Distributed Storage Control Algorithms for Dynamic Networks

George Iosifidis, Iordanis Koutsopoulos, and Georgios Smaragdakis

***Abstract*— Recent technological advances have rendered storage a readily available resource, yet there exist few examples that use it for enhancing network performance. We revisit in-network storage and we evaluate its usage as an additional degree of freedom in network optimization. We consider the network design problem of maximizing the volume of end-to-end transferred data and we derive storage allocation (placement) solutions. We show that different storage placements have different impact on the performance of the network and we introduce a systematic methodology for the derivation of the optimal one. Accordingly, we provide a framework for the joint optimization of routing and storage control (usage) in dynamic networks for the case of a single commodity transfer. The derived policies are based on time-expanded graphs and ensure maximum performance improvement with minimum possible storage usage. We also study the respective multiple commodity problem, where the network link capacities and node storage resources are shared by the different commodities. A key advantage of our methodology is that it employs algorithms that are applicable to both centralized as well as to distributed execution in an asynchronous fashion, and thus, no tight synchronization is required among the various involved storage and routing devices in an operational network. We also present an extensive performance evaluation study using the backbone topology and actual traffic traces from a large European Internet Service Provider, and a number of synthetic network topologies. Our results show that indeed our approach offers significant improvements in terms of delivery time and transferred traffic volume.**

***Index Terms*— Network optimization, in-network storage.**

I. Introduction

A. Motivation

DURING the last few years we are witnessing an unprecedented growth in user demand for high speed communication and ubiquitous network access. The population of users increases continuously and novel devices and applications create an ever growing volume of data traffic. In recent reports released by Ericsson [1] and Cisco [2] it is forecasted that, until 2018, mobile data traffic will increase up to 18 times and IP traffic will quadruplicate. These developments pose new challenges for communication networks that cannot be addressed solely by the traditional network upgrading and capacity over-provisioning methods.

Indeed, economics, especially cost reduction, is a main driving force for revisiting the use of network resources. First, increasing the capacity of a network requires serious investment in usage rights and infrastructure. To exemplify, in 2010, AT&T paid 1.93 billion dollars for buying additional spectrum in order to serve the increasing traffic of its mobile users [3]. Even worse, such network upgrading methods are expected to be out-paced by the continuously growing demand [2]. Second, the expenditures of transit and access ISPs increase since the link capacity price decrease cannot compensate for the increasing traffic. Third, cloud providers incur high leasing cost for the transit bandwidth [4], especially in regions where transit cost is still high, e.g., in Asia (where the traffic cost is 3 to 5 times more expensive compared to the US and Europe), that is required for connecting their datacenters. Clearly, today more than ever it is imperative to devise novel network design and management methods and utilize network resources that are hitherto unexploited.

A resource that is currently overlooked at network design is node storage capacity. Recent technological advances have rendered storage a cheap and readily available resource which can be used at small portable devices, at base stations, and at central nodes (e.g., routers) of wireline networks or datacenters. Moreover, emerging virtualization techniques made it possible to easily lease storage resources to third parties [6]. However, in spite of the recent research interest for in-network storage [5], and the related products that are now commercially available (e.g., see [7], [8]) it is still considered an auxiliary resource and remains in the background. In this work, we consider storage as an additional degree of freedom in network design and optimization.

We systematically study the benefits of in-network storage, i.e., storage capacity of the network nodes. Our goal is to identify the conditions under which storage can improve network performance and also to quantify these benefits. We use as performance metric the amount of data that can be conveyed end-to-end over the network in a certain time interval or, equivalently, the incurred delay for the transfer of a certain amount of data. The main idea is to exploit the time diversity in the available link capacity variation patterns. The presented methodology can be easily extended to include other metrics such as the energy consumption per unit of transferred data. We study mechanisms for realizing these storage benefits by employing algorithms that can be executed in a centralized or distributed fashion. The latter is particularly useful for the

Manuscript received March 23, 2014; revised December 28, 2014, October 5, 2015, and June 28, 2016; accepted November 1, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Sengupta. The work of I. Koutsopoulos was supported by the ERC08-RECITAL Project, co-financed by Greece and the European Union (European Social Fund) through the Operational Program Education and Lifelong Learning - NSRF 2007-2013. The work of G. Smaragdakis was supported by the ERC Starting Grant ResolutioNet under Grant ERC-StG-679158.

G. Iosifidis is with the School of Computer Science and Statistics, Trinity College Dublin, University of Dublin, Dublin 4, Ireland, and also with the CONNECT Centre, Dublin 2, Ireland (e-mail: iosifidg@tcd.ie).

I. Koutsopoulos is with the Department of Informatics, Athens University of Economics and Business, 104 34 Athens, Greece (e-mail: jordan@aueb.gr).

G. Smaragdakis is with the Technical University of Berlin, 10623 Berlin, Germany, and also with the MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), Cambridge, MA 02139 USA, and with Akamai Technologies, Cambridge, MA 02142 USA (e-mail: gsmaragd@mit.edu).

Digital Object Identifier 10.1109/TNET.2016.2633370

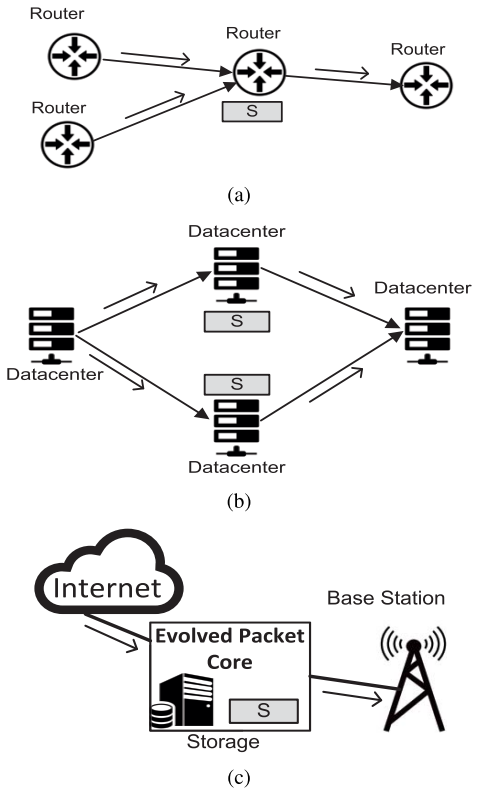

Fig. 1. Instances of in-network storage (a) ISP network with storage enhanced routers (b) Overlay of Datacenters (c) Cellular network with in-network storage.

design of decentralized network mechanisms for large-scale systems [11], [12].

Some specific instances where in-network storage can improve the performance of a network are the following:

- Access ISPs which lease transit bandwidth from higher-tier ISPs based on peak-traffic pricing schemes, i.e., 95%-percentile pricing, can leverage in-network storage for the cost-efficient transfer of data across distant geographic areas. The rational is to temporarily store data at intermediate nodes along a path, as shown in Fig. 1(a) and push it further only when the traffic is low so as to exploit the unutilized already-paid-for bandwidth [13]. The same strategy can be employed by Tier-1 ISPs which dimension their links based on peak traffic. Such methods are increasingly important nowadays due to the advent of network function virtualization (NFV) [9] that allows relatively fast reconfigurations.
- Online service companies and cloud providers typically operate a number of geographically dispersed datacenters. Inter-datacenter traffic constitutes a large portion of their traffic [14] and requires large capital for leasing transit bandwidth [4]. In this case data can be stored at intermediate nodes and routed whenever traffic load is low, i.e., during off-peak hours [15], in order to exploit left-over already-paid-for transit bandwidth [16], see Fig. 1(b). Thus, the cost of applications such as data back up can be significantly reduced using in-network distributed storage of third-parties (e.g., Tier-1 operators) [30].
- Mobile network operators can use in-network storage at their core components or at their small base stations so as to store popular content items which can be cached there during off-peak traffic hours [17], Fig. 1(c). Such architectures are already in place [18], have significant performance/cost benefits [19], and a variety of different applications. For example, delivery of software updates to mobile phones can be benefited and alleviate bottlenecks in the wireless backhaul [20].
- In delay tolerant networks storage is used whenever the path towards the indented receiver is disrupted [21]. When the link capacity variations are known in advance, proper store-and-forward policies can minimize the end-to-end data transfer delay.

The above scenarios share some important characteristics. First of all, the need for storage depends on conditions (related to link available capacities) which may vary across the network. Therefore, data should be stored locally and as close as possible to the intended destination node. Moreover, the links connecting the different nodes (e.g., the different datacenter sites or ISP central nodes) have time-varying traffic loads. In most cases, this is due to the fact that users demands and traffic follows strong diurnal patterns [15], [22], [23]. On the other hand, the link capacities are dimensioned based on peak traffic. Therefore, these systems exhibit periodic variations of the available link capacities that can be predicted in advance.

For these network instances, we are interested in addressing the following questions to improve the network performance[1]: (i) *How much storage should be allocated (added) in each node*? (ii) *Given the storage allocation, in which node(s) and for how long should data be stored?*

B. Methodology and Contributions

We study flow-level storage control policies for dynamic networks. We employ the technique of *time-expanded graphs* (TEG) [25] which map the time evolution of these networks to ordinary static graphs. Storage is modeled by defining a special type of *storage link* connecting different time instances of the same node. The minimum cut (*min-cut*) of the expanded graph represents the upper bound of the data amount that can be transferred by the network within a given time period, and can be increased under certain conditions by adding storage.

We are interested in identifying these conditions and devise the optimal *storage allocation* (*placement*) policy. This is an important design problem since different storage placements have different impact on the end-to-end traffic. The optimal allocation is the one that ensures the maximum possible network performance improvement. We show that this storage allocation depends both on the network topology and the patterns of the available link capacity. To that end, we define the *storage control* policy, which determines how much data should be stored in each node.

Moreover, we suggest that storage control must be considered in conjunction with routing and derive the optimal *Joint*

[1]Henceforth, we will use this term to describe the maximum amount of data that can be transferred end-to-end by a network in a certain time period.

Storage control and Routing (JSR) policy. This JSR policy ensures the maximization of the transferred data within the time period of interest, while using the minimum required storage capacity. Such a scheme can be implemented with an overlay mechanism that determines the routing and store decisions in the network layer. Our methodology is inspired by the *Network Utility Maximization* (NUM) framework [12], [24], yet it significantly departs from previous works due to the joint consideration of these different types of network resources.

