The Impact of Storage Capacity on End-to-End Delay in Time Varying Networks

George Iosifidis and Iordanis Koutsopoulos

Computer & Communications Engineering Dept.
University of Thessaly, Volos, Greece
Centre for Research & Technology Hellas, Greece

Georgios Smaragdakis

Deutsche Telekom Laboratories
Technical University of Berlin, Germany

***Abstract*—Recent technological advances have rendered storage a cheap and at large scale available resource. Yet, there exist only few examples in networking that consider storage for enhancing data transfer capabilities. In this paper we study networks with time varying link capacity and analyze the impact of node storage on their capability to convey data from source to destination. We show that storage capacity is quite beneficial in terms of the amount of data that can be pushed from the source to the destination within a given time horizon. Equivalently, storage can be used to reduce incurred delay for the delivery of a certain amount of data. For linear networks, we show that this performance improvement depends on the relative patterns of link capacity variations. We extend our study to general networks and we use a novel method that iteratively updates the minimum cut of the time expanded graph, in a constructive manner, in the sense that during the process, the storage capacity allocation in the network is shown. Next, we incorporate routing in our methodology and derive a joint storage capacity management and routing policy to maximize the amount of data transferred to the destination. This policy stems from the solution of the maximum flow problem defined for the dynamic network over a certain time period, by using the ϵ-relaxation solution method. The later is amenable to distributed implementation, which is a very desirable property for the large scale modern networks which operate without central control.**

I. Introduction

The last few years we are witnessing an unprecedent growth of demand for ubiquitous and high speed networking around the world. High-tech communication devices and novel traffic demanding applications generate a large amount of data that must be routed efficiently at minimum cost and within the minimum possible time. At the same time, recent technological advances have made storage a readily affordable resource. Nowadays storage is cheap compared to bandwidth, [1], and with least space and power requirements. Hence, it can be used both in small portable devices and in large amounts located at central communication nodes of backbone networks. In light of these observations, it is challenging to study the performance benefits of storage in terms of the amount of data that can be conveyed to the destination, to identify possible methods for exploiting storage and to characterize the performance improvements and conditions under which these benefits are enlarged.

Before attempting to address the above issues, let us first present a simple example to motivate the benefit of storage capacity. Consider the 3 nodes linear network of Figure 1 and assume a time slotted operation. The link capacities (packets/sec) change every 2 time slots according to a periodic pattern, $\{C_{AB}(n) = D,\ C_{BC}(n) = 1\}$, $\{C_{AB}(n+2) = 1,\ C_{BC}(n+2) = D\}$, with $n = 1, \ldots, T-2$, $D > 1$ and remain constant in between. Transmission delay over each link is considered to be equal to the slot duration, T_0 time units. In this setting, we ask the question: How much time is required to convey an amount of D packets of data from node A to node C in case (i) no node has storage, (ii) node B has storage of $S_B > D$ packets? The answer is straightforward, yet very illuminating. In the first case, node A can push in each time slot n only as much data as node B is able to forward in the immediately next time slot $(n+1)$, i.e. $C_{BC}(n+1)T_0$. Hence, given the capacity pattern above, the required time for data transfer is $(D+1)$ slots, i.e. $(D+1)T_0$ time units. However, when $S_B > D$, node A pushes up to $C_{AB}(n)T_0$ data and the excess amount that cannot be immediately routed to the destination C, i.e. $(C_{AB}(n) - C_{BC}(n+1))T_0$ is stored in node B. In the subsequent slot $(n+2)$, when C_{BC} is high, stored data along with the new arrived data from A is finally delivered to the sink. Therefore, in this case the required time for the delivery of D units of data is only 2 time slots.

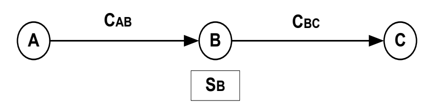


Fig. 1. A tandem (linear) network of 3 nodes. Node A is the source, node C the sink and S_B is the available storage capacity at node B.

Clearly, the use of storage significantly reduces the incurred end-to-end delay for the data transfer and in this sense improves the performance of the network. Equivalently, 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. In other words, in this example node storage is actually used as a special type of link capacity and augments the average end-to-end network capacity. However, it can be inferred from the previous observations that the benefit from storage utilization depends on relative link capacity values. Namely, the link capacities should vary with time and moreover their variation patterns should be diverse, i.e. they should reach upper and lower bounds in a non synchronized fashion. Link capacity variations and link failures are common in contemporary information networks such as peer-to-peer overlays and ad hoc networks. Additionally, even in backbone networks the available link capacity for an operator varies due to variations in the traffic of flows of other operators that traverse the same link. Such traffic variations are beyond the control of the operator. Finally, more often than not, time varying pricing schemes are employed to regulate traffic and result to the deployment of networks with correspondingly time varying link capacities, [2].

In this work we consider dynamic networks and analyze the impact of node storage on their capability to convey data from source to destination. We use the technique of time-expanded graphs [3], which map the time evolution of dynamic networks, within a specific time period, to ordinary graphs. In these graphs, storage is modeled by defining special type of links connecting different time instances of the same node. The main issue is to find the min-cut of the expanded graph. This represents the upper bound for the amount of data transferred in the dynamic network for the given time period. Under certain conditions, this bound can be increased by utilizing storage links. We are interested in identifying these conditions and devise the optimal storage management policy, that is, decide at which node, at which time instant and how much information to store, and how much to forward further to the next node. Moreover, we go one step beyond and study methods to achieve this upper bound. We formulate the max flow problem on the time-expanded graph and derive the optimal routing and storage policies. The max flow problem can be solved in polynomial time using the ϵ-relaxation method which is also amenable to distributed implementation. This last feature enables the derivation of *distributed* joint storage and routing protocols for networks with time varying link capacities.

