larger than the estimated nominal value, HFTraC still provides an advantage over a FIFO strategy, and in particular, when the actual value is larger than estimated, HFTraC mitigates the effects of the unexpectedly larger traffic demands, resulting in a lower packet loss %. Similarly, in Fig. 10b, we illustrate the performance of HFTraC when the source rate fluctuation is triangularly distributed – HFTraC once again significantly outperforms the FIFO strategy. The source rate is taken to be i.i.d. $\mathcal{N}(30, 6^2)$ and compared to rates that are i.i.d. with a symmetric triangular distribution between $[0, 60]$.

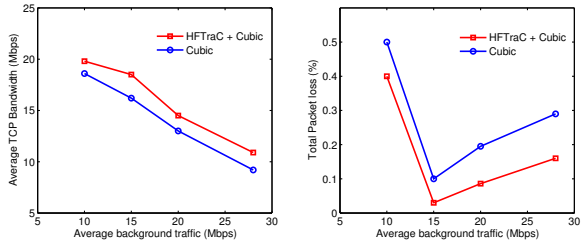
B. WAN Experiments

We evaluate the performance of HFTraC on a WAN testbed with 4 sites spread across 3 continents, as shown in Fig. 11(a). Delays are specified on each link in Fig. 11a. The experimental setup is shown in Fig. 11b, where capacities (Mbps) are also labeled on each link. All buffer limits are $0.8\,\text{Mbits}$. Each site has one virtual machine with OVS running as the WAN-facing switch. The link delays used during the design phase are estimated by running Ping between each pair of switches and computing an average delay. The state information and



(a) Network topology (b) Loss rate

Fig. 12. Abilene network emulation.



(a) TCP average bandwidth (b) Total loss rate

Fig. 13. TCP average bandwidth and total loss rate with/without coordinated HFTraC.

control decision messages are distributed to controllers using the links shared with normal traffic – as such these control packets are subject to delay jitter and the occasional drop. The sampling/update period is $35\,\mathrm{ms}$.

Our approach to handling dropped control packets is to assume that the network is operating under nominal conditions until updated information arrives. This means that when a control packet is dropped, the corresponding buffer simply transmits at its TE nominally specified rate f_l^*. Similarly, if the state information ($\Delta f_l(n)$ and $b_l(n)$ for link l) is lost at time slot n, it is assumed that $\Delta u_l(n) = \Delta f_l(n) = 0$ and $b_l(n) = b_l(n-1)$. Notice that the effects of lost state are only felt by the system until a fresh state is received by the controller, after which it can reset itself immediately – therefore the system is robust to control packet losses.

Because we are subject to physical propagation delays, we cannot implement the GOD architecture – all other architectures are implemented and compared to a FIFO strategy. We use standard deviations in the traffic sources of $\sigma = 5$ and $\sigma = 10$, and empirically determine the maximum achievable link utilization achieved subject to loss rates of 0.1% and 0.5%, respectively.[8] Fig. 11c shows that all implementations of HFTraC improve the link utilization over FIFO, with more substantial benefits coming in the face of more volatile traffic – when $\sigma = 10$, the coordinated HFTraC implementation leads to a nearly 8% increase in utilization as compared to a FIFO strategy. We emphasize that we do not enforce synchronization in traffic rate updates across buffers, relying instead on the inherent robustness of our feedback based implementation.

C. Backbone Network Emulation

We use Mininet to emulate the Abilene network, shown in Fig. 12a, which has 11 nodes and 28 $100\,\mathrm{Mbps}$ links with propagation delays in ms specified by their labels. We use UDP traffic to replay the demands extracted from the data sampled in prior studies [16]. The maximum link utilization is set to 85% and we evaluate the packet loss rate for traffic demand fluctuations with standard deviations of $\sigma = 5$ and $\sigma = 10$. Fig. 12b shows that all implementations of HFTraC allow for a significant decrease in packet loss % relative to FIFO control, and that the coordinated HFTraC implementation in particular decreases packet loss rate by over 50%.

[8]The higher allowable loss rate for $\sigma = 10$ is why higher link utilization is achieved.