Additionally, we use the ϵ-relaxation algorithm to find the optimal JSR policy which is amenable to distributed asynchronous execution [26]. This approach allows us to design decentralized mechanisms (when necessary) which enable the network to adapt its JSR policy in a scalable way using local information and minimum signaling among nodes [11]. Besides, such mechanisms alleviate the problem of single point of failure and distribute evenly the computation load across all nodes [12]. Accordingly, we extend our study for the case different flows (commodities) traverse the network. In this scenario, more often than not, the flows must share the link capacity and intermediate node storage resources.

We evaluate the benefits of in-network storage addition using a spectrum of different network architectures, including the architecture of an actual (operational) large European ISP, and respective real traffic traces. Moreover, we considered different sets of link capacity variation patterns as well as various time periods to evaluate the performance benefits of our approach in a spectrum of settings. In summary, the main contributions of this work are the following:

(i) We provide a methodology for identifying under which conditions in-network storage can improve the data transfer capability of a network with time-varying link capacities.

(ii) For linear (tandem) networks, we analytically quantify the storage benefit as a function of the capacity patterns.

(iii) For general networks we provide a methodology, based on Time-Expanded Graphs (TEG), for finding the storage enhanced min-cut and the optimal storage allocation policy. That is, we find the storage placement beyond which no further improvement is possible in the amount of data transferred within a time interval.

(iv) We introduce the Joint Storage control and Routing (JSR) policy as the solution to a max-flow problem for a single commodity. We explain that in order to maximize performance benefits with the minimum storage capacity, storage usage must be considered in conjunction with routing.

(v) The JSR policy is derived using the ϵ-relaxation algorithm, which is amenable to distributed asynchronous execution, and hence enables the derivation of decentralized JSR mechanisms that are robust to signaling delays and other similar network impairments. Our analysis extends the NUM framework by proposing this joint optimization and employing a hitherto overlooked algorithm.

(vi) We extend our work to the multicommodity scenario where different commodities are transferred over the network and study the respective max-flow multicommodity JSR (MJSR) problem.

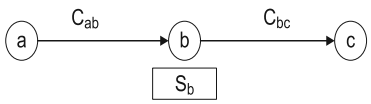

Fig. 2. A 3-node tandem network, where a is the source node, c the sink (destination), and S_b is the available storage at node b.

(vii) A detailed performance evaluation study using the topology of a large European ISP, real traffic traces, and a number of synthetic networks verifies our theoretical findings.

The rest of the paper is organized as follows. In Section II we quantify the performance benefits of in-network storage for linear networks. In Section III we introduce the optimal storage allocation policy for general networks. In Section IV we study joint storage control and routing (JSR) policies and present an algorithm for their derivation. Accordingly, in Section V the respective multicommodity problem is considered. Section VI provides performance evaluation results, and Section VII describes the related works. We conclude in Section VIII.

II. Storage Capacity Allocation for Linear Networks

We begin our analysis by considering linear networks where routing is fixed, from a node to its next one towards the destination. In this case, storage benefit can be analytically quantified for arbitrary link capacity variation patterns.

A. Motivating Example

Consider the three-node linear network depicted in Fig. 2 and assume time slotted operation with slot duration T_0. The link capacities, measured in packets/sec, change every other slot t according to a periodic pattern, i.e., $C_{ab}(t) = D$, $C_{bc}(t) = 1$ and $C_{ab}(t+2) = 1$, $C_{bc}(t+2) = D$, with $D > 1$. In between the 2 slots the capacity of both links remain constant. Transmission delay over each link is equal to the slot duration T_0. In this setting, we ask the question: How much time is required to convey D data packets from node a to node c if (i) the intermediate node b has no storage capacity, (ii) node b has storage capacity of $S_b > D$ packets?

The answer is straightforward, yet illuminating. In the first case, node a can push in each time slot t only as much data as node b is able to forward in the immediately next time slot $t+1$, i.e., $C_{bc}(t+1)T_0$. Hence, for the link capacity variation above, the required time for data transfer is $D+1$ slots. However, when $S_b > D$, node a pushes up to $C_{ab}(t)T_0$ data and the excess amount that cannot be immediately routed to destination node c, i.e., $(C_{ab}(t) - C_{bc}(t+1))T_0$ is stored at node b. In the subsequent slot $t+2$, when C_{bc} is high, stored data along with the new arrived data from node a is delivered to the sink. Therefore, in this case the required time for the delivery of D packets is only 2 slots.

Clearly, the use of storage reduces significantly the incurred end-to-end delay for the data transfer. From a different perspective, storage increases the maximum amount of data that the network can deliver within a certain time interval. Notice that in order to achieve the same performance without storage use, we would have to increase the capacity of either one of the two links up to $D-1$ units. Thus, node storage is actually used as a special type of link capacity and augments the end-to-end

data transfer capability of the network. Finally, it is apparent that the benefit from storage utilization depends on the relative link capacity values. Namely, link capacities should vary with time and, moreover, the capacity variation patterns of the links should be different with each other.

B. Storage Benefits in Linear Networks

We now pose the following questions: (i) what is the incurred delay Del to deliver an amount of D data packets to the destination? and (ii) what is the amount B of data that we can deliver from node a to node c in time period of T time slots, if node b has storage capacity?

When node b has zero storage capacity, the network end-to-end capacity, i.e., the rate at which data can be transferred end-to-end at each slot t is:

$$C_{ac}(t) = \min\{C_{ab}(t), C_{bc}(t+1)\} \tag{1}$$

We denote with $\boldsymbol{C}_{ab} = (C_{ab}(t) : t = 1, 2, \ldots, T)$ and $\boldsymbol{C}_{bc} = (C_{bc}(t) : t = 1, 2, \ldots, T)$ the capacity vectors of links (a, b) and (b, c) respectively for the time period T. In this case, the incurred delay for the transfer of D units of data from node a to node c is $Del = MT_0$ seconds, where M is the minimum integer for which it holds:

$$\sum_{t=1}^{M} C_{ac}(t)T_0 \geq D \tag{2}$$

Also, the amount of data transferred in T time slots is:

$$B = \sum_{t=1}^{T-2} C_{ac}(t)T_0 \tag{3}$$

Notice that, since we assumed that transfer delay is equal to the slot duration T_0, it takes two slots for each packet to reach node c. Hence, data must depart from source node a before slot $T-2$ in order to reach the destination within the T slots.

When node b has storage capacity of S_b data packets, the network end-to-end capacity, denoted $\widehat{C}_{ac}$, changes and can be significantly increased under certain conditions. Specifically, $\widehat{C}_{ac}(t)$ is the minimum of the data that is available at node b at slot t, and the capacity of the last hop link $C_{bc}(t+1)$:

$$\widehat{C}_{ac}(t) = \min\{C_{ab}(t) + Z_b(t), C_{bc}(t+1)\} \tag{4}$$

where $Z_b(t)$ is the accumulated data at node b.

Let us explain the intuition behind this expression. On the one hand, the maximum possible amount of data that can be delivered at node c at slot $t+2$ is upper bounded by the capacity of link (b, c) at $t+1$. On the other hand, if $C_{bc}(t+1)$ is very large, the data that can be delivered is upper bounded by the data that is available at node b at the previous slot. The latter consists of the data that has been transmitted from node a in the exactly previous slot, which is upper bounded by the instantaneous capacity $C_{ab}(t)$, and the amount of data $Z_b(t)$ that has been accumulated until slot t at node b. This quantity is always nonnegative and upper bounded by node b storage capacity S_b. A recursive formula for its calculation is the following:

$$Z_b(t+1) = \min\{S_b, [\big(C_{ab}(t) - C_{bc}(t+1)\big)T_0 + Z_b(t)]^+\} \tag{5}$$

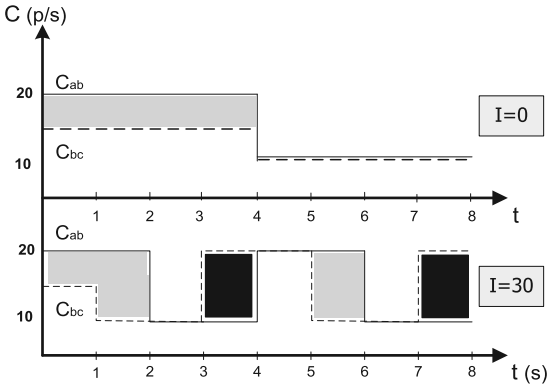

Fig. 3. Dissimilarity Index I, for two different capacity variation patterns in network of Fig. 2. Grey-colored areas correspond to data accumulation in S_b while black-colored areas to data release from S_b.

where $Z_b(0) = 0$ and $[\cdot]^+$ is the projection onto the nonnegative orthant. Therefore in this case the end-to-end capacity of the three-nodes network is $\widehat{C}_{ac}(t) \geq C_{ac}(t)$, $t = 1, \ldots, T$.

This means that the use of storage at intermediate node b improves significantly both the delay and the amount of end-to-end transferred data. The exact improvement that comes with storage at node b depends on the relative variation pattern of link capacities $\boldsymbol{C}_{ab}$ and $\boldsymbol{C}_{bc}$. The more diverse the capacity value sequences are, the larger is the benefit from the storage usage.

In order to quantify the variability in capacity patterns of links (a, b) and (b, c) of graph G, for the time period T, we define a metric that we name *Dissimilarity Index* $I(G, T)$. This metric, which is a function of the capacity variation patterns, measures the aggregate amount of data[2] that can be temporarily stored in node b and subsequently delivered in node c within the period T, assuming that there is no storage capacity constraint. Specifically, parameter $I(G, T)$ is defined as follows:

$$I(G, T) = \sum_{t=1}^{T} \min\left\{Z_b(t), [C_{bc}(t+1) - C_{ab}(t)]^+\right\} \tag{6}$$

where $Z_b(t)$ is calculated from (5) for $S_b = \infty$. An example for $I(G, T)$ is depicted in Fig. 3.

The dissimilarity index reveals *the conditions under which storage is beneficial* for every network G and time interval T. Clearly, when $I = 0$, storage does not increase the amount of end-to-end transferred data. This happens in the following scenarios:

- *No data is accumulated in node b.* This is the case when the capacities of the 2 links have equal values in successive slots, $C_{ab}(t) = C_{bc}(t+1)$, $\forall\, t$, or when link (a, b) has always lower capacity than link (b, c), i.e., $C_{ab}(t) < C_{bc}(t+1)$, $\forall t$. In this case there is no need to utilize intermediate storage at node b since all data that is transmitted from node a can be immediately (after 2 slots) delivered to the destination node c.

[2]Clearly, the storage benefits can be calculated by using directly the capacity variation patterns. However, this metric allows the systematic study of the conditions that render storage beneficial.