In summary, our contributions are as follows: (i) we define the storage assisted network performance enhancement problem described above and designate the conditions under which storage utilization is beneficial for linear networks, (ii) we extend our analysis in general networks and provide a methodology for finding the storage enhanced min-cut and the optimal storage policy, (iii) we propose the conjunction of storage with routing and define the joint storage management and routing problem for a single commodity, and (iv) we provide a distributed method for its solution. The rest of the paper is organized as follows. In Section II we discuss related works, in section III we analyze the performance of linear networks with storage capable nodes and in section IV we introduce the optimal storage management policy for general networks. In section V we introduce the joint storage - routing policy as a solution to a max flow problem and in section VI we discuss the importance of the available information about the future network state. Finally, in VII we present numerical results that verify our analysis and in section VIII we conclude our study.

II. Related Work

Storage has been considered in wireless networks in the context of Delay Tolerant Networks (DTN) [4], in order to alleviate intermittent connectivity problems between the source and the destination. In these networks often there exist no permanent end-to-end paths and therefore traditional routing algorithms fail. Hence, various Store and Forward (SnF) policies are employed to circumvent the issue above. Data is stored in intermediate nodes and is transmitted whenever required links are available. Apparently, available information about current and future state of the network determines the performance of these SnF strategies, [5]. However, in this class of problems the objective is to guarantee delivery of packets and storage is used as a strategy whenever routing is not possible.

In a similar context, the transfer of delay tolerant bulk data was introduced in [6], and [2]. The authors consider tandem (linear) wireline networks where intermediate nodes have storage capability. The objective is to achieve the transmission of large amounts of data with minimum monetary cost under certain pricing schemes. The method is extended in [7], where the authors study general network graphs with time varying but known in advance link capacities. Node storage varies with time in terms of both capacity and cost. It is explained that through the combination of time-expanded graphs and flow optimization techniques, a centralized solution provides the optimal (minimum-cost) transfer of data.

In more abstract modeling terms, the problem of minimum delay routing in networks with node storage is considered in [8], and [9]. In detail the work [8], studies a single commodity dynamic network with varying capacities and presents a centralized algorithm for deriving an *earliest arrival* flow over a time period T. This flow maximizes the amount of data that reaches the sink for every τ, $\tau \in (0, T)$ and hence it is a minimum delay flow. It is assumed that there exists full knowledge about the network state evolution for the specific time period and this information is used to construct the time-expanded graph. The requirement for future knowledge is relaxed in [9], where the link state represent the incurred delay for a data packet transfer. The network is modeled through a stochastic formulation with known state space and empirically calculated state transition probabilities. The objective is to find the shortest path for the delivery of a packet to the destination. In this setting routing is an adaptive policy and storage is considered simply as an additional routing option. The authors prove that this problem is intractable in general.

Finally, other works have also studied flow algorithms in dynamic networks with storage capable nodes, [10] (and references therein). Among them [11], presents an interesting negative result according to which node storage does not improve minimum cost flows over time. However, this result refers to networks with constant link capacities through time. The above works do not explicitly study whether storage actually contributes and under what conditions in the performance of the networks. On the contrary, in this paper we both identify the conditions that render storage use beneficial and at the same time propose methods for achieving this improvement. Until now, storage has been considered mainly in networks where delay was not the main performance criterion, i.e. delay tolerant wireless or wireline networks. Instead, here we analyze how storage can reduce the incurred delay. Storage is considered an additional resource that should be managed efficiently in conjunction with link capacity. Moreover, we devise centralized and distributed algorithms that ensure the maximum possible performance improvement within a certain time period. The methods are of particular interest in contemporary networks which exhibit link capacity variations.

III. Impact of Storage Capacity in Linear Networks

We start from linear networks where routing options for every node are confined to its downstream (next hop) neighbor. In this case, the storage benefit can be directly calculated even for arbitrary link capacity variation patterns. For example, consider again the 3 node network of Figure 1, where links may change every 2 time slots, with slot duration T_0. We set the following questions: (i) what is the incurred delay to deliver an amount of D units of data? and (ii) how much data can we deliver from node A to node C in a certain time period of T time slots, if node B has storage capacity available?

When node B has zero storage the network end-to-end capacity at every time slot n due to the flow conservation constraint is:

$$C_{AC}(n) = \min\{C_{AB}(n), C_{BC}(n+1)\} \tag{1}$$

Hence, the required time for the transfer of D units of data from node A to node C is $Del = ZT_0$, where Z is the minimum integer for which it holds:

$$\sum_{n=1}^{Z} C_{AC}(n)T_0 \geq D \tag{2}$$

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

$$D = \sum_{n=1}^{T-2} C_{AC}(n)T_0 \tag{3}$$

On the other hand, when node B can store data, the network end-to-end storage enhanced capacity is:

$$C_{AC}^{S_B}(n) = \min\{X_B(n+1), C_{BC}(n+1)\} \tag{4}$$

where $X_B(n+1)$ is the available data at node B at time slot $(n+1)$. This amounts to the instant capacity of the link (A, B) and the accumulated stored data in node B, Y_B. That is:

$$X_B(n+1) = C_{AB}(n) + Y_B(n+1)/T_0 \tag{5}$$

where $Y_B(n+1) = \max\{Y_B(n) + [C_{AB}(n) - C_{BC}(n+1)]T_0, 0\}$ is always nonnegative and upper bounded by node B maximum storage $S_B > 0$, and it is $Y_B(0) = 0$.