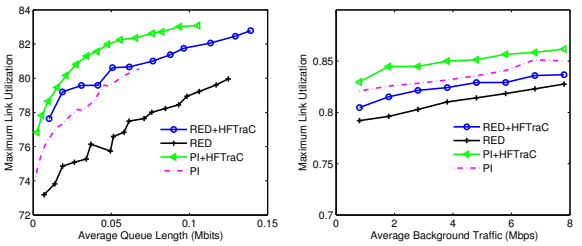
D. HFTraC and Congestion Control

Here we show that HFTraC is complementary to established TCP/AQM congestion control strategies that operate at similar timescales, leading to improvements in average throughput, maximum utilization and packet loss %.

HFTraC improves TCP throughput: We evaluate the performance of the coordinated HFTraC implementation working in conjunction with TCP congestion control using the lab testbed in Fig. 7a. We use iPerf to send TCP traffic from $Src1$ to Dst, with Cubic [24] enabled in $Src1$. Different levels of background UDP traffic are introduced using appropriately scaled CAIDA traces as driving inputs. Fig. 13 clearly shows that the coordinated HFTraC implementation reduces the total loss rate significantly (by over 50% when the average background traffic is between $15\,\mathrm{Mbps}$ and $20\,\mathrm{Mbps}$), and thus the average throughput of the TCP flow is also increased. The high loss rate at low-levels of background traffic is to be expected: most losses occur at $S1$ when the TCP rate is increased beyond the capacity of link $S1 \to S3$.

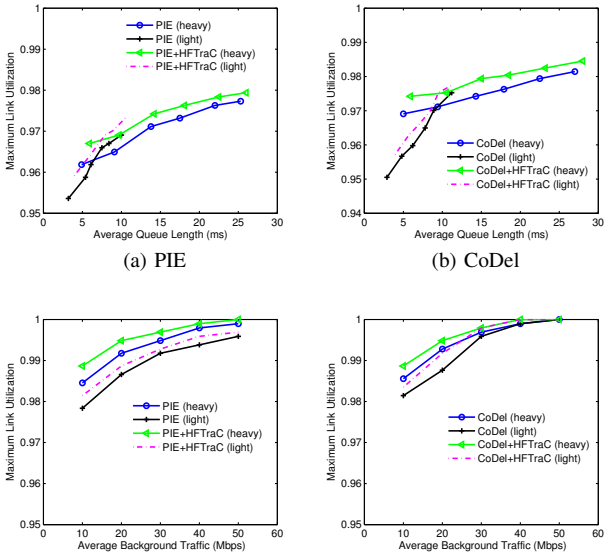
HFTraC improves TCP/AQM link utilization: For the following series of experiments, we use a discrete-event network simulation of the experimental three-node topology to evaluate HFTraC working in conjunction with TCP and AQM. In particular, we implement RED [4] or PI [25] AQM strategies in the routers. We send 20 sessions of TCP flows from both $Src1$ and $Src2$ to Dst. The packet size is $1\,\mathrm{kBytes}$. ECN is enabled and hosts implement TCP Reno. We evaluate the average queue length-utilization tradeoff for RED and PI working with and without HFTraC by sending only TCP traffic, and varying the minimum queue threshold of RED and queue reference of PI. In the second simulation scenario, we introduce varying background UDP traffic from $Src2$ to Dst and fix the minimum threshold to $15\,\mathrm{packets}$ in RED and queue reference to $10\,\mathrm{packets}$ in PI. Fig. 14 shows that in both cases RED/PI + HFTraC achieves up to 5% more utilization than RED/PI alone under the same levels of buffer occupancy, and subject to the same background traffic.

Using the Backbone Network emulator, we show the compatibility of HFTraC with TCP Vegas and PIE or CoDel. There are four TCP senders and one receiver, as is illustrated in Fig. 12a, with all traffic routed along the smallest RTT path. We enable PIE and CoDel in the Linux Kernel (4.2.0-42-generic) via the TC interface with the following settings: $tupdate = 30\,\mathrm{ms}$ for PIE, $interval = 100\,\mathrm{ms}$ for CoDel, and limit $= 200$ packets for both. We run the tests for 100s in two congested



(a) (b)

Fig. 14. Maximum link utilization of RED and PI working with/without HFTraC. (a) Varying targeted queue length. (b) Varying background UDP traffic.