TABLE I
EXAMPLE FOR NETWORK OF FIG. 2, $T_0 = 1$, $I = 24$, $S_b = 30$

Slot t	C_{ab} (p/sec)	C_{bc} (p/sec)	C_{ac} (p/sec)	D (p)	Z_b (p)	$\widehat{C}_{ac}$ (p/s)	$\widehat{D}$ (p)
1	10	6	2	0	0	2	0
2	12	2	0	0	8	0	0
3	14	0	10	2	20	10	2
4	2	10	2	2	24	12	2
5	2	12	2	12	14	10	12
6	4	10	4	14	6	10	24
7	6	14	0	16	0	0	34
8	0	0	0	**20**	0	0	**44**

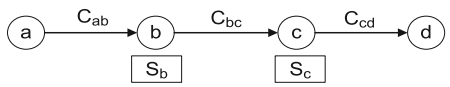

Fig. 4. A tandem (linear) network G of 4 nodes. S_b and S_c is the available storage capacity at node b and c, respectively.

- *Accumulated amount of data in node b cannot be delivered*. Moreover, even if the link capacity variation patterns are such that data is accumulated in node b, it may be impossible to push it further if link (b, c) is always the network bottleneck, i.e., $C_{bc}(t+1) < C_{ab}(t)$, $\forall t$.

In Table I we give a numerical example for the network of Fig. 2 and demonstrate the performance benefit of storage at node b. Capacities are measured in packets/sec (p/s) while data and storage in packets (p).

Similar results hold for linear networks with more than three nodes. For example, consider the four-node linear network depicted in Fig. 4. Node d is the destination while nodes b and c are assumed to have available storage capacity of S_b and S_c packets respectively. For this network we define the dissimilarity index as the aggregate amount of data that is stored either at node b or node c and subsequently is released and delivered to the destination node d. Notice that $I(\cdot)$ depends on the network and on the time interval of interest. The end-to-end data rate is:

$$\widehat{C}_{ad}(t) = \min\{\widehat{C}_{ac}(t) + Z_c(t),\, C_{cd}(t)\}$$
$$Z_c(t+1) = \min\{S_c,\, [\big(\widehat{C}_{ac}(t) - C_{cd}(t+1)\big)T_0 + Z_c(t)]^+\}$$

and the dissimilarity index:

$$I(G,T) = \sum_{t=1}^{T} \min\{Z_c(t), [C_{cd}(t+1) - \widehat{C}_{ac}(t)]^+\} \quad (7)$$

From this analysis, it is clear that the accrued benefit increases with the storage capacity of the nodes and is upper bounded by $I(G,T)$. However, determining the storage allocation policy in general graphs is a non-trivial task. One should carefully determine in which nodes and how much storage to add. We address this problem in the following section.

III. STORAGE CAPACITY ALLOCATION FOR GENERAL NETWORKS

In this section, we provide a method for devising the optimal storage allocation (placement) for a general dynamic network. Such storage placement decisions fall into the class of network planning problems and are made by employing traffic statistics collected over large time periods. Recall that the performance of a network, in terms of data transfer capability, is upper bounded by the capacity $C(Q_{min})$ of the min-cut Q_{min} of its graph [25]. This represents the maximum flow that can be delivered from the source to the destination node. Hence, for a time period of T units, the maximum amount of delivered data is $D = C(Q_{min})T$. Obviously, by increasing the capacity of the min-cut, we increase the maximum amount of transferred data.

Consider a directed network $G = (\mathcal{N}, \mathcal{E})$ with a set of nodes $\mathcal{N}$ and a set of links $\mathcal{E}$. The network is dynamic, i.e., every link capacity $C_{kl}(t)$, $(k,l) \in \mathcal{E}$ changes with time according to a predefined pattern. We assume again a time slotted operation $t = 1, 2, \ldots, T$, with slot duration T_0. Link capacities remain constant within each time slot. First, we assume no storage capacity at all nodes. We also consider the traversal time to be identical for all links and equal to the slot duration. In order to study network G for a certain time period, we use the technique of time-expanded graphs.

Time Expanded Graphs (TEG) were introduced in [25] in order to facilitate the analysis of dynamic networks. Using this method, one can map a dynamic network G for a given time period of T slots to an equivalent static network G_T. This way, it is possible to study the properties and the performance of G for T slots, by applying to G_T the well known methods and results that have been derived for static graphs.

The transformation of a dynamic network to the respective static TEG is accomplished as follows. For every node $k \in \mathcal{N}$ of the original network G, we create in G_T, T nodes, $k^{(t)}$, $t = 1, \ldots, T$. Moreover, for every arc $(k,l) \in \mathcal{E}$ of G, we add a set of corresponding arcs $(k^{(t)}, l^{(t+1)})$, $t = 1, \ldots, T-1$. Hence, the time-expanded network $G_T = (\mathcal{N}_T, \mathcal{E}_T)$ contains $|\mathcal{N}_T| = |\mathcal{N}| \cdot T$ nodes. Notice that we connect node instances in successive time slots because we have assumed that the link traversal time is equal to one slot. In other words there are no links connecting nodes in the same time plane t, or in time planes with distance more than one time slots. One can relax this assumption and follow a similar method for connecting node instances according to the data transfer delay (e.g., if this delay is two slots we would have connected node instances every two slots).

The graph G_T incorporates the notion of time. Namely, the amount of data that can be transferred by network G within time horizon T is upper bounded by the minimum cut of G_T. This is stated in the following lemma:

Lemma 1: The maximum amount of data transferred from source to destination in a dynamic network G, within a time horizon of T slots, is upper bounded by the capacity of the minimum cut of the respective time-expanded graph G_T.

Proof: The proof follows directly from the definition of time-expanded graphs [25]. □

In this work we propose the increase of the min-cut capacity of G_T by the addition of *storage links*. Assume that $Q_T = [\mathcal{M}_T, \mathcal{N}_T \setminus \mathcal{M}_T]$ is the initial min-cut of G_T with capacity $C(Q_T)$, where $\mathcal{M}_T$ is the set of nodes in which the source node belongs and $\mathcal{N}_T \setminus \mathcal{M}_T$ is the set containing the sink node. The *critical observation* is the following. If there exists

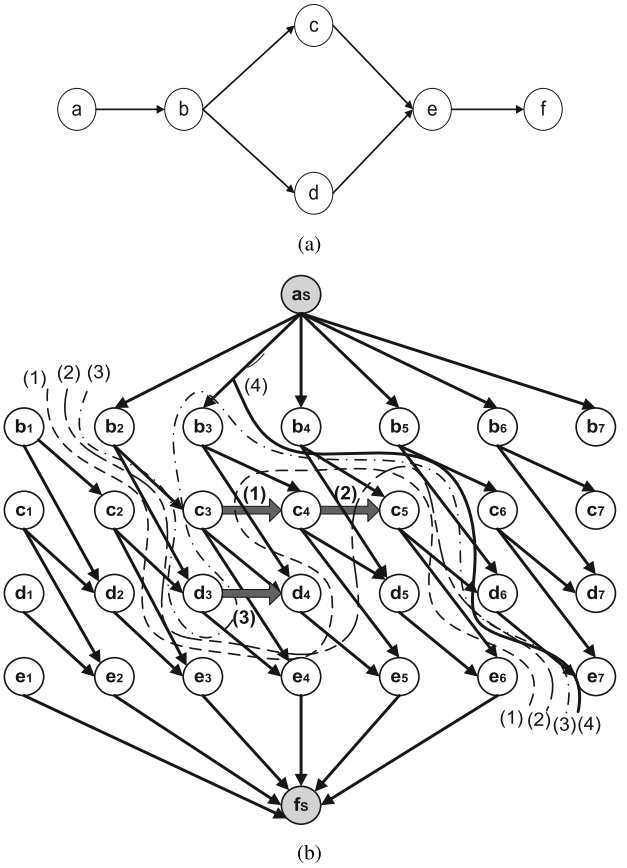

Fig. 5. A dynamic graph with initial min-cut $C(Q_T) = 34$ packets (p). Link capacities for $T = 6$ slots: C_{ab} = (18,16,18,20,16,18), C_{bc} = (4,10,10,4,6,4), C_{bd} = (6,16,16,4,4,6), C_{cd} = (6,10,12,2,12,8), C_{ce} = (6,8,2,10,12,8), C_{de} = (4,6,2,12,12,8), C_{ef} = (10,12,14,10,20,22). In step (1) we add storage $S_c(3)$ =8, in step (2) $S_c(4)$ =4 and in (3) $S_d(4)$ =6. Final capacity is $C(Q_f)$ =42. (a) Original graph G. (b) The Storage enhanced time-expanded graph G_T.

a node $k \in G$ such that $k^{(t)} \in \mathcal{M}_T$ and $k^{(t+1)} \in \mathcal{N}_T \setminus \mathcal{M}_T$ then we can increase $C(Q_T)$ and hence the data transfer capability of the network, by connecting $k^{(t)}$ and $k^{(t+1)}$ with a link of capacity $S_k(t)$. This special type of link represents the capability of node k for storing data during time slot t up to an amount of $S_k(t)$ packets. With the addition of this virtual link the new min-cut capacity of the network increases to $C(Q_T) + S_k(t)$ units. Adding enough storage to node k at time t, renders Q_T a non-minimum cut. If the new min-cut $Q_T^{'}$ contains a node for which the above condition also holds, then we add again storage so as to make $Q_T^{'}$ a non minimum cut.

An example of this method is given in Fig. 5 where we have also merged all time instances of the source and the destination node to nodes a_s and f_s respectively. The method can be incorporated in a greedy algorithm for storage placement in a dynamic network G for a period of T slots. Namely, Algorithm 1 describes the proposed methodology which is applied to a graph G_T and creates a new, storage-enhanced graph G_T^s. The latter has the same set of nodes with G_T, but an increased set of links $\mathcal{E}_T^s$ which contains all links of $\mathcal{E}_T$ and additionally the new storage links. In each step, the min-cut can be found either through a flow-based technique, i.e., using max-flow algorithms [27], or by non-flow techniques [28].