Clearly, $C_{AC}^{S_B}(n) \geq C_{AC}(n)$ for every time slot n. This means that the use of storage at intermediate node B does never deteriorate the network performance and in many cases it significantly improves both the delay and the maximum amount of transferred data. The exact improvement depends on the variation pattern of link capacities and specifically on the relative values of C_{AB} and C_{BC} across all time slots. The more diverse the capacity value sequences are, the larger is the benefit from the storage usage. We define the *Dissimilarity Index* to quantify the variability in link capacity patterns for a certain time period T, as follows:

TABLE I
Example for 3-node Network, $T_0 = 1$, $L = 24$, $S_B = 30$

Slot n	C_{AB} (p/sec)	C_{BC} (p/sec)	C_{AC} (p/sec)	D (p)	Y_B (p)	C_{AC}^S (p/s)	D^S (p)
1	10	6	2	0	0	2	0
2	12	2	0	2	8	0	2
3	14	0	10	2	20	10	2
4	2	10	2	12	24	12	12
5	2	12	2	14	14	8	24
6	4	10	4	16	6	4	32
7	6	14	-	**20**	0	6	**36**

$$L = \sum_{n=1}^{T} \min\left\{J(n), \max\sum_{i=1}^{n-1}(U(i) - J(i)), 0\right\} \tag{6}$$

where

$$U(i) = \max\{[C_{AB}(i) - C_{BC}(i+1)]T_0, 0\}$$

and

$$J(i) = \max\{[C_{BC}(i+1) - C_{AB}(i)]T_0, 0\}$$

In other words, L simply measures the aggregate amount of data that was stored in previous time slots and released in some subsequent slot within the time period T, assuming that there is no storage space constraint.

This parameter actually reveals the conditions under which storage is beneficial. Namely, when it holds $L = 0$, storage does not improve the network performance. This is realized when the capacities of the 2 links are equal at every time slot, $C_{AB}(n) = C_{BC}(n+1)$ or if the link (A, B) has always lower capacity, i.e. $C_{AB}(n) < C_{BC}(n+1)$, $\forall n$. In this case no data is ever stored in node B. Moreover, even if there exist accumulated stored data it might be impossible to push it further if link (B, C) is the network bottleneck in each time slot, i.e. $C_{BC}(n+1) < C_{AB}(n)$, $\forall n$. In Table I we present a numerical example for this network and demonstrate the benefit of storage utilization at node B. Capacities are measured in packets/sec (p/s) while data and storage in packets (p). Analogous results hold for every linear network with more than 3 nodes where we can define *Del*, *D* and *L* in a similar fashion.

From the analysis above, we infer that it is possible to exploit the diverse evolution of the links capacities and use intermediate nodes' storage to augment the end-to-end network capacity within a certain time horizon. Notice that in linear networks storage policy is very simple. Each node accepts all the incoming traffic and stores the excess data that it cannot forward in the current time slot so as to exploit possible capacity increase in the subsequent slots. The more the available storage is at intermediate nodes, the more we benefit from its use for certain

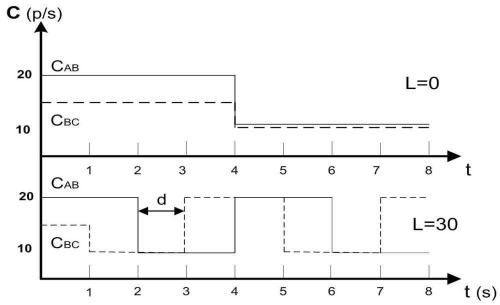


Fig. 2. Dissimilarity Index, L, for two different capacities variation patterns. The bottom plot represents the case where the links are as much diverse as possible, reaching upper and lower bounds in an antisymmetric fashion. The parameter d represents the link traversal time.

Algorithm 1 (Storage Policy - SP)

(**Step 0**) Find the current min-cut $Q = [W, N \setminus W]$ and calculate its capacity $C(Q)$.
(**Step 1**) If there exists a node i of the original network G for which it holds that $i^{(t)} \in W$ and $i^{(t+1)} \in (N \setminus W)$, then add a link (*storage link*) connecting these two nodes and go to Step 2. Else, go to Step 4.
(**Step 2**) Increase the new storage link capacity S_i^t as much as required so as to render Q a non minimum cut. This increase is bounded by the maximum available node storage. If Q becomes a non minimum cut go to Step 3. Else (Q is still the min-cut) go to Step 4.
(**Step 3**) Find the new min-cut $Q' = [W', N \setminus W']$. Set $Q = Q'$ and go to Step 1.
(**Step 4**) Set $Q_f = Q$. The current min-cut Q_f gives the maximum storage enhanced average capacity for the initial network G over time period T.
(**Step 5**) Locate all storage links that do not belong to the final min-cut Q_f and decrease their capacity as long as Q_f remains unchanged. The algorithm terminates.

values of L. However, unlike linear networks, determining the storage management policy in general graphs is a complicated task and requires the knowledge of the capacity evolution for all network links. In the next section we consider general networks with known capacity variation patterns and provide a method for deriving the optimal storage management policy.

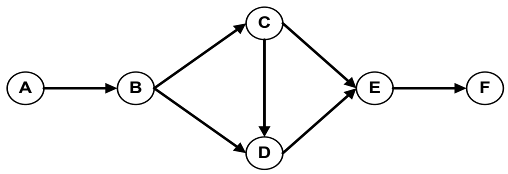
IV. Storage Management Policy for Networks with Known Capacity Variation Patterns

For many networks it is possible to know in advance or predict with precision the future values of link capacities. For example, small scale networks with predictable capacity evolution such as networks of satellites, [8], fall within this class. Another scenario is networks with constant link capacities but periodic time varying traffic patterns where we attempt to exploit residual capacity, [2]. Additionally, very often the network administrator determines himself the links capacity schedule for a certain time period. We provide here a method for devising the optimal **storage management policy**, i.e. the policy that leads to maximum possible benefit from storage. Recall that the performance of a network is bounded by the capacity $C(Q_{min})$ of the minimum cut Q_{min} of its graph, [3]. This represents the maximum flow that can be delivered from the source to the sink. 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 minimum cut, we increase the maximum amount of data that can be transferred to the destination.