(a) PIE (b) CoDel

(c) PIE (d) CoDel

Fig. 15. Max utilization achieved by TCP Vegas with PIE/CoDel with/without HFTraC as a function of average buffer occupancy and background UPD traffic.

schemes: light and heavy traffic loads, corresponding to 5 and 30 TCP Vegas flows per sender, respectively. We first use only TCP flows and vary the target delay for PIE and CoDel. Figs. 15a and 15b show the throughput of the bottleneck link (edge link for Dst) with respect to the average queue length of all routers along the path from $Src1$ to Dst as the target delay varies from $5\,\mathrm{ms}$ to $30\,\mathrm{ms}$. We then fix the target delay at $20\,\mathrm{ms}$ and add 5 UDP flows from $Src3$ to Dst. We show the bottleneck link throughput under various background traffic loads in Figs 15 (c) and (d). In all cases, HFTraC is compatible with PIE and CoDel, leading to increased link utilization.

VI. Discussion

Comparing architectures: Notice that any centralized algorithm can be implemented on a coordinated architecture, and that likewise, any coordinated algorithm can be implemented on the GOD architecture. Similarly any myopic algorithm can be implemented on a coordinated architecture. This simple observation lets us order the performance achieved by the best control policy on each of these architectures. Let ν_{GOD}, ν_{cen}, ν_{coord} and ν_{myop} denote the optimal cost achieved in optimal control problem (10) by a control policy implemented using the GOD, centralized, coordinated and myopic architectures, respectively. We then have that $\nu_{\text{GOD}} \leq \nu_{\text{coord}} \leq \nu_{\text{cen}}$ and $\nu_{\text{GOD}} \leq \nu_{\text{coord}} \leq \nu_{\text{myop}}$. This qualitative ranking of architectures does not quantify their performance gap – there may be situations where a centralized architecture is preferable (perhaps because logically centralized algorithms can be simpler to implement, debug and maintain). By computing the optimal cost achieved by a centralized controller and comparing to that achieved by a coordinated controller, we can *quantify the tradeoff between centralization and performance*. For the example in Fig. 7, the computed norms were $\nu_{\text{GOD}} = 585.49 \leq \nu_{\text{coord}} = 634.5 \leq \nu_{\text{myopic}} = 791.48 \leq \nu_{\text{cen}} = 1009.269$, which is consistent with the results of that experiment.

Recovering FIFO and smoothing algorithms: When the λ_l tend to infinity, the cost function only penalizes buffer size variance without any consideration for link rate deviations. Now given that at a specific buffer, $b_l(n+1) = b_l(n) + \tau(\Delta x_l(n) - \Delta u_l(n))$ (Eq. (5)) it is clear that the optimal policy with respect to this cost function is simply to set $\Delta u_l(n) = \Delta x_l(n) + \frac{1}{\tau} b_l(n)$, that is to say, to empty out the buffer immediately, thus recovering a FIFO approach. This policy is optimal as the resulting cost of optimal control problem (10) is 0. Conversely, when the λ_l are set to zero, the cost function only penalizes link rate deviations without any consideration for queue occupancy. Now given that $\Delta f_l^i(n) = \Delta u_l(n-1)\beta_l^i$, we achieve a cost of 0 if we set $\Delta u_l(n) = 0$ for all time n. Recall that setting $\Delta u_l(n) = 0$ means that the optimal policy specified in this case is simply to set $u_l(n) = f_l^*$ for all time n, hence recovering a static smoothing policy – this is a good model for the rate limiting used in [1]. Thus by varying λ between 0 and ∞, one traces out a Pareto-optimal curve in which buffer size is traded off against "flow smoothness," as we showed in §III & §V.

Incremental deployment:. This suggests an incremental deployment strategy for HFTraC – simply set the appropriate weights λ_l to either 0 or to be very large depending on whether a non-HFTraC buffer is implementing a FIFO or smoothing policy.

VII. Related Work

As far as we are aware, this paper represents the first use of distributed optimal control in the field of networking, and provides the first example of network-scale control being used to manage the inherently fast timescale phenomena of congestion. Below we summarize other related work.

Optimal Control: We draw upon classical results in optimal control [22] – these in particular are applicable to synthesizing GOD and centralized control laws. When designing coordinated architectures, we build on recent results from the distributed optimal control literature, see [26], [27], [28], [29], [30], [31], [32], [33], [23] and references therein. The use of optimal control in networking is limited, with a notable exception being [34], where the authors use Model Predictive Control to maximize user Quality of Experience in the context of video streaming. They similarly show that a control-theoretic approach expands the design space of dynamic adaptive video streaming algorithms, leading to significant improvements over the state of the art.