Algorithm 1 Storage Capacity Allocation (SCA)

Input : $G_T = (\mathcal{N}_T, \mathcal{E}_T)$
Output: $G_T^s = (\mathcal{N}_T, \mathcal{E}_T^s)$, $\mathcal{E}_T^s \supseteq \mathcal{E}_T$
Set $\mathcal{E}_T^s \leftarrow \mathcal{E}_T$;
$conv\_flag \leftarrow 0$; # *convergence flag*
while $conv\_flag = 0$ **do**
  Find min-cut $Q_T = [\mathcal{M}_T, \mathcal{N}_T \setminus M_T]$ of G_T^s;
  Construct set $\mathcal{K}$ which contains nodes $k^{(t)} \in G_T^s$, with $k \in \mathcal{E}$, $k^{(t)} \in \mathcal{M}_T$ and $k^{(t+1)} \in (\mathcal{N}_T \setminus \mathcal{M}_T)$;
  if $\mathcal{K} \neq \emptyset$ **then**
    Select randomly a node $k^{(t)} \in \mathcal{K}$;
    Add link $(k^{(t)}, k^{(t+1)})$ with $S_k(t) = \epsilon > 0$;
    while Q_T *is the min-cut* **do**
      $S_k(t) \leftarrow S_k(t) + \epsilon$; # *storage increase*
    end
    $\mathcal{E}_T^s \leftarrow \mathcal{E}_T^s \cup (k^{(t)}, k^{(t+1)})$
  else
    $Q_f \leftarrow Q_T$; # *final capacity reached*
    $conv\_flag \leftarrow 1$;
  end
end

Notice here that we don't consider storage input-output delay in our model. This is a reasonable assumption for networks with line speeds of few Gbps as these can be supported by storage devices that are commercially available today [8], [30]. For higher line speeds, we can easily extent our study by modeling these delays through proper storage links in the time-expanded graph [31], that incorporate this additional delay.

Let us now discuss the complexity and the optimality of the Storage Capacity Allocation (SCA) algorithm. The maximum number of added storage links is bounded by the number of nodes. Additionally, the maximum amount of storage placement in each node is confined by the capacity of the links. In other words, storage addition will eventually result in a minimum cut consisting only of communication links (i.e., links between different nodes and not links between different time instances of the same node). After this point, adding more storage cannot further increase the min-cut of the network. Hence, we can find the storage placement beyond which no further network improvement is achievable.

However, it is easy to see that this greedy algorithm may result in excessive storage placement. Namely, the storage link added in each iteration may render previously added storage capacity redundant. Our next goal is to devise a method for finding the minimum amount of storage capacity that maximizes the network data transfer capability. As we will explain in the sequel, this can be achieved if storage is considered in conjunction (i.e., jointly) with routing. In the following we provide a method for deriving the optimal joint storage control and routing policy for dynamic networks.

IV. Joint Storage Control and Routing Optimization

In this section we define and solve the joint storage control and routing (JSR) optimization problem as a maximum

TABLE II
VARIABLES AND NOTATION

Symbol	Description
k, l	Nodes of initial graph $G = (\mathcal{N}, \mathcal{E})$, $k, l \in \mathcal{N}$
i, j	Nodes of $G_T = (\mathcal{N}_T, \mathcal{E}_T)$, $i, j \in \mathcal{N}_T$
(i, j)	Link of G_T, $(i, j) \in \mathcal{E}_T$, with $i \triangleq k^{(t)}$, $j \triangleq l^{(t+1)}$
x_{ij}	Flow sent over link $(i, j) \in \mathcal{E}_T$
y_{in}	Flow stored at $k \in \mathcal{N}$ in t, $i \triangleq k^{(t)}$, $n \triangleq k^{(t+1)}$
x_{ds}	End-to-end transferred data in T slots
F_i	Forward *communication* (child) nodes of $i \in \mathcal{N}_T$
B_i	Backward *communication* (parent) nodes of $i \in \mathcal{N}_T$

flow problem on the respective time-expanded graph $G_T^s = (\mathcal{N}_T, \mathcal{E}_T^s)$. The notation is summarized in Table II. The solution yields the optimal store and routing decisions that maximize data transfer for a certain time horizon of T slots. Our methodology builds upon the network utility maximization framework [11], [12] and extends it by the joint consideration of these two different types of resources (storage and link bandwidth), and the employment of the ϵ-relaxation algorithm.

Specifically, consider the dynamic network $G = (\mathcal{N}, \mathcal{E})$ for the time interval T. Using the SCA algorithm, we can construct the storage-enhanced graph G_T^s which has the desirable property that its performance is not constrained by the storage capacity of its nodes. Given such a network, our goal here is to find how much data we should route over each link and store in each node, in every time slot, so as to maximize the amount of end-to-end transferred data. We will prove that this policy also ensures the minimum storage usage. In other words, while the previous algorithm is possible to introduce excessive storage placement, the JSR algorithm, which we will define in the sequel, makes use only of the minimum required storage capacity.

We need to emphasize here that the solution of the JSR problem does not presume the execution of algorithm SCA so as to find the storage-enhanced graph G_T^s. Instead, we can construct and use a network (again, we use the notation G_T^s for this graph) that stems from G_T by adding at each node $i \in \mathcal{N}_T$ storage capacity that is bounded by an upper limit S_i^{max}. The exact amount of storage that is actually required - and hence the storage allocation - will be found by the solution of the max-flow problem that we present in the sequel.

Let us now define for each node $k \in \mathcal{N}$ of G the storage control vector $\boldsymbol{S}_k = (S_k(t) : t = 1, 2, \ldots, T)$, where $S_k(t)$ is the amount of data that is stored at node k at time t. Also, we introduce for every link $(k, l) \in \mathcal{E}$ the routing control vector $\boldsymbol{R}_{kl} = (R_{kl}(t) : t = 1, 2, \ldots, T)$ where $R_{kl}(t)$ is the amount of data that is sent over link (k, l) at time t. Finally, we define the network storage control policy $\boldsymbol{S}$ and the network routing policy $\boldsymbol{R}$, for the time period of interest, as follows:

$$\boldsymbol{S} = (\boldsymbol{S}_k : k \in \mathcal{N}), \ \boldsymbol{R} = (\boldsymbol{R}_{kl} : (k, l) \in \mathcal{E}) \tag{8}$$

The JSR problem is formally described as follows:

Definition 1. Joint Storage Control and Routing Max-Flow Problem (JSR Problem). *Given a dynamic network $G = (\mathcal{N}, \mathcal{E})$ with a single source and a single destination, and with nodes with certain storage capacity S_i^{max}, $i \in \mathcal{N}$, find the storage control policy $\boldsymbol{S}^*$ and the routing policy $\boldsymbol{R}^*$, that maximizes the amount of data transferred from source to destination in a given time horizon of T slots.*

We focus on a solution approach based on the ϵ-relaxation method [26] which is amenable to distributed and asynchronous implementation. This means that each node will be able to independently decide how much data to store and route over each link, in every slot. The nodes however need to coordinate their decisions and agree about the optimal JSR policy. This is accomplished through message passing. The circulated messages contain the information that is related to the traffic load for each node, the congestion on its links, and their available storage capacity. We stress here that the proposed algorithm can be also executed centrally whenever this is possible, e.g., for small-scale networks. In this case, the decomposability property of the algorithm can be exploited so as to enable its parallel and hence faster execution.

The proposed scheme can be implemented with an overlay mechanism which adjusts the rate allocation and storage utilization for each link and each node according to the solution of the respective optimization problem. Such distributed mechanisms are particularly important for large-scale networks since (i) they do not presume the existence of a central controller, (ii) allow the computation of the policy to be executed (and hence shared) by many nodes, and (iii) do not require the information about storage and link capacity availability to be communicated to a central point for calculating the policy and back to the nodes for implementing it. Decentralized network mechanisms have received paramount research attention the last few years where various distributed NUM approaches [24] have been proposed for wired and wireless networks [11], [12].

A. JSR Problem Formulation

First we add to G_T^s an artificial link (d, s) connecting the sink d with the source s. This will facilitate the modeling and the analysis of the problem. Each node $i \in \mathcal{N}_T$ which corresponds to a node k of graph G for a certain time slot t, $i \triangleq k^{(t)}$, is connected with a node $m \in \mathcal{N}_T$ which represents the previous instance of the same node, i.e., $m \triangleq k^{(t-1)}$. Similarly, each node $i \in \mathcal{N}_T$ is connected with a node $n \in \mathcal{N}_T$ that represents the subsequent instance of the same node, i.e., $n \triangleq k^{(t+1)}$. The capacity of these links represent the available storage at node k at slots t and $t+1$. For each node $i \in \mathcal{N}_T$ we define the set of forward (child) communication nodes $F_i = \{j : (i, j) \in \mathcal{E}_T\} \setminus \{n\}$, and the set of backward communication nodes $B_i = \{j : (j, i) \in \mathcal{E}_T\} \setminus \{m\}$.

Summarizing, there exist two classes of links in the expanded graph:

- The *communication links* that connect two different nodes i and j at a specific time slot t with capacity $\boldsymbol{C} = \{C_{ij} : i \in \mathcal{N}_T, j \in F_i\}$.
- The *storage links* that connect different time instances of the same node with (maximum) storage capacity $\boldsymbol{S}^{max} = \{S_i^{max} : i \in \mathcal{N}_T, i \triangleq k^{(t)}, k \in \mathcal{N}\}$.

Since the notion of time is incorporated in the time-expanded graph the capacity of all links is measured in data packets. Also, packet size is considered very small and hence the capacities of communication and storage links are considered real (positive) numbers.

Let us define the flow[3] matrix for the graph G_T^s as:

$$\boldsymbol{x} = (x_{ij} \in \mathcal{R}^+ : i \in \mathcal{N}_T, j \in F_i) \tag{9}$$

where $x_{ij} \geq 0$ denotes the amount of data that is sent over link (i,j) with $i \triangleq k^{(t)}$, $j \triangleq l^{(t+1)}$ and $(k,l) \in \mathcal{E}$. Similarly, we define the store decision matrix:

$$\boldsymbol{y} = (y_{in} \in \mathcal{R}^+ : i \in \mathcal{N}_T, i \triangleq k^{(t)}, n \triangleq k^{(t+1)}, k \in \mathcal{N}) \tag{10}$$

where $y_{in} \geq 0$ denotes the amount of data that is stored at node $k \in \mathcal{N}$ during time slot t. Clearly, there is a one-to-one mapping from $\boldsymbol{x}$ to $\boldsymbol{R}$ and from $\boldsymbol{y}$ to $\boldsymbol{S}$, which is shown in Table III.

The optimal storage control and routing policy $(\boldsymbol{x^*}, \boldsymbol{y^*})$ for the time period T is derived from the solution of the max-flow problem which is defined over the corresponding time-expanded graph (JSR Max-Flow Problem):

$$\min_{\boldsymbol{x},\boldsymbol{y}} (-x_{ds})$$

subject to

$$\sum_{j \in F_i} x_{ij} + y_{in} = \sum_{j \in B_i} x_{ji} + y_{mi},\ i \in \mathcal{N}_T \setminus \{s,d\} \tag{11}$$

$$0 \leq x_{ij} \leq C_{ij},\ i \in \mathcal{N}_T, j \in F_i \tag{12}$$

$$0 \leq y_{in} \leq S_i^{max},\ i,n \in \mathcal{N}_T \tag{13}$$

$$\sum_{j \in B_d} x_{jd} = x_{ds}, \qquad \sum_{j \in F_s} x_{sj} = x_{ds} \tag{14}$$

Equation (11) is the flow conservation constraint, and x_{ds} is the amount of data that is transferred from destination to the source node through the added artificial link. Due to (14), this quantity is equal to the data delivered to the destination node. After solving the JSR problem, we find the respective routing and store variables, $R_{kl}^*(t)$ and $S_k^*(t)$ respectively of network G using Table III.