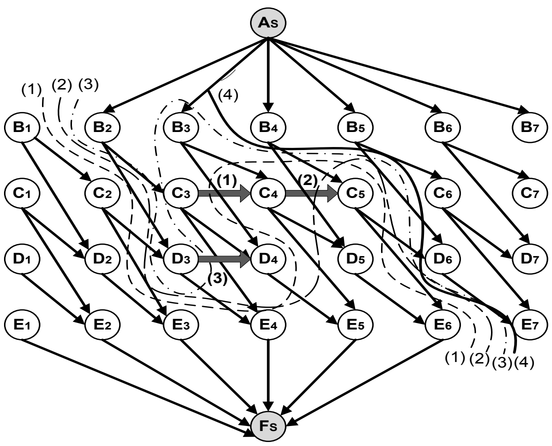
Consider a directed network graph $G = (V, E)$, with $N = |V|$ nodes and $H = |E|$ links. The network is dynamic, i.e. every link capacity $C_{ij}(t)$, $(i, j) \in E$, changes with time according to a predefined pattern. We assume a time slotted operation, $t = 1, 2, \dots, T$, with slot duration T_0. Link capacities remain constant within each time slot and initially no node has storage capability. We assume that the traversal time is identical for all links and equal to the slot duration. Using the technique of time-expanded graphs, [3], we construct the corresponding static network G_T for T slots. The transformation is simple. For every node $i \in V$ of the original network G we add T nodes in the G_T, $i^{(t)}$, $t = 1, \dots, T$. Moreover, for every arc $(i, j) \in E$ of G, we add a set of corresponding arcs $(i^{(t)}, j^{(t+1)})$, $t = 1, \dots, T-1$. Finally, we substitute all the time instances of the source and the destination nodes with 2 super-nodes for ease of presentation.

The graph G_T incorporates the notion of time and is used in order to analyze the properties of G for the time period T. Namely, the amount of data that can be transferred by G, within horizon T, is upper bounded by the minimum cut of G_T. Here, we propose the increase of this min-cut capacity by the addition of specific links, the *storage links*. The rationale of the method is visualized in Figure 3. In detail, assume that $Q = [W, N \setminus W]$ is the initial min-cut of G_T with capacity $C(Q)$, where W is the set of nodes in which the source node belongs and $(N \setminus W)$ the set containing the sink node. The critical observation is the following. If there exists a node $i \in G$ such that $i^{(t)} \in W$ and $i^{(t+1)} \in (N \setminus W)$ then we can increase the network capacity by connecting $i^{(t)}$ and $i^{(t+1)}$ with a link of capacity $S_i^{(t)}$. This connection amounts to adding storage of $S_i^{(t)}$ units to node i and using it during time slot t. With the addition of this virtual link the capacity of the network is increased up to $(C(Q) + S_i^{(t)})$ units. Adding enough storage to node i at time t renders Q a non-minimum cut. If the new min-cut Q' contains a node for which the above condition also holds, then we add again storage so as to make Q' a non min-cut.

Specifically, Algorithm 1 , describes our proposed algorithm for **Storage Policy, (SP)** which is applied to graph G_T. **Step 5** is required in order to ensure that there is no excessive storage usage. The termination of the algorithm and its optimality are easily verifiable. The maximum number of added storage links is bounded by the number of nodes. Additionally, the amount of storage is confined by the capacity of links. Adding more storage will eventually result in a minimum cut consisting only of communication links. The **SP** algorithm guarantees the



(a) Initial Graph.



(b) The Storage Enhanced Time-Expanded Graph.

Fig. 3. A graph with time varying links and initial capacity $C(Q) = 34$. Node A is the source and node F the sink. Link capacities for $T = 7$ slots: $AB(18, 16, 18, 20, 16, 18)$, $BC(4, 10, 10, 4, 6, 4)$, $BD(6, 16, 16, 4, 4, 6)$, $CD(6, 10, 12, 2, 12, 8)$, $CE(6, 8, 2, 10, 12, 8)$, $DE(4, 6, 2, 12, 12, 8)$, $EF(10, 12, 14, 10, 20, 22)$. In step (1) we add the storage link $S_C^3 = 8$, in step (2) we add link $S_C^4 = 4$ and in step (3) the link $S_D^4 = 6$. The final capacity is $C(Q_f) = 42$.

maximum possible benefit from node storage utilization. The modified network graph has augmented average capacity over period T, equal to the *enhanced min-cut*, and reduced lower delay bounds for the transfer of certain amounts of data. Notice that this enhancement becomes an intrinsic characteristic of the network. However, **SP** does not provide a method for achieving this bound. In order to exploit the potential of storage it is required to consider it in conjunction with routing. In the next section we will provide a method for deriving the joint storage and routing policy for a single commodity data transfer.

V. Joint Storage and Routing Policy for Networks with Known Capacity Variation Patterns

Consider again the dynamic network $G = (V, E)$ and assume a slightly different setting where every node has a certain storage capacity which may vary with time. The storage management policy that achieves highest possible amount of transferred data for this network is given by the **SP** algorithm. In this section we find the exact routing and storage decisions that achieve this bound. We formulate the joint storage-routing (JSR) policy as an optimization problem and specifically as a maximum flow problem on the time-expanded graph $G_T = (V_T, E_T)$. Namely, the **JSR** problem is defined as follows:

Definition 1. *(Optimal Joint Storage and Routing - JSR Problem) Given a dynamic network $G = (V, E)$ with a single source and a single destination, and with nodes that have time varying storage capacity, find how much data should be stored in each node and how much data should be routed over each link, in every time slot, in order to maximize the amount of transferred data within a certain time period of T slots.*

Similar approaches have also been proposed in [8], and recently in [7]. However, in this work we use an algorithm which is amenable to distributed implementation and hence it is more suitable for the contemporary networking environment. First we add to G_T an artificial link (d, s) connecting the sink d with the source s. Every node i is connected with its backward instance at the previous time slot $(m, i) : m = i^{(t-1)}$ and its forward instance at the next time slot $(i, n) : n = i^{(t+1)}$. The capacity of these links represents the available storage at node i which is bounded by given limits. Additionally, for each node $i \in V_T$ we define the set of downstream (child) communication nodes $F_i = \{j : (i, j) \in E_T\} \setminus \{n\}$, and the set of parent communication nodes $B_i = \{j : (j, i) \in E_T\} \setminus \{m\}$. Therefore, there exist two distinct classes of links in the expanded graph: (i) the *communication links* that connect two different nodes i and j at a specific time slot t with capacity $\mathbf{C} = \{C_{ij}, (i, j) : i \in V_T, j \in F_i\}$, and (ii) the *storage links* that connect different time instances of the same node with capacity (i.e. storage) $\mathbf{S} = \{S_{in}\,(i, n) : i \in V_T, n = i^{(t+1)}\}$. Notice that the *communication links* include the notion of time and actually they represent the maximum amount of data that can be conveyed in the respective slot. Hence, both the communication and storage link capacity is measured in data packets.