Traffic Engineering: Traffic engineering is of great importance for network management and optimization and has attracted much attention over the years, see [35], [36], [37], [38], [39], [40], [41], [42], [43] for a non-exhaustive example list. In particular, the community has recognized the importance of implementing responsive (dynamic) traffic engineering [44], [45], [46], [47], [48], [49], [50]. HFTraC works with fixed routing topologies, and aims to mitigate the effects of congestion caused by fast timescale traffic fluctuations around the average demand values typically used in TE algorithms.

TCP/AQM: Congestion control and avoidance algorithms built around TCP, AQM and ECN have a rich history in the networking community. The corresponding control analysis, see [51], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61], [62], [63], [64], [65], [5] for an incomplete list, use models that also explicitly include physical propagation delay. As demonstrated in §V-D, HFTraC should be viewed as a congestion management algorithm that is and complementary to the congestion control and avoidance provided by TCP/AQM.

Backpressure Techniques: The backpressure algorithm (BP) and its many variants [66], [67], [68], [69], [70], [71] have similar goals to those of HFTraC, in that both BP and HFTraC attempt to distribute packets that have already entered the network so as to minimize some measure of overall congestion. Both algorithms attempt to meet their goals by controlling queue service rates – however, whereas BP selects each packet's next hop based on one hop queue size information, HFTraC assumes a fixed routing strategy but allows for network-wide coordination. This is an important difference, because in contrast to HFTraC, BP algorithms are not compatible with existing TE solutions – further because BP may lead to variable end-to-end delays for packets within the same flow, out-of-order packet arrivals can be an issue when used in conjunction with TCP/AQM schemes. Finally, HFTraC algorithms allow for a principled exploration of tradeoffs between average queue-length and packet drop %.

SDN: We are partially inspired by the recent results in SDN based traffic management. Specific examples include [1], which achieves high utilization through rating limiting maximum smoothing; [72], in which a hierarchical bandwidth allocation infrastructure is proposed; and [2], which addresses issues related to transient congestion. We are also motivated by the richness of the newly expanded network architecture design space enabled by the introduction of SDN, which range from completely centralized [10], to completely decentralized [73].

REFERENCES

[1] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *SIGCOMM*, 2013.

[2] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *SIGCOMM*, 2013.

[3] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, Aug. 1988. [Online]. Available: http://doi.acm.org/10.1145/52325.52356

[4] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking (ToN)*, 1993.

[5] K. Ramakrishnan and S. Floyd, "A proposal to add explicit congestion notification (ECN) to IP," RFC 2481, January, Tech. Rep., 1999.

[6] D. Lapsley and S. Low, "Random early marking for internet congestion control," in *Global Telecommunications Conference, 1999. GLOBECOM '99*, vol. 3, 1999, pp. 1747–1752 vol.3.

[7] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 89–102, Aug. 2002. [Online]. Available: http://doi.acm.org/10.1145/964725.633035

[8] L. S. Brakmo and L. L. Peterson, "Tcp vegas: end to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1465–1480, Oct 1995.

[9] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast tcp: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, Dec 2006.

[10] J. Perry *et al.*, "Fastpass: A centralized zero-queue datacenter network," in *SIGCOMM*, 2015.

[11] N. Beheshti *et al.*, "Experimental study of router buffer sizing," in *ACM SIGCOMM conference on Internet measurement*, 2008.

[12] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, A. Greenberg, and C. Kim, "Eyeq: Practical network performance isolation at the edge," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 297–311.

[13] V. Sivaraman *et al.*, "Packet pacing in small buffer optical packet switched networks," *IEEE/ACM Transactions on Networking (TON)*, 2009.

[14] Y. Cai *et al.*, "A practical on-line pacing scheme at edges of small buffer networks," in *INFOCOM*, 2010.

[15] N. Wu, Y. Bi, N. Michael, A. Tang, J. Doyle, and N. Matni, "Hftrac: High-frequency traffic control," in *Proceedings of the 2017 ACM SIGMETRICS / International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '17 Abstracts. New York, NY, USA: ACM, 2017, pp. 43–44. [Online]. Available: http://doi.acm.org/10.1145/3078505.3078557

[16] P. Tune and M. Roughan, "Internet traffic matrices: A primer," *Recent Advances in Networking*, 2003.