Finally, please notice that the JSR policy can be also derived for the case that there is uncertainty about the actual traffic and therefore the available (residual) capacity of the links. This can be achieved by robust optimization techniques [33]. In particular, for the JSR problem we only need to consider the worst case estimations (highest traffic, thus lowest available link capacities) for the upper bounds of the routing variables. This will ensure that the obtained policy is feasible even under some estimation errors.[4]

B. Distributed Algorithm for the JSR Problem

The JSR problem is a constrained optimization problem and can be solved by dual ascend methods [26]. These methods solve the respective dual problem by iteratively updating the dual variables so as to improve the dual objective function. There are two classes of such algorithms: the primal-dual and the relaxation methods which use different ascent directions but admit fairly similar implementation. The relaxation method is usually faster in practice.

TABLE III
VARIABLE MAPPING FROM GRAPH G TO G_T

Variables of $G = (\mathcal{N}, \mathcal{E})$	Variables of $G_T = (\mathcal{N}_T, \mathcal{E}_T)$
$R_{kl}(t)$	$x_{ij},\ i \triangleq k^{(t)},\ j \triangleq l^{(t+1)}$
$S_k(t)$	$y_{in},\ i \triangleq k^{(t)},\ n \triangleq k^{(t+1)}$
$0 \leq R_{kl}(t) \leq C_{kl}(t)$	$0 \leq x_{ij} \leq C_{ij}$
$0 \leq S_k(t) \leq S_k^{max}(t)$	$0 \leq y_{in} \leq S_i^{max}$

Since objective of the JSR problem is a linear function (hence the dual objective is non-differentiable) typical relaxation methods (or primal-dual methods) may not converge to the optimal problem solution.[5] Therefore, we use the ϵ-relaxation method which converges in polynomial time and is highly suitable for distributed implementation [26], [27]. The underlying idea is that dual variable updates are allowed even if they worsen the dual cost function. The produced pairs of primal-dual variables satisfy the ϵ-complementary slackness which is a perturbed version of the typical complementary slackness conditions. Finally, this algorithm can be executed in an asynchronous fashion which renders it suitable for networks with signaling delays (e.g., delays in the circulation of control messages among the nodes).

First, we define the Lagrange function $L(\cdot)$ by relaxing constraint (11) and introducing the vector of dual variables $\boldsymbol{p} = (p_i : i \in \mathcal{N}_T)$:

$$L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{p}) = -x_{ds} + \sum_{i \in \mathcal{N}_T} \sum_{j \in F_i} (p_j - p_i) x_{ij} + \sum_{i \in \mathcal{N}_T} (p_n - p_i) y_{in}$$

This relaxation admits an interesting interpretation since it decouples the storage and routing decisions of the nodes. The respective dual problem is:

$$\max_{\boldsymbol{p}} q(\boldsymbol{p}) \tag{15}$$

where

$$q(\boldsymbol{p}) = \min_{0 \leq x_{ij} \leq C_{ij}, 0 \leq y_{in} \leq S_i^{max}} L(\boldsymbol{x},\boldsymbol{y},\boldsymbol{p}) \tag{16}$$

The ϵ-relaxation converges to the optimal solution under some mild conditions. We omit the very details and refer the reader to [26, Chap.5.3].

In a distributed setting the variables are circulated among nodes and therefore they need to be time-stamped. Notice that these time stamps τ refer to the algorithm execution time and they should not be confused with the actual time t that represents the slots of T. The basic idea is to exploit the separability property of the dual problem and group the decision variables per node. Specifically, each node $i \in \mathcal{N}_T$ maintains the following variables:

- $p_i(\tau)$: dual variable (or *price*) of node i at time τ.
- $p_j(i,\tau)$: dual variable of $j \in F_i \cup B_i \cup \{n\} \cup \{m\}$, communicated from j to i at time τ. This is the local copy of the dual variable of j, updated by i.

[3]With a slight abuse of terminology, we use the term *flow* for the amount of data ransferred between nodes during each slot. The same term is used also for the amount of data that is stored at each node in each slot.

[4]For example, if e_{ij} denotes the maximum possible estimation error for the available capacity of link $(i,j) \in \mathcal{E}_T$, then we can substitute the respective capacity upper bound C_{ij} with $\tilde{C}_{ij} = C_{ij} - e_{ij}$.

[5]Alternatively, one can use primal-dual methods with subgradient updates and properly selected step sizes.

Procedure 1- Update of Local Variables
if $g_i(\tau) > 0$ **then**
  Update local variables: $p_i(\tau)$, $p_j(i,\tau)$, $x_{ij}(i,\tau)$, $x_{ji}(i,\tau)$, $y_{ij}(i,\tau)$, $y_{ji}(i,\tau)$, by executing Steps 1-4 of the ϵ-relaxation alg. [26, Chap.5.3];
end

Procedure 2- Notification.
Send $\{p_i(\tau), x_{ij}(i,\tau)\}$ to every child node $j \in F_i$;
Send $\{p_i(\tau), x_{ji}(i,\tau)\}$ to every parent node $j \in B_i$;

Procedure 3- Coordination.
For every message received at $\tau' < \tau$ from a child node $j \in F_i$, node i updates its variables:
if $p_j(i,\tau) \leq p_j(\tau')$ **then**
  $p_j(i,\tau) \leftarrow p_j(\tau')$;
end
if $[p_i(\tau) \leq p_j(\tau') + \alpha]\,\&\,[x_{ij}(i,\tau) > x_{ij}(j,\tau')]$ **then**
  $x_{ij}(i,\tau) \leftarrow x_{ij}(j,\tau')$;
end
For every message received at $\tau' < \tau$ from a parent node $j \in B_i$, node i updates its variables:
if $p_j(i,\tau) \leq p_j(\tau')$ **then**
  $p_j(i,\tau) \leftarrow p_j(\tau')$;
end
if $[p_i(\tau) \leq p_j(\tau') - \alpha]\,\&\,[x_{ji}(i,\tau) < x_{ji}(j,\tau')]$ **then**
  $x_{ji}(i,\tau) \leftarrow x_{ji}(j,\tau')$;
end
where $\alpha = -1$ if $(i,j) = (d,s)$ and $\alpha = 0$

Procedure 4- Store Decisions Update.
For every message received at $\tau' < \tau$ from instances $m = i^{(t-1)}$ and $n = i^{(t+1)}$, update store decisions:
if $p_n(i,\tau) \leq p_n(\tau')$ **then**
  $p_n(i,\tau) \leftarrow p_n(\tau')$;
end
if $[p_i(\tau) \leq p_n(\tau')]\,\&\,[y_{in}(i,\tau) > y_{in}(n,\tau')]$ **then**
  $y_{in}(i,\tau) \leftarrow y_{in}(n,\tau')$;
end
if $p_m(i,\tau) \leq p_m(\tau')$ **then**
  $p_m(i,\tau) \leftarrow p_m(\tau')$;
end
if $[p_i(\tau) \leq p_m(\tau')]\,\&[y_{mi}(i,\tau) < y_{mi}(m,\tau')]$ **then**
  $y_{mi}(i,\tau) \leftarrow y_{mi}(m,\tau')$;
end

- $x_{ij}(i,\tau)$: amount of data i forwards to $j \in F_i$ at time τ.
- $x_{ji}(i,\tau)$: amount of data that i decides to admit from node $j \in B_i$ at time τ.
- $y_{in}(i,\tau)$, $y_{mi}(i,\tau)$: amount of data i stores during slots $t-1$, t, where $i \triangleq k^{(t)}$, $m \triangleq k^{(t-1)}$ and $n \triangleq k^{(t+1)}$.
- $g_i(\tau)$: data *surplus* of i, i.e., :

$$g_i(\tau) = \sum_{j \in B_i} x_{ji}(i,\tau) - \sum_{j \in F_i} x_{ij}(i,\tau) + \\ + y_{mi}(i,\tau) + y_{in}(i,\tau) \quad (17)$$

Algorithm 2 JSR Max-flow
Input : $G_T^s = (\mathcal{N}_T, \mathcal{E}_T^s)$
Output: Optimal store $\boldsymbol{S}^*$ and routing $\boldsymbol{R}^*$ policies
Execution: continuous in a time sequence $\tau = (0, \tau_0, 2\tau_0, \ldots)$ until termination;
1 $\tau \leftarrow 0$;
2 $term \leftarrow 0$; # *termination flag*
while $term = 0$ **do**
3 $i \leftarrow random\{\mathcal{N}_T\}$;
4 $proc \leftarrow random\{1,2,3,4\}$; # *select a random proc.*
5 Node i executes Procedure $proc$;
6 Update $g_i(\tau)$ through (17);
7 **if** $[g_j(\tau) = 0\,\&\,x_{ij}(i,\tau) = x_{ji}(j,\tau)\,\&\,p_j(\tau) = p_j(i,\tau),\ \forall i,j \in \mathcal{N}_T]$ **then**
     $term \leftarrow 1$;
   end
8 $\tau \leftarrow \tau + \tau_0$;
end

The nodes circulate messages with their variables in order to coordinate. For example, the final value of the data that node i pushes to node j should be equal to the data that node j decides to admits, i.e., $x_{ij}^* = x_{ji}^*$. The detailed description of the method is given in Algorithm 2 (JSR Max-flow). The algorithm is executed continuously and the nodes adjust their decisions in an asynchronous fashion.

Specifically, the nodes update their routing and store decisions in successive time instances, with very small distance of $\tau_0 > 0$ units, until the optimal solution is reached. In each time instance, a randomly selected node $i \in \mathcal{N}_T$ executes (randomly) one of the four *Procedures* 1 to 4 (for more details about the procedures please refer to [26]). Namely, each node i executes *Procedure* 1 to update its variables if the local surplus $g_i(\tau)$ is currently non-negative. Also, node i may execute *Procedure* 2 in order to notify its neighbors about its routing and store decisions as well as its dual variables, by sending certain *notification messages*.