A. Joint Storage and Routing Problem Formulation

Let us define the vector $\mathbf{x} = \{x_{ij}, (i, j) : i \in V_T, j \in F_i\}$ where x_{ij} denotes the amount of data that is sent over link (i, j) during the respective slot. Similarly, we define the vector of storage variables $\mathbf{y} = \{y_{in}, (i, j) : i \in V_T, n = i^{(t+1)}\}$ which denote the amount of data that can be stored at each node i in every time slot. The optimal storage-routing policy $(\mathbf{x}^*, \mathbf{y}^*)$ for the time period T is derived from the solution of the max flow problem defined over the corresponding time-expanded graph, **(Problem JSR)**:

$$\min(-x_{ds}) \tag{7}$$

subject to

$$\sum_{j\in F_i} x_{ij} + y_{in} = \sum_{j\in B_i} x_{ji} + y_{mi},\ i \in V_T \tag{8}$$

$$0 \le x_{ij} \le C_{ij},\ i \in V_T, j \in F_i \tag{9}$$

$$0 \le y_{in} \le S_{in},\ i \in V_T, n = i^{(t+1)} \tag{10}$$

Where it is $y_{in},\ x_{ij} \ge 0$. Equation (8) models the data transfer constraint for every node, in analogy with the flow conservation constraint, and $(-x_{ds})$ is the amount of data that is transferred from source to the sink in graph G_T. The solution of the **JSR** problem determines the routing, x^*_{ij}, and the storage decisions,

y^*_{in} that ensure the transfer of the maximum possible amount of data x^*_{ds} in the network G, during the entire period T. From a different perspective, this formulation can be used to find the incurred delay for the transfer of a certain amount of data D. Actually this is the minimum time T^* for which the solution of the respective **JSR** problem satisfies $x^*_{ds} \geq D$. We can find T^* by using a binary or another search method, similarly to section III.

In order to solve the **JSR** problem we can use a standard Primal-Dual method, [12]. First, we define the Lagrangian by relaxing the constraint (8) and introduce the vector of dual variables $\mathbf{p} = \{p_i : i \in V_T\}$:

$$L(\mathbf{x}, \mathbf{y}, \mathbf{p}) = -x_{ds} + \sum_{i \in V_T} \sum_{j \in F_i} (p_j - p_i) x_{ij} + \sum_{i \in V_T} (p_n - p_i) y_{in} \tag{11}$$

The dual problem is

$$\max_{p \in R} q(\mathbf{p}) \tag{12}$$

where

$$q(p) = \min_{[x_{ij} \leq C_{ij}, y_{in} \leq S_{in}]} L(\mathbf{x}, \mathbf{y}, \mathbf{p}) \tag{13}$$

The objective function of the primal problem is linear and therefore the dual function is non-differentiable. To overcome this difficulty we employ the ϵ-relaxation method that ensures convergence to the optimal solution if $\epsilon < \frac{1}{NT}$, in polynomial time $O(N^3T^3)$. We omit the detailed description of the algorithm and refer the reader to [12, Chap.5.3]. The algorithm is directly applicable to this problem since the variables x_{ij} and y_{in} can be treated jointly by substituting with f_{ij} and f_{in} respectively. Nevertheless, the main advantage of this method is that it is amenable to distributed implementation.

B. Distributed Algorithm for the JSR Problem

The ϵ-relaxation method can be executed in a distributed synchronous or even asynchronous fashion which is a very desirable property. We will focus on the later version here and cast the algorithm presented in [12], to fit our problem. In a distributed setting the variables are circulated among nodes and therefore they need to be time-stamped. Notice that these time stamps refer to the algorithm execution time and they should not be confused with data transmission time, which we denote below by t_G. The basic idea as in standard primal-dual methods is to exploit the separability property of the dual problem and group the variables per node. Specifically, every node $i \in V_T$ maintains the following variables:

- $p_i(t)$: dual variable of node i at time t
- $p_j(i,t)$: dual variable of node j, $j \in F_i \cup B_i \cup \{n\} \cup \{m\}$, communicated to node i at time t.
- $x_{ij}(i,t)$: estimate of node i for the data routed to node j, $j \in F_i$ at time t.
- $x_{ji}(i,t)$: estimate of node i for the data that it must admit from node $j \in B_i$ at time t.
- $y_{in}(i,t)$, $y_{mi}(i,t)$: estimates of node i for the data that must be stored at the time slots which correspond to the *storage links* (i,n), $n = i^{(t_G+1)}$, and (m,i), $m = i^{(t_G-1)}$ respectively, at time t.
- $g_i(t)$: estimate of node i for the data and storage surplus at time t, i.e. $g_i(t) = \sum_{j \in B_i} x_{ji}(i,t) - \sum_{j \in F_i} x_{ij}(i,t) + y_{mi}(i,t) + y_{in}(i,t)$

The nodes circulate messages with their variables in order to achieve coordination and collectively solve the **JSR** problem. Notice that adjacent nodes (neighbors) calculate the same variables and therefore it is required to reach consensus. 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. $x^*_{ij} = x^*_{ji}$. In detail, the distributed asynchronous algorithm for the solution of the **JSR** problem is presented below (Algorithm 2).

This algorithm solves the **JSR** problem even when there does not exist a central network controller with global knowledge.