[17] R. Teixeira *et al.*, "Traffic matrix reloaded: Impact of routing changes," in *Passive and Active Network Measurement*. Springer, 2005.

[18] Y. Zhang *et al.*, "Fast accurate computation of large-scale ip traffic matrices from link loads," in *SIGMETRICS*, 2003.

[19] M. Roughan *et al.*, "Experience in measuring backbone traffic variability: Models, metrics, measurements and meaning," in *ACM SIGCOMM Workshop on Internet measurement*, 2002.

[20] P. Tune and M. Roughan, "Spatiotemporal traffic matrix synthesis," in *SIGCOMM*, 2015.

[21] "The CAIDA UCSD Anonymized Internet Traces - February 2012," http://www.caida.org/data/passive/passive_2012_dataset.xml, 2012.

[22] K. Zhou *et al.*, *Robust and optimal control*. Prentice Hall New Jersey, 1996.

[23] Y.-S. Wang *et al.*, "Localized LQR optimal control," in *Decision and Control*, 2014.

[24] S. Ha *et al.*, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS Operating Systems Review*, 2008.

[25] C. V. Hollot, V. Misra *et al.*, "On designing improved controllers for aqm routers supporting tcp flows," in *INFOCOM 2001. IEEE.*

[26] M. Rotkowitz and S. Lall, "A characterization of convex problems in decentralized control," *Automatic Control, IEEE Transactions on*, 2006.

[27] M. Rotkowitz *et al.*, "Convexity of optimal control over networks with delays and arbitrary topology," *International Journal of Systems, Control and Communications*, 2010.

[28] L. Lessard and S. Lall, "A state-space solution to the two-player decentralized optimal control problem," in *Communication, Control, and Computing (Allerton)*, 2011.

[29] C. W. Scherer, "Structured $\mathcal{H}_\infty$-optimal control for nested interconnections: A state-space solution," *System & Control Letters*, 2013.

[30] P. Shah and P. A. Parrilo, "$\mathcal{H}_2$-optimal decentralized control over posets: A state space solution for state-feedback," in *Decision and Control*, 2010.

[31] A. Lamperski and L. Lessard, "Optimal state-feedback control under sparsity and delay constraints," in *IFAC Workshop on Distributed Estimation and Control in Networked Systems*, 2012.

[32] A. Lamperski and J. C. Doyle, "The $\mathcal{H}_2$ control problem for quadratically invariant systems with delays," *Automatic Control, IEEE Transactions on*, 2015.

[33] N. Matni, "Distributed control subject to delays satisfying an $\mathcal{H}_\infty$ norm bound," in *Decision and Control*, Dec 2014.

[34] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 325–338. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787486

[35] R. G. Gallager, "A minimum delay routing algorithm using distributed computation," *Communications, IEEE Transactions on*, 1977.

[36] B. Fortz and M. Thorup, "Increasing internet capacity using local search," *Computational Optimization and Applications*, 2004.

[37] D. Xu *et al.*, "Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering," *IEEE/ACM Transactions on Networking (TON)*, 2011.

[38] A. Sridharan *et al.*, "Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks," *IEEE/ACM Transactions on Networking (TON)*, 2005.

[39] S. Srivastava *et al.*, "Determining link weight system under various objectives for OSPF networks using a lagrangian relaxation-based approach," *Network and Service Management, IEEE Transactions on*, 2005.

[40] D. O. Awduche, "MPLS and traffic engineering in IP networks," *Communications Magazine, IEEE*, 1999.

[41] A. Elwalid *et al.*, "MATE: MPLS adaptive traffic engineering," in *INFOCOM*, 2001.

[42] D. Applegate and E. Cohen, "Making routing robust to changing traffic demands: algorithms and evaluation," *IEEE/ACM Transactions on Networking (TON)*, 2006.

[43] M. Kodialam *et al.*, "Oblivious routing of highly variable traffic in service overlays and IP backbones," *IEEE/ACM Transactions on Networking (TON)*, 2009.

[44] A. Shaikh *et al.*, "Load-sensitive routing of long-lived IP flows," in *SIGCOMM*, 1999.

[45] T. Benson *et al.*, "MicroTE: Fine grained traffic engineering for data centers," in *CoNEXT*, 2011.

[46] S. Kandula *et al.*, "Walking the tightrope: Responsive yet stable traffic engineering," in *SIGCOMM*, 2005.

[47] N. Michael *et al.*, "Optimal link-state hop-by-hop routing," in *ICNP*, 2013.