Similarly, a node may select to execute *Procedure* 3 and coordinate its strategy with its neighbors, by exploiting the notification messages it has received from them. That is, each node i uses the latest notification messages it received and updates its local copies of the variables $p_j(i,\tau)$, $x_{ij}(i,\tau)$ and $x_{ji}(i,\tau)$ for each one of its neighbors. Finally, a node may update the store decisions using Procedure 4. Notice that each node k of graph $G = (\mathcal{N}, \mathcal{E})$ can execute all these tasks for each slot, i.e., for each one of its time instances $i = k^{(t)} \in \mathcal{N}_T$, obviously without requiring message passing among them.

The algorithm converges when the nodes have reached consensus about the JSR policy and the surplus $g_i(\cdot)$ is zero, i.e., when (17) is satisfied. Additionally, each node can initiate the execution of the algorithm if it detects local changes in its storage availability or the expected capacity variation patterns. In this case, the node can execute Procedure 1 that in turn will trigger the execution of Algorithm 2 from the affected nearby nodes. This algorithm has time complexity of $O(N_T^3)$, where $N_T = N \cdot T$ is the number of nodes in the time-expanded

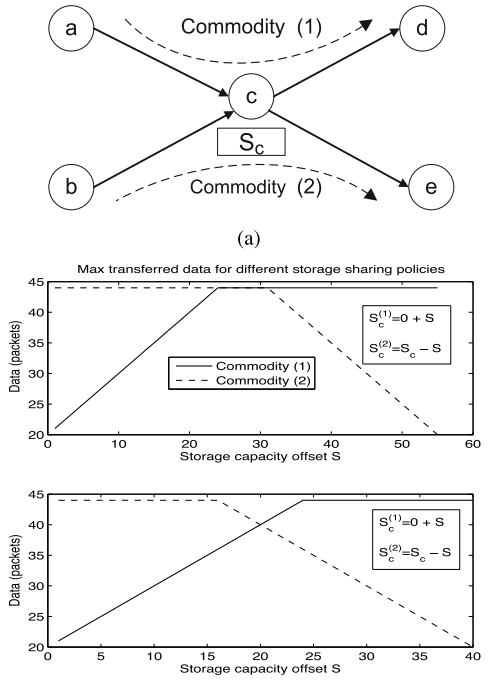

Fig. 6. Transfer of two commodities via node c which has storage capacity of S_c packets. Nodes (a, d) and (b, e) are the source-destination pairs for commodity (1) and (2) respectively. Link capacities vary $\boldsymbol{C}_{ac} \triangleq \boldsymbol{C}_{bc}$ =(10,12,14,2,2,4,6,0) and $\boldsymbol{C}_{cd} \triangleq \boldsymbol{C}_{ce}$ =(6,2,0,10,12,10,14,0). $S_c^{(1)}$ and $S_c^{(2)}$ is the storage capacity allocated to com. (1) and (2) respectively. (a) Two commodities flowing through node c. (b) Maximum amount of transferred data.

graph G_T. Therefore, its complexity with respect to the initial graph G is non-polynomial [31], since, in general, the time period T is not necessarily polynomial on the input size N.

V. Multicommodity JSR Policies

We will now extend our study to the case where data from two or more different sessions, which we call henceforth *commodities*, need to be transferred to respective destinations. We need to emphasize here that a commodity, in this work, represents flow aggregates where the aggregation is performed e.g., based on the application (and not individual flows). For example, in the context of ISP backbone networks, an application can be a bulk-data transfer, or streaming, or inter-data center communication from one PoP to another PoP.

Our goal is to derive the *Multicommodity JSR* policy (MJSR) that determines how much traffic of each commodity should be routed over each link and stored in each intermediate node, in every time slot. The different flows may have different delivery priorities due to their specific QoS requirements or respective SLAs (e.g., see [34]). We assume that the network is aware of these priorities and takes them into account in the respective resource allocation decisions.

A. Example of Storage Sharing Policy

Let us give at this point an example of storage sharing. Consider the network of Fig. 6(a) which conveys two commodities, denoted (1) and (2), with source and destination pairs (a, d) and (b, e) respectively. Both commodities flow through node c which has a certain storage capacity of S_c packets. For simplicity, we assume that $C_{ac}(t) = C_{bc}(t)$ and $C_{cd}(t) = C_{ce}(t)$, $t = 1, 2, \ldots, T$. In Fig. 6(b) we depict for each commodity the maximum amount of transferred data for various allocations of storage capacity S_c. We denote with $S_c^{(1)}$ the portion of storage that is allocated to commodity (1) and $S_c^{(2)}$ the respective storage capacity for commodity (2). Clearly, the total storage capacity constraint must be always satisfied, i.e., $S_c^{(1)} + S_c^{(2)} = S_c$.

For the upper plot we have assumed that $S_c = 55$. We observe that there is a set of storage capacity values, $S_c^{(1)}$ and $S_c^{(2)}$, for which the transferred data is maximized for both commodities. Clearly, in this case, where the storage resources are ample, it is straightforward to derive the optimal storage control policy.

On the contrary, for smaller values of S_c, the aggregate amount of data that need to be stored may exceed the storage capacity. Therefore, the storage sharing policy must determine how much storage capacity should be allocated to each commodity. This is depicted in the lower plot of Fig. 6(b), where $S_c = 40$. It can be seen that one cannot simultaneously maximize the transferred data for both commodities.

B. Multicommodity JSR Policy

We can define the multicommodity max-flow problem by extending the single commodity JSR problem. Specifically, we consider a set $\mathcal{W} = \{1, 2, \ldots, W\}$ of $W = |\mathcal{W}|$ commodities that are transferred over the same dynamic network G within a time period T. First, we introduce the respective multicommodity flow and storage control matrices. Specifically, we define $\boldsymbol{x}_{\mathcal{W}} = (x_{ij}^{(w)} : i \in \mathcal{N}_T, j \in F_i, w \in \mathcal{W})$, where $x_{ij}^{(w)} \geq 0$ denotes the amount of data of commodity w that is sent over link (i, j) with $i \triangleq k^{(t)}$ and $j \triangleq l^{(t+1)}$, $(k, l) \in \mathcal{E}$. Similarly, we define the matrix $\boldsymbol{y}_{\mathcal{W}} = (y_{in}^{(w)} : i \in \mathcal{N}_T, i \triangleq k^{(t)}, n \triangleq k^{(t+1)}, w \in \mathcal{W})$ where $y_{in}^{(w)} \geq 0$ denotes the amount of commodity w that is stored at node $k \in \mathcal{N}$ during time slot t, where $i = k^{(t)}$.

Accordingly, we define the *priority* parameter $r_w \geq 0$, $\forall w \in \mathcal{W}$. The larger the value of r_w is, the highest is the priority of commodity w. These parameters are determined by the network which is assumed to be aware of the specific needs of each commodity. Additionally, they can be selected based on certain service level agreements that the network (e.g., and ISP) has with his clients. The optimal MJSR policy is given by the solution of the following problem:

$$\max_{\boldsymbol{x}_{\mathcal{W}}, \boldsymbol{y}_{\mathcal{W}}} \sum_{w \in \mathcal{W}} r_w x_{ds}^{(w)} \tag{18}$$

subject to

$$\sum_{j \in F_i} x_{ij}^{(w)} + y_{in}^{(w)} = \sum_{j \in B_i} x_{ji}^{(w)} + y_{mi}^{(w)}, i \in \mathcal{N}_T \setminus \{s, d\}, w \in \mathcal{W} \tag{19}$$

$$0 \leq \sum_{w \in \mathcal{W}} x_{ij}^{(w)} \leq C_{ij}, \forall i \in \mathcal{N}_T, j \in F_i \tag{20}$$

$$0 \leq \sum_{w \in \mathcal{W}} y_{in}^{(w)} \leq S_i^{max}, \forall i, n \in \mathcal{N}_T \tag{21}$$

$$\sum_{j \in B_d} x_{jd}^{(w)} = x_{ds}^{(w)}, \ \sum_{j \in F_s} x_{sj}^{(w)} = x_{ds}^{(w)}, \ \forall w \in \mathcal{W} \tag{22}$$

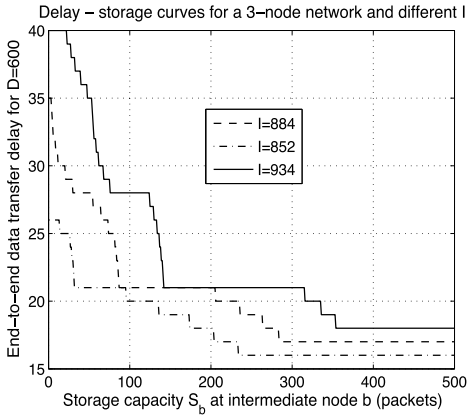

Fig. 7. Delay - Storage curves for the transfer of $D = 600$ data packets and various values of I, for network of Fig. 2.

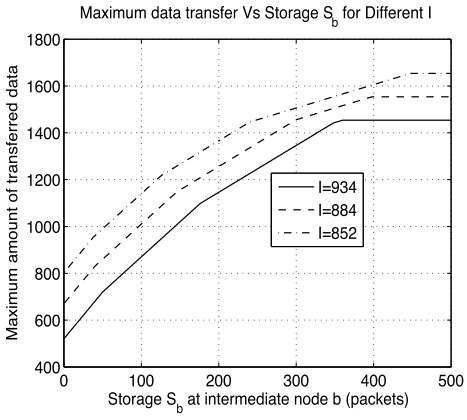

Fig. 8. Maximum amount D of transferred data in $T = 40$ time slots for the network of Fig. 2 for different intermediate storage S_b.

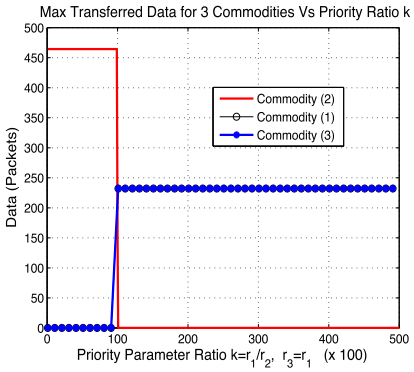

Fig. 9. Maximum amount of transferred data for 3 commodities and different priorities, for the network in Fig. 5.

Constraints (20) and (21) indicate that the link capacities and node storage must be shared among the different commodities.

The MJSR problem can be solved in a distributed fashion using a methodology as the one we presented for the JSR problem. For example, one can use a relaxation method where the constraints that couple the different flows will be relaxed and each node, for each commodity, will decide the routing and store policy based on the circulated dual variables. Due to the different priority parameters, the problems for the different commodities will admit different solutions regarding link capacity and storage capacity allocations. As it will be shown in the numerical results section, the priority parameters bias the routing and storage control decisions of the network. The higher r_w is, the more network resources commodity w will use and hence the larger will be the respective delivered amount of data.