Algorithm 2 (Joint Storage Routing - JSR)

Execution: The algorithm is executed continuously in a time sequence $t = (t_0, t_1, \ldots)$. Each specific time every node i executes one of the actions (Action 1 through 4) and checks the Termination Condition.

Termination Condition: The algorithm terminates when the data and storage surplus for all nodes becomes zero, i.e. $g_i(t) = 0\ \forall i \in V_T$.

(**Action 1** - *Vars Update*): Calculate data and storage surplus $g_i(t)$ and:

1. **If** $[g_i(t) > 0]$ **Then** update local variables $[p_i(t), p_j(i,t), x_{ij}(i,t), x_{ji}(i,t), y_{ij}(i,t), y_{ji}(i,t)]$ by executing Steps 2 - 5 of the ϵ-relaxation algorithm, [12, Chap.5.3].

(**Action 2** - *Notification*):

1. Send $Nfc_{ij} = [p_i(t), x_{ij}(i,t)]$, at every node $j \in F_i$
2. Send $Nfc_{ij} = [p_i(t), x_{ji}(i,t)]$, at every node $j \in B_i$

(**Action 3** -*Coordination*):

1. $\forall$ Nfc_{ji} received at $t' < t$, $j \in F_i$, update local variables:
   1.1 **If** $[p_j(i,t) \leq p_j(t')]$ **Then** set $p_j(i,t) = p_j(t')$
   1.2 **If** $[p_i(t) \leq p_j(t') + \alpha$ & $x_{ij}(i,t) > x_{ij}(j,t')]$ **Then** set $x_{ij}(i,t) = x_{ij}(j,t')$
2. $\forall$ Nfc_{ji} received at $t' < t$, $j \in B_i$, update local variables:
   2.1 **If** $[p_j(i,t) \leq p_j(t')]$ **Then** set $p_j(i,t) = p_j(t')$
   2.2 **If** $[p_i(t) \leq p_j(t') - \alpha$ & $x_{ji}(i,t) < x_{ji}(j,t')]$ **Then** set $x_{ji}(i,t) = x_{ji}(j,t')$

where $\alpha = -1$ if $(i,j) = (d,s)$ and $\alpha = 0$, otherwise.

(**Action 4** - *Storage Update*): Node i considers the information from its instances $m = i^{(t_G-1)}$ and $n = i^{(t_G+1)}$ for $t' < t$ and updates its storage decisions as follows:

1. **If** $[p_n(i,t) \leq p_n(t')]$ **Then** set $p_n(i,t) = p_n(t')$.
2. **If** $[p_i(t) \leq p_n(t')$ & $y_{in}(i,t) > y_{in}(n,t')]$ **Then** set $y_{in}(i,t) = y_{in}(n,t')$.
3. **If** $[p_m(i,t) \leq p_m(t')]$ **Then** set $p_m(i,t) = p_m(t')$.
4. **If** $[p_i(t) \leq p_m(t')$ & $y_{mi}(i,t) < y_{mi}(m,t')]$ **Then** set $y_{mi}(i,t) = y_{mi}(m,t')$.

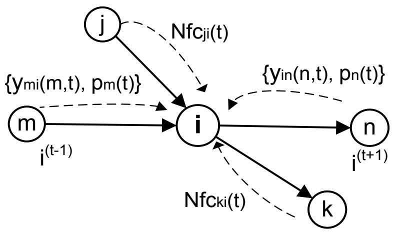


Fig. 4. Distributed execution of the ϵ-relaxation algorithm. Each node i receives coordination messages from its neighbors, $Nfc_{ji} = \{x_{ji}(j,t), p_j(t)\}$ and $Nfc_{ki} = \{x_{ki}(k,t), p_k(t)\}$; and updates its storage decisions by considering its forward $n = i^{(t_G+1)}$ and backward $m = i^{(t_G-1)}$ instances.

Instead it is only required every node to be aware of its own link capacity variation patterns. This scenario is very important since it models a large set of networking examples. However, future knowledge about the network state is still a prerequisite. In the following section we explain why it is important to know the patterns of the link capacities and we discuss some cases where it is possible to derive suboptimal solutions even when there is lack of information.

VI. Instances of Limited Knowledge About Link State

When there is no information about the future state of the network, a subset of the constraint set of the **JSR** problem is not known nor can it be determined through message passing. In this case, the joint storage management and routing policy becomes an online problem where nodes must decide using only the currently available information. In general, these problems are solved through dynamic programming techniques and optimal solutions are difficult to characterize and derive, [13]. Letting storage aside, distributed dynamic routing has been studied both for wireless [14], and wire-line networks, [15]. The underlying idea is the same in both algorithms. Namely, each node independently takes routing decisions so as to balance network load by forwarding its data packets to its neighbors with the less backlog, i.e. the smaller queues. If all of its neighbors are congested, the node refrains from sending its packets and the detected congestion is gradually signaled back to the source which temporary pauses data transmission. This scheme constitutes a proactive end-to-end flow control mechanism which may degrade the data delivery capability of the network. Therefore, these algorithms are not delay-aware and there is much ongoing research aiming at their improvement [**?**], [16].

In this context, we can consider storage utilization as a method for modifying the above congestion detection mechanism in order to reduce data transfer delay. To make this clear, consider a congested node which is aware that its outgoing link capacity will significantly increase in the near future. In this case, this node would be able to decide not to signal back to its parent nodes the congestion so as to keep receiving data from them. The excess data would be stored in the storage area of the node and returned to the queues when the backlog is reduced as described in Figure 5. This way, the node would prevent time consuming temporary pausing of flow and hence eventually enable faster data delivery to the sink. In other words, node storage could be used to transform the end-to-end flow control to a hop-by-hop operation. The challenge in this setting is to detect the conditions that render storage utilization beneficial for the network performance. Without information about the future state of the network links the described congestion-biasing technique may deteriorate the network performance. Clearly, when a node decides to store some data it must know that this cannot be routed at that time from alternative shorter non-congested paths.