[48] H. Wang *et al.*, "COPE: traffic engineering in dynamic networks," in *SIGCOMM*, 2006.

[49] I. Gojmerac *et al.*, "Adaptive multipath routing for dynamic traffic engineering," in *GLOBECOM*, 2003.

[50] S. Fischer *et al.*, "REPLEX: dynamic traffic engineering based on wardrop routing policies," in *CoNEXT*, 2006.

[51] T. Alpcan and T. BAŞAR, "Global stability analysis of an end-to-end congestion control scheme for general topology networks with delay," *Turkish Journal of Electrical Engineering & Computer Sciences*, 2004.

[52] S. Deb and R. Srikant, "Global stability of congestion controllers for the internet," *Automatic Control, IEEE Transactions on*, 2003.

[53] C. Hollot *et al.*, "A control theoretic analysis of red," in *INFOCOM*, 2001.

[54] C. Hollot and Y. Chait, "Nonlinear stability analysis for a class of tcp/aqm networks," in *Decision and Control*, 2001.

[55] R. Johari and D. K. H. Tan, "End-to-end congestion control for the internet: Delays and stability," *Networking, IEEE/ACM Transactions on*, 2001.

[56] K. B. Kim *et al.*, "A stabilizing aqm based on virtual queue dynamics in supporting tcp with arbitrary delays," in *Decision and Control*, 2003.

[57] S. Liu *et al.*, "Pitfalls in the fluid modeling of rtt variations in window-based congestion control," in *INFOCOM*, 2005.

[58] S. H. Low *et al.*, "Linear stability of tcp/red and a scalable control," *Computer Networks*, 2003.

[59] L. Massoulie, "Stability of distributed congestion control with heterogeneous feedback delays," *Automatic Control, IEEE Transactions on*, 2002.

[60] F. Paganini *et al.*, "Congestion control for high performance, stability, and fairness in general networks," *Networking, IEEE/ACM Transactions on*, 2005.

[61] Z. Wang and F. Paganini, "Global stability with time-delay in network congestion control," in *Decision and Control*, 2002.

[62] G. Vinnicombe, "On the stability of networks operating tcp-like protocols," in *IFAC*, 2002.

[63] L. Ying *et al.*, "Global stability of internet congestion controllers with heterogeneous delays," *IEEE/ACM Transactions on Networking (TON)*, 2006.

[64] D. Katabi *et al.*, "Congestion control for high bandwidth-delay product networks," in *SIGCOMM*, 2002.

[65] N. Dukkipati, N. McKeown, and A. G. Fraser, "Rcp-ac: Congestion control to make flows complete quickly in any environment," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–5.

[66] Tassiulas and Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE transactions on automatic control*, 1992.

[67] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control," *IEEE/ACM Transactions on Networking (TON)*, 2007.

[68] X. Lin and N. B. Shroff, "Joint rate control and scheduling in multihop wireless networks," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, 2004.

[69] E. Modiano *et al.*, "Maximizing throughput in wireless networks via gossiping," in *ACM SIGMETRICS Performance Evaluation Review*, 2006.

[70] L. Georgiadis *et al.*, "Resource allocation and cross-layer control in wireless networks," *Foundations and Trends® in Networking*, 2006.

[71] L. Jiang and J. Walrand, "Scheduling and congestion control for wireless and processing networks," *Synthesis Lectures on Communication Networks*, 2010.

[72] A. Kumar *et al.*, "BwE: Flexible, hierarchical bandwidth allocation for WAN distributed computing," in *SIGCOMM*, 2015.

[73] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," in *SIGCOMM*, 2014.

[74] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience New York, 1972.

APPENDIX A
OPTIMAL CONTROL LQR

The discrete-time Linear Quadratic Regulator (LQR) problem [74], as an important problem in control theory, is described as follows: given a discrete-time linear system with dynamics of form

$$\begin{aligned} X(n+1) &= AX(n) + Bu(n) + w(n), \\ Y(n) &= CX(n), \end{aligned} \tag{11}$$

where $X(n)$, $u(n)$, $Y(n)$, and $w(n)$ are state variables, control variables, measured outputs and disturbances respectively at the nth sampling time in the discrete time system, the goal is to determine a control sequence $u(n), 0 \leq n \leq N$ so as to minimize the cost function

$$\sum_{n=0}^{N-1} \left(\| u(n) \|^2 + \| Y(n) \|^2 \right) + X^{\intercal}(N) Q X(N).$$

The optimal control sequence is given by

$$u(n) = KX(n), \tag{12}$$

where K is the optimal gain matrix solved by the LQR algorithm given a well-defined optimal control problem.