VI. Performance Evaluation

In this section we provide numerical results, based on actual traffic traces, that verify the validity of our model and the applicability of our methodology. The performance metric is either the amount of data that can be transferred within a given time period or, equivalently, the incurred delay for the transfer of a certain amount of data from source to destination. We begin with the 3-node linear network of Fig. 2 the operation of which is described by (1)-(5). In Fig. 7(a) we depict the delay for the transfer of $D = 600$ data units from node a to c for different values of $I(G,T)$, and $T = 40$. As storage capacity S_b increases, the incurred delay reduces down to a minimum value. For example, for $I = 934$, storage addition $S_b = 360$ yields the minimum possible delay of $M = 17$ slots, which cannot be further improved even if one adds more storage. Also, Fig. 8 depicts the maximum amount of transferred data D, for a time period of $T = 40$ time slots from source to sink in the network of Fig. 2. We see that this quantity increases with the available storage S_b up to a certain point, and further increase in storage capacity does not improve the performance of the network. This maximum value depends on the dissimilarity index I of the links for the time period T.

Next, we consider the dynamic network of Fig. 5(a) and study the transfer of three commodities, denoted (1), (2), and (3) from node a to node f. Intermediate nodes b, c, d and e have storage capacity which is uniformly distributed in the interval $S^{max} \in [200, 400]$. Similarly, the link capacities are time varying and uniformly drawn from the interval $C_{ij} \in [100, 200]$. We study the network for a time interval of $T = 10$ slots. Our goal in this case is to investigate the impact of the priority parameters r_1, r_2, and r_3 on the amount that is transferred from each commodity, where we have set $r_1 = r_3$. These results are plotted in Fig. 9. We can see that as the ratio $k = r_1/r_2 = r_3/r_2$ increases, the transferred data of commodity (2) decrease and commodities (1) and (3) share the network resources.

In Figure 10 we show the topology of the operational backbone network of a large European ISP (Tier-1). These networks contain typically 10 to 20 central points of presence (PoP), e.g., aggregation nodes in large cities. Each one of the PoPs supports a set of smaller nodes (e.g., those of smaller cities) connected in a star topology. Please notice that there is link redundancy only in the backbone network and hence it is possible to optimize storage allocation and routing only in that part of the network. Such a Tier-1 ISP can also offer in-network virtualized storage and bandwidth elastic resources to third parties, e.g., CDNs, or even to end-users [29].

In Fig. 11 we studied the maximum possible performance benefits for the above network, in terms of end-to-end data transfer (normalized values are used). To understand the importance of the resource elasticity assume that the capacity variation pattern (or, in general, the pattern of the available

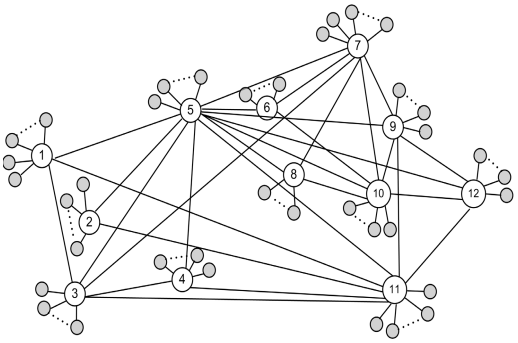

Fig. 10. An operational European ISP backbone network of 12 PoPs.

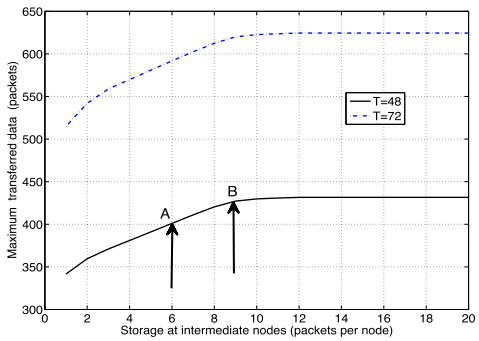

Fig. 11. Benefits from additional storage reservation for the network of Fig. 10. The different lines in each plot correspond to different time windows, $T = 48$ and $T = 72$ slots. Points A and B correspond to the initial and the subsequent reservations of in-network storage, respectively.

bandwidth) is initially such that the optimal total storage allocation according to the JSR algorithm is in point A. Now, assume that after few time windows T, the links capacity availability changes and the updated JSR solution dictates that it is necessary to reserve more storage. Due to the storage elasticity, it is possible to update the policy and shift the system to point B. This yields a performance improvement of the order of 12% as can be seen in Fig. 11. Also, it is worth noticing that the longer is the time window over which the JSR optimization is realized, the larger is the benefit from in-network storage. In NFV systems such additional information can be exploited dynamically, i.e., whenever becomes available.

Finally, in Fig. 12 we present actual traffic traces from two links (called A-B, and B-C) of the network in Fig. 10, measured over a period of 15 days and with a slot window of 5-minutes. Due to confidentiality issues, we present only the normalized traffic volumes; we give in the sequel more details on the order of magnitude for these quantities. Let us focus first on the upper subfigure. Based on the actual link utilization, the volume of the (normalized) traffic that is actually transferred over this link is $D_1 = 11,665 \cdot 10^3$ data units. However, if we exploit the off-peak traffic windows of this link, then we can transfer additional $9,945 \cdot 10^3$ units. Therefore, the usage of storage in the origin node of this link can achieve an 85% increase of the volume of the transferred data, over a period of 15 days. Clearly, these numbers refer to the maximum possible data transfer increase, and are only achievable if the incoming links to the node have highly diverse traffic patterns. In most cases, we expect that the

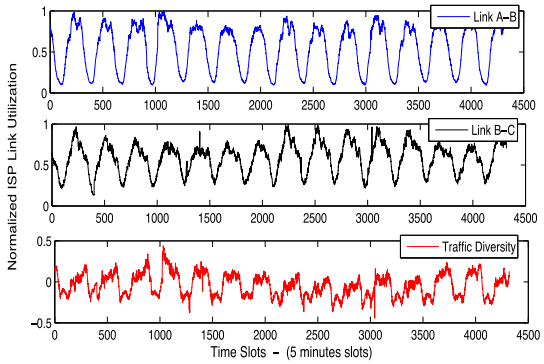

Fig. 12. Upper two subplots: normalized traffic (utilization) in two successive links of a large European ISP. Lower subplot: their diverse traffic evolution over time, which captures the amount of data that can be stored and pushed within a time interval of 15 days.

performance benefits will be smaller, due to the correlation of the traffic variation patterns of the different links.

For example, if we consider the traffic pattern of a successive link (in the network of Fig. 10) (link B-C), then we can plot the difference between the traffic of the two links. In case there is available storage capacity in node B, then we can increase the amount of data that can be conveyed from node A to C. Namely, we see that in-network storage in this case can achieve an end-to-end data transfer increase of 10.04% compared to the scenario with no storage at node B. Typical links that connect the different points-of-presence (PoP) of large ISPs have capacity of 1Gbps and more often of 10 or 40Gbps [37]. Therefore, a data transfer increase of this order (for the 15-days time interval) corresponds to 16,686 Gbytes for the 1Gbps links, to 166,860 Gbytes for the 10Gbps links, and to 667,440 Gbytes for the 40Gbits links.

VII. Related Work

Storage has been considered in wireless networks in the context of Delay Tolerant Networks (DTN) [21] and more recently, in vehicular networks [35]. In these cases data is stored at intermediate nodes and is transmitted whenever required links are available, e.g., see [36] and references therein. The objective is to guarantee delivery of packets, and storage is used whenever routing is not feasible. On the contrary, we find the storage placement and control policies in order to reduce data transfer delay.

Another instance of storage-assisted networking is presented in [13] and [37]. The authors consider tandem networks that span across large areas, and derive store-and-forward policies for cost-efficient transfer of bulk data by exploiting already-paid-for bandwidth. This methodology has been also applied to datacenters [16] serving requests that follow strong diurnal patterns [4]. In all these cases, the benefits arise due to the periodicity of the users' demand and the fact that the links are dimensioned based on peak traffic values. In this class of works storage allocation is fixed and solutions are centralized.

In more abstract modeling terms, [38] presents a centralized algorithm that yields the minimum delay flow for a certain time period, with given link capacity patterns. The requirement for complete information is relaxed in [39] where the network is described by a set of stochastic processes, one for each link,

with known state space. The objective is to find the shortest path for the delivery of a packet to the destination. It is an online problem which the authors prove that is intractable. Finally, an interesting survey of flow algorithms in dynamic networks with storage-capable nodes is provided in [31].

At item level, i.e., file or file chunk, there exist various caching policies which identify item placement at node caches based on their popularity so as to minimize item retrieval time. For example in [17] a wireless network architecture of distributed caching is used to satisfy requests while consuming low backhaul bandwidth. Additionally, in content centric networking in-network storage affects the content availability and the network performance in terms of content retrieval delay [40]. Similarly, other recent network architectures like the publish/subscribe systems [41], or peer-to-peer video on-demand solutions [42] make use of in-network storage. All these different classes of application scenarios reveal the important role of node storage and further motivate our analytical study of storage-assisted networking.

In an even smaller scale the class of Backpressure-based dynamic routing Policies (BP) [43], [44] study queues which are orders of magnitude less than the mass storage we consider here. BP policies focus on the buffer-level traffic dynamics and extract the capacity region of the network. On the contrary, we focus on storage policies at the flow level, and require the knowledge only of the average aggregate traffic in each link, which most often follow a known periodic pattern. For these dynamics we try to assess the storage amounts and the routing that achieve the maximum end-to-end data transfer capacity. Our methodology is closely related to the network utility maximization framework [24] that has been extensively applied in communication networks [11]. Yet, we depart significantly from previous works as we present a novel model for the joint optimization of storage and link capacity, and employ the ϵ-relaxation algorithm that can be executed, when required, in a distributed and asynchronous fashion [26]. This is particularly important especially for large networks, since it allows fast calculations of the JSR policy, is robust and adaptive to small network changes, and lightweight in terms of communication overheads.

Joint storage control and routing algorithms are gaining increasing attention from industry. This is best exemplified by the recent decisions of many telecommunication companies to deploy their own content distribution networks, as L3 did [45], or collaborate closely with CDNs, as AT&T [46] and Swisscom [47] did with Akamai. At the same time, the emering paradigm of NFV systems [9], [32] and related products [6], [7], [8] which can support fast storage allocation policies, render our approach of high practical importance. Namely, our work provides a solid framework for jointly allocating storage and bandwidth capacity in such systems by exploiting their resource elasticity. Despite the proliferating works about the technical challenges in NFV systems, e.g., [48], [49], this allocation problem has been not studied before.