The fundamental difficulties encountered in the online version of the **JSR** problem motivate the exploration of specific network operation scenarios where suboptimal solutions are possible. For example, consider a network where the administrator (or the nodes) can predict the future values of link capacities and node available storage with bounded error. In this case, we can solve a variation of the **JSR** problem, name it **EJSR**, which stems from the original problem if we substitute the actual with the estimated parameters at the constraint set. Namely, instead of C_{ij}, and S_{ij}, we can use the worst-case predictions $\hat{C_{ij}} = (C_{ij} - e^c_{ij})$ and $\hat{S_{ij}} = (S_{ij} - e^s_{ij})$. The quantities e^c_{ij} and e^s_{ij} represent the maximum prediction error in the link capacities and node storage respectively. Obviously, the optimal solution of the **EJSR** problem is feasible for the respective **JSR** while its optimality depends on the accuracy of predictions.

Another interesting scenario is when the nodes are aware of their links and storage capacities only for the near future. In this case we can find the short-term storage policy using a similar algorithm with the one presented in section IV. Namely, we can use distributed algorithms such as those in [17], for finding the current min-cut $Q^{(t)}$ and also the min-cut of the next few slots, $Q^{(t+1)}, \ldots, Q^{(t+M)}$. This is accomplished through message passing among nodes with information about their current and future state. Now assume that a certain node i belongs both to the set $W^{(t)}$ of the $Q^{(t)}$ and to the set $N \backslash W^{(t+1)}$ of the $Q^{(t+1)}$ cut. In this case, node i can infer that it must use its storage capacity and admit the excess data that is routed to it. This policy enables the network performance improvement through the use of storage, although this is not the maximum possible, as was the case with the **SP** algorithm.

VII. Numerical Results

In order to verify the validity of our approach we simulated the operation of 3 networks with storage-capable nodes. Two of them are linear networks with 3 and 5 nodes respectively, while the third one is the graph of Figure 3(a). The objective was to demonstrate the impact of intermediate storage to the performance of the network for various capacity evolution scenarios. These scenarios are modeled through the link capacities dissimilarity index L. Recall that higher values of L imply more diverse capacity patterns. The performance metric is either the amount of data that can be transmitted within a given time

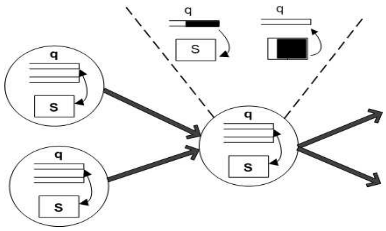


Fig. 5. Storage and queue management for enabling hop-by-hop flow control. When queue backlog increases, excess data is moved to storage area to avoid congestion signaling. Then, when backlog is reduced, data is restored in respective queues.

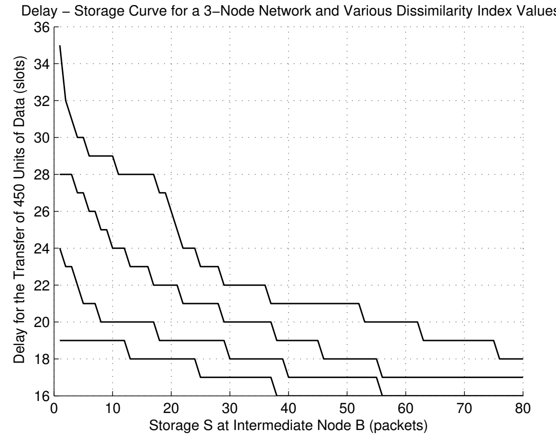


Fig. 6. Delay - Storage curves for a 3-node linear network with intermediate storage, for the transfer of $D = 450$ units of data and various values of L. From the lower to the upper curve, it is $L = 867$, $L = 884$, $L = 909$, and $L = 934$.

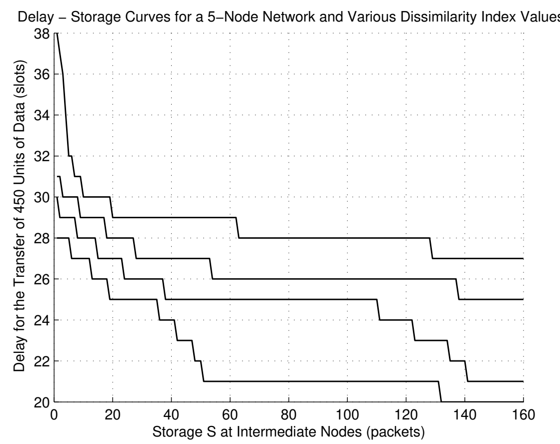


Fig. 7. Delay - Storage curves for a 5-node linear network with intermediate storage, for the transfer of $D = 450$ units of data and various values of L. From the lower to the upper curve it is $L = 1120$, $L = 1204$, $L = 1306$, and $L = 1421$. Storage is equally distributed to nodes.

period or equivalently, the incurred delay for the transfer of a certain amount of data from source to sink. The later is visualized through *Delay - Storage curves* where we depict the delay versus the aggregate storage of nodes for the transfer of various amounts of data. Clearly, the benefit of using storage varies for different networks and different values of L from zero to substantial improvement on performance.

We begin with the 3-nodes linear network of Figure 1 where node B has storage capability of S_B units. We consider a time slotted operation for $T = 40$ slots and assume that link capacities C_{AB} and C_{BC} vary with time. The network operation is described by equations (1) - (5). In Figure 6 we depict the delay for the transfer of $D = 450$ units of data from node A to node C for different values of L. We see that as storage S_B increases, the incurred delay reduces down to a minimum value. Further usage of storage does not improve the performance of the network. Similarly, in Figure 7 we depict the delay for the transfer of $D = 450$ units of data in a linear network of 5 nodes. We see again that the benefit from storage use to the network performance is almost proportional to the dissimilarity index L. Namely, notice that the distance of the maximum to the minimum delay value for every plot increases with L.