As a core component of HFTraC design, the control algorithm aims at solving the optimal control problem described in (10). This problem can perfectly fit into LQR problem, where the state variables consist of buffer length $b_l(n)$ and link rate deviation $\Delta f_l(n)$, and the control outputs are the change of service rate $\Delta u_l(n)$. Therefore, the corresponding optimal gain matrix K can be obtained by solving the LQR problem off-line and used to compute the control laws $\Delta u_l(n)$ by (12) at each control update. Fig. 1 shows how HFTraC works within one update of some TE algorithm given invariant network conditions. The mean demand $(f_l^i)^*$ will be updated once after a TE's update.

We next show how to map network flow model with delay constraints to the LQR problem. We first define the variables in the state-space equations of the discrete-time linear system. Let N be the number of links $l \in \mathcal{L}$ involved in the system. Suppose the control interval τ and the propagation delay δ_l for link l satisfy that $\delta_l = n_l \times \tau$ for some integer n_l. Let $\Delta f_l(n)$ denote the link rate change at timestamp n on link l. A vector $\Delta \mathbf{F}_l(n)$ represents the reverse sequence of link rate

changes from n to $n - n_l$, i.e. $\Delta\mathbf{F}_l(n) = (\Delta f_l(n), \Delta f_l(n-1), \ldots, \Delta f_l(n-n_l))^\top$. We further let vectors $\Delta\mathbf{f}(n)$ and $\mathbf{b}(n)$ denote the rate changes and buffer lengths for all links :

$$\Delta\mathbf{f}(n) = (\Delta\mathbf{F}_1(n), \ldots, \Delta\mathbf{F}_l(n), \ldots)^\top,$$
$$\mathbf{b}(n) = (b_1(n), \ldots, b_l(n), \ldots)^\top, \forall l \in \mathcal{L}.$$

The lengths of the column vectors are $N_f = \sum_{l\in\mathcal{L}}(n_l + 1)$ and $N_b = N$ respectively.

So the state variable $\mathbf{X}(n) \in \mathbb{R}^{N_f+N_b}$ consists of the states of link rate changes and buffer lengths, i.e.

$$\mathbf{X}(n) = (\Delta\mathbf{f}(n), \mathbf{b}(n))^\top.$$

And the control variable $\mathbf{u}(n)$ is a vector of service rate changes for all links, i.e.

$$\mathbf{u}(n) = (\Delta u_1(n), \ldots, \Delta u_l(n), \ldots)^\top, \forall l \in \mathcal{L}.$$

We then determine state-space equation in the following form:

$$\begin{bmatrix}\Delta\mathbf{f}(n+1)\\ \mathbf{b}(n+1)\end{bmatrix} = A\begin{bmatrix}\Delta\mathbf{f}(n)\\ \mathbf{b}(n)\end{bmatrix} + B\Delta\mathbf{u}(n) + C\Delta\mathbf{w}(n), \quad (13)$$

We know that $\Delta F_l(n+1) \in \mathbb{R}^{n_l+1}$ for any link l only depends on $\Delta F_l(n)$ and control variable $\Delta u_l(n)$, according to Eq. (6). Assuming no disturbances in the system, we have

$$\Delta F_l(n+1) = A_l\Delta F_l(n) + B_l\Delta u_l(n),$$

where the matrix $A_l \in \mathbb{R}^{(n_l+1)\times(n_l+1)}$ and vector $B_l \in \mathbb{R}^{(n_l+1)}$ are in the explicity form

$$A_l = \begin{bmatrix}0 & 0 & \ldots & 0 & 0\\ 1 & 0 & \ldots & 0 & 0\\ 0 & 1 & \ldots & 0 & 0\\ \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \ldots & 1 & 0\end{bmatrix}, B_l = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}.$$

So we can write the general expression for vector $\Delta\mathbf{f}(n+1)$ as

$$\Delta\mathbf{f}(n+1) = A_f\mathbf{X}(n) + B_f\mathbf{u}(n),$$
$$A_f = [A_{f1}, 0_{N_f,N_b}].$$

The matrices $A_{f1} \in \mathbb{R}^{N_f\times N_f}$ and $B_f \in \mathbb{R}^{N_f\times N_b}$ are block matrices with blocks A_l and B_l respectively, and $0_{N_f,N_b}$ is an $N_f \times N_b$ zero matrix.