VIII. Conclusions

In this work we explored under what conditions and in what extent in-network storage can improve the data transfer capability of communication networks. The optimal storage allocation policy is the one that guarantees maximum benefit from storage use and can be derived for every network using the presented Storage Capacity Allocation algorithm. In order to realize this benefit, storage must be considered in conjunction with routing. The joint storage control - routing (JSR) policy can be derived through the solution of a single- or multicommodity max-flow problem defined over a time-expanded graph. At the same time, JSR policies utilize the minimum necessary storage capacity. The presented policies can be derived (and implemented) through optimization algorithms that are amenable both to centralized and to distributed execution. This is very important especially for large-scale systems.

The proposed framework extends the network utility maximization methodology and provides a unified treatment of storage-assisted networking. It can be applied to small or larger ISPs, datacenter networks or even to cellular networks. In all these cases, it is desired to work with links with high aggregation of traffic and monitoring data that are frequently updated. A variety of different products are nowadays commercially available to support such architectures, and related forward-looking solutions for network functions virtualization have already been supported and deployed by major market players. Clearly, the emerging paradigm of NFV offers a fertile ground for the application of these resource allocation methods. Moreover, this work opens many interesting research directions. For example, one could study how storage can be used to optimize network performance with respect to other criteria such as the energy consumption per transferred byte.

References

[1] *Mobility Report: On the Pulse of Networked Society*, Ericsson, Stockholm, Sweden, 2014.

[2] Cisco, "Cisco visual networking index: Global mobile data traffic forecast," Cisco, San Jose, CA, USA, White Paper 1454457600805266, Feb. 2016.

[3] *AT&T to Pay* 1.93 *Billion for Mobile Spectrum*, Bloomberg News, New York, NY, USA, 2010.

[4] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, 2009.

[5] Internet Engineering Task Force (IETF). (2011). *A Survey of In-Network Storage Systems*. [Online]. Available: http://tools.ietf.org/html/rfc6392

[6] *Why is This Storage Guy Getting so Caught up in NFV?* Hewlett-Packard, Palo Alto, CA, USA, Mar. 2014.

[7] Hewlett Packard. (Nov. 2014). *HPE 3PAR StoreServ Storage. Primary Storage Architecture Products*. [Online]. Available: http://www8.hp.com/uk/en/products/data-storage/3parstoreserv.html

[8] NetApp. (2013). *Flexpod Solutions*. [Online]. Available: http://www.netapp.com

[9] ETSI, European Telecommunications Standards Institute, "Network functions virtualization: Use cases," White Paper GS NFV 001, V.1.1.1, Oct. 2013.

[10] G. Iosifidis, I. Koutsopoulos, and G. Smaragdakis, "The impact of storage capacity on end-to-end delay in time varying networks," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1494–1502.

[11] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1439–1451, Aug. 2006.

[12] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

[13] N. Laoutaris and P. Rodriguez, "Good things come to those who (can) wait: Or how to handle delay tolerant traffic and make peace on the Internet," in *Proc. 7th ACM HotNets*, 2008, pp. 1–6.

[14] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! datasets," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1620–1628.

[15] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft, "Structural analysis of network traffic flows," in *Proc. ACM SIGMETRICS*, 2004, pp. 61–72.

[16] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *Proc. ACM SIGCOMM*, 2011, pp. 74–85.

[17] N. Golrezaei, K. Shanmugam, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless video content delivery through distributed caching helpers," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1107–1115.

[18] S. Woo *et al.*, "Comparison of caching strategies in modern cellular backhaul networks," in *Proc. ACM MobiSys*, 2013, pp. 319–332.

[19] K. Poularakis, G. Iosifidis, A. Argyriou, and L. Tassiulas, "Video delivery over heterogeneous cellular networks: Optimizing cost and performance," in *Proc. IEEE INFOCOM*, Apr./May 2014, pp. 1078–1086.

[20] The Guardian. (Sep. 2014). *iOS 8 Causes Bandwidth Spikes Nationwide, Despite Slow Uptake*. [Online]. Available: http://www.theguardian.com/technology/2014/sep/18/apple-ios-8-bandwidth-spikes-uk-uptake

[21] A. Krifa, C. Barakat, and T. Spyropoulos, "Optimal buffer management policies for delay tolerant networks," in *Proc. IEEE SECON*, Jun. 2008, pp. 260–268.

[22] M. Roughan *et al.*, "Experience in measuring backbone traffic variability: models, metrics, measurements and meaning," in *Proc. IMW*, 2002, pp. 91–92.

[23] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proc. ACM IMC*, 2009, pp. 35–48.

[24] F. P. Kelly, "Charging and rate control for elastic traffic," *Eur. Trans. Commun.*, vol. 8, no. 1, pp. 33–37, 1997.

[25] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ, USA: Princeton Univ. Press, 1962.

[26] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont, MA, USA: Athena Scientific, 1997.

[27] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, MA, USA: Athena Scientific, 1998.

[28] M. Stoer and F. Wagner, "A simple min-cut algorithm," *J. ACM*, vol. 44, no. 4, pp. 585–591, 1997.

[29] R. Stoenescu *et al.*, "In-Net: In-network processing for the masses," in *Proc. ACM EuroSys*, 2015, Art. no. 23.

[30] B. Frank *et al.*, "Pushing CDN-ISP collaboration to the limit," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 2, pp. 34–44, 2013.

[31] B. Kotnyek, "An annotated overview of dynamic network flows," INRIA, Rocquencourt, France, Tech. Rep. 4936, 2003.

[32] Hewlett Packard. (Nov. 2014). *HP Network Functions Virtualization (NFV)*. [Online]. Available: http://www8.hp.com/us/en/cloud/nfv-overview.html?

[33] A. Ben-Tal and A. Nemirovski, "Robust solutions of linear programming problems contaminated with uncertain data," *Math. Program.*, vol. 88, no. 3, pp. 411–424, 2000.

[34] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.

[35] F. Malandrino, C. Casetti, C.-F. Chiasserini, and M. Fiore, "Optimal content downloading in vehicular networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 7, pp. 1377–1391, Jul. 2013.

[36] S. Gitzenis, G. Konidaris, and S. Toumpis, "Flow optimization in delay tolerant networks using dual decomposition," in *Proc. WiOpt*, 2012, pp. 444–451.

[37] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram, "Delay tolerant bulk data transfers on the Internet," in *Proc. ACM SIGMETRICS*, 2009, pp. 229–238.

[38] R. G. Ogier, "Minimum-delay routing in continuous-time dynamic networks with piecewise-constant capacities," *Networks*, vol. 18, no. 4, pp. 303–318, 1988.

[39] A. Orda, R. Rom, and M. Sidi, "Minimum delay routing in stochastic networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 2, pp. 187–198, Apr. 1993.

[40] G. Carofiglio, M. Gallo, and L. Muscariello, "Bandwidth and storage sharing performance in information centric networking," in *Proc. ACM SIGCOMM Workshop ICN*, 2011, pp. 26–31.

[41] M. Diallo, S. Fdida, V. Sourlas, P. Flegkas, and L. Tassiulas, "Leveraging caching for Internet-scale content-based publish/subscribe networks," in *Proc. IEEE ICC*, Jun. 2011, pp. 1–5.

[42] K. Lee *et al.*, "An optimized distributed video-on-demand streaming system: Theory and design," in *Proc. 50th Allerton Conf.*, 2012, pp. 1347–1354.

[43] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

[44] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, no. 1, pp. 1–144, 2006.

[45] *Level3 CDN*, accessed on Nov. 30, 2016. [Online]. Available: http://www.level3.com/en/products/content-delivery-network/

[46] *Akamai and AT&T Forge Global Strategic Alliance to Provide Content Delivery Network Solutions*, Akamai Press Releases, Cambridge, MA, USA, Dec. 2012. [Online]. Available: https://www.akamai.com/us/en/about/news/press/2012-press/akamai-and-att-provide-content-delivery-network.jsp

[47] *Swisscom and Akamai Enter Into a Strategic Partnership*, Akamai Press Release, Berne, Switzerland, Mar. 2013. [Online]. Available: https://www.akamai.com/us/en/about/news/press/2013-press/swisscom-and-akamai-enter-into-a-strategic-partnership.jsp

[48] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. USENIX, NSDI*, 2014, pp. 459–473.

[49] J. Sherry *et al.*, "Rollback-recovery for middleboxes," in *Proc. ACM SIGCOMM*, 2015, pp. 227–240.

George Iosifidis received the Diploma degree in telecommunications engineering from the Greek Air Force Academy in 2000, and the M.S. and Ph.D. degrees in electrical engineering from the University of Thessaly in 2007 and 2012, respectively. He was a Post-Doctoral Researcher with CERTH, Greece, and Yale University, USA. He is currently the Ussher Assistant Professor in Future Networks with Trinity College Dublin, and a Funded Investigator with CONNECT Center, Ireland. His research interests lie in the broad area of wireless networks and network economics.

Iordanis Koutsopoulos received the Diploma degree from the National Technical University of Athens, Greece, in 1997, and the M.S. and Ph.D. degrees in from the University of Maryland, College Park, MD, USA, in 1999 and 2002, respectively, all in electrical and computer engineering. He was a Lecturer from 2005 to 2010 and an Assistant Professor from 2010 to 2013 with the Department of Computer Engineering and Communications, University of Thessaly, and an Assistant Professor with the Department of Informatics of Athens University of Economics and Business (AUEB) from 2013 to 2016. He is currently an Associate Professor with the Department of Informatics, AUEB. His research interests involve network control and optimization in wireless networks, social and community networks, crowd-sensing systems, smart-grid, and cloud computing. He received the Single-Investigator European Research Council Competition runner-up Award for the Project RECITAL: Resource Management for Self-coordinated Autonomic Wireless Networks (2012–2015).

Georgios Smaragdakis received the Diploma degree in electronic and computer engineering from the Technical University of Crete, and the Ph.D. degree in computer science from Boston University in 2009. From 2008 to 2014, he was a Senior Researcher with Deutsche Telekom Laboratories and TU Berlin. In 2008, he was a Research Intern with Telefonica Research. He is currently a Professor with Technical University (TU) Berlin and a Researcher with the Massachusetts Institute of Technology and Akamai Technologies. His research interests include the measurement, performance analysis, and optimization of content distribution systems on the Internet, economic, peering, collaboration, and policy aspects of content delivery, and Internet, Web, and content delivery analytics. He received the European Research Council Starting Grant Award (2015), a Marie Curie International Outgoing Fellowship (2013), and the Best Paper Award at the ACM IMC (2011 and 2016) and the ACM CoNEXT (2015).