In Figure 8 we fix the value of L and plot the delay for different amounts of transferred data for the 3-nodes network. Notice that the lower bounds of incurred delay are different for different amounts of data. Finally, in Figure 9 we depict the maximum amount of transferred data D, for a time period of $T = 20$ time slots from source to sink in the network of Figure 3(a). We see that this amount increases as a function of aggregate available storage at intermediate nodes B, C, D, and E up to a maximum value. Further increase in storage capacity does not improve the performance of the network. This upper limit depends both on the network graph and on the dissimilarity index L of the links for the time period T.

VIII. Conclusion

In this work we showed that storage under certain conditions can improve the network performance of dynamic networks. This improvement is realized either as increase of the amount of data that can be transported from the source to the destination within a finite time horizon or, equivalently, as reduction of the incurred delay for delivery of certain amounts of data. The optimal storage management policy is the one that guarantees maximum benefit from storage use and can be derived for every network using the presented **SP** algorithm. In order to realize this benefit, storage must be considered in conjunction with routing. The joint storage management - routing policy can be derived through the solution of a max flow problem defined over a time-expanded graph. More importantly, this problem can be solved in a distributed fashion.

The proposed methodology has many interesting applications. For example, it can be used to analyze and improve inter-data center communication where the cost of bulk data transfer is extremely high and time varying, [2]. Intra-data center networking is another area that we believe it can benefit from this analysis. Designing the architecture of a data center

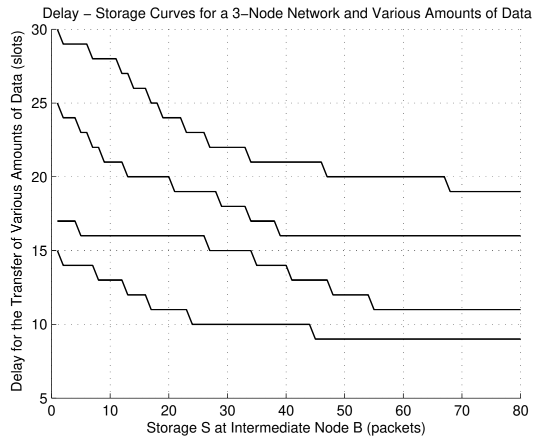


Fig. 8. Delay - Storage curves for different amounts of transferred data in a 3-node linear network with fixed intermediate storage S_B, and dissimilarity index $L = 909$. From the lower to the upper curve it is $D = 200$, $D = 300$, $D = 400$, and $D = 500$.

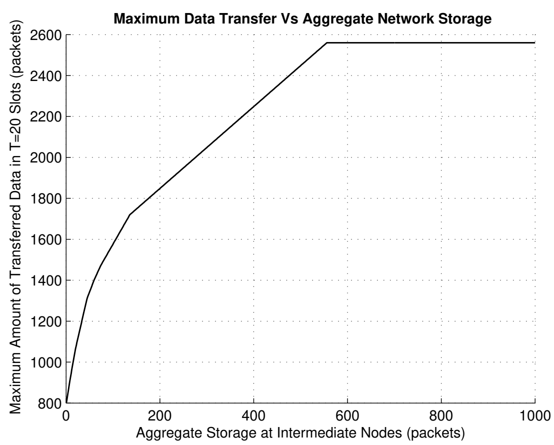


Fig. 9. Maximum amount of data transferred in $T = 20$ time slots for the network of Figure 3(a), as a function of the total available storage at the intermediate nodes.

is a very challenging task and must take into account both the performance and the cost of the equipment, [18]. Hence, it is very important to use efficiently both the available storage and links capacity resources. Moreover, node storage can be used to enhance the operation of peer-to-peer systems where the performance bottleneck is the uplink capacity, [19]. Finally, in many cases, the network operator can employ the proposed method to appropriately use nodes storage and reduce the required link capacity without degrading network performance.

This is a first attempt to understand the impact of storage capacity in networks with full knowledge over the link capacity state and its evolution. The next big step we will pursue will be towards understanding the online version of the problem. In this case, link capacities obey a known discrete or continuous probability distribution, but the controller knows only the current value of link capacities just before taking a decision. Again, it is imperative to consider algorithms which are amenable to distributed implementation.

IX. Acknowledgment

The first two authors acknowledge support of the European Commission through the STREP project PURSUIT (FP7-ICT-2010-257217).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above the Clouds: A Berkeley View of Cloud Computing,” UC Berkeley Technical Report EECS-2009-28, 2009.

[2] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram, “Delay Tolerant Bulk Data Transfers on the Internet,” in *ACM SIGMETRICS*, 2009.

[3] L. R. Ford and D. R. Fulkerson, *Flows in Networks*. Princeton University Press, 1962.

[4] S. Jain, K. Fall, and R. Patra, “Routing in a Delay Tolerant Network,” in *ACM SIGCOMM*, 2004.

[5] E. P. C. Jones, L. Li, J. K. Schmidtke, and P. Ward, “Practical Routing in Delay-Tolerant Networks,” *IEEE Trans. Mobile Computing*, vol. 6, pp. 943–959, 2007.

[6] 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 *ACM HotNets*, 2008.

[7] P. Chhabra, V. Erramilli, N. Laoutaris, R. Sundaram, and P. Rodriguez, “Algorithms for Constrained Bulk-transfer of Delay-Tolerant Data,” in *IEEE ICC*, 2010.

[8] R. G. Ogier, “Minimum-Delay Routing in Continuous-Time Dynamic Networks with Piecewise-Constant Capacities,” *Networks*, vol. 18, no. 4, pp. 303–318, 1988.

[9] A. Orda, R. Rom, and M. Sidi, “Minimum Delay Routing in Stochastic Networks,” *IEEE/ACM Trans. Netw.*, vol. 1, no. 2, pp. 187–198, 1993.

[10] B. Kotnyek, “An Annotated Overview of Dynamic Network Flows,” INRIA, Technical Report No 4936, 2003.

[11] L. Fleischer and M. Skutella, “Minimum Cost Flows Over Time without Intermediate Storage,” in *ACM