From Eq. (5), we have

$$\mathbf{b}(n+1) = A_b\mathbf{X}(n) + B_b\mathbf{u}(n),$$
$$A_b = [A_{b1}, I_{N_b}], B_b = -\tau \times I_{N_b}$$

where I_{N_b} is an $N_f \times N_f$ identity matrix, and A_{b1} depends on the routing topology.

Our objective function is

$$\underset{\Delta\mathbf{u}(n)}{\text{minimize}} \quad \sum_{n=1}^{N}\text{cost}(\Delta\mathbf{f}(n), \mathbf{b}(n), \Delta\mathbf{u}(n))$$

where the cost is defined as a quadratic function of the form

$$\text{cost}(\Delta\mathbf{f}(n), \mathbf{b}(n), \Delta\mathbf{u}(n)) =$$
$$\Delta\mathbf{f}(n)^\top S_f\Delta\mathbf{f}(n) + \mathbf{b}(n)^\top S_b\mathbf{b}(n) + \Delta\mathbf{u}(n)^\top R\Delta\mathbf{u}(n),$$

where S_f, S_b and R are symmetric and positive-semidefinite matrices, and we set $R \approx 0$. The parameter λ introduced in our original problem (10) is applied here in matrix S_b.

The form of problem described above perfectly fits into the LQR problem. So the optimal control gain matrix K for this problem can be easily obtained by using some LQR solver. In the GOD architecture, the GOD controller is able to compute the control law $\mathbf{u}(n)$ simply by Eq. 12. In the centralized architecture, the centralized controller will also use K in computing the control laws, but with a predicted state variable $\mathbf{X}'(n)$ instead of $\mathbf{X}(n)$ due to the delay constraints. The prediction is given by

$$\mathbf{X}'(n) = E[\mathbf{X}(n)|\mathbf{X}(n-n_k)]$$
$$= A\mathbf{X}(n-1) + B\mathbf{u}(n-1)$$
$$= A^2\mathbf{X}(n-2) + AB\mathbf{u}(n-2) + B\mathbf{u}(n-1) = \ldots$$
$$= A^{n_k}\mathbf{X}(n-n_k) + A^{n_k-1}B\mathbf{u}(n-n_k+1) + \cdots + B\mathbf{u}(n-1),$$

where n_k is the largest RTT from router to the centralized controller. In the coordinated architecture, we adopted another control theory techniques in order to mitigate the negative effect placed by the delay of network state coordination.

We leverage the recently developed localized optimal control framework [23] to solve control problem (10).[9] This framework allows the control law $u_l(n)$ to be implemented using finite impulse response(FIR) filter banks. In particular, the localized solution to (10) outputs two collections of matrices, $\{M(t)\}_{t=1}^T$ and $\{R(t)\}_{t=1}^T$, for T a user specified horizon,[10] such that the control action $u_l(n)$ of buffer b_l at time instant n can be computed according to the following algorithm

$$\begin{aligned}\mathbf{w}(n) &= X(n) - \hat{X}(n)\\ u_l(n) &= \textstyle\sum_{\tau=0}^{T-1} M_l(\tau+1)\mathbf{w}(n-\tau)\\ \hat{X}_l(n+1) &= \textstyle\sum_{t=0}^{T-2} R_l(\tau+2)\mathbf{w}(n-\tau),\end{aligned} \quad (14)$$

where we use $X(n)$ to denote a stacked vector of the network state

$$\{(b_l(n))_{k\in L}, (\Delta f_l(n)_{l\in L})\}$$

at time n. Eq. (14) makes it clear that to enforce information sharing constraints, it suffices to impose suitable sparsity constraints on the matrices $\{M(t)\}_{t=1}^T$ and $\{R(t)\}_{t=1}^T$ defining the necessary filter banks. The coordinated architecture imposes specific sparsity patterns on these matrices such that $u_l(n)$ respects the information sharing delays imposed by the propagation delays of the network.

[9] Alternative approaches exist to solving these control problems, but we choose the localized approach due to its simple implementation via FIR filter banks.

[10] See [23] for details on how to choose this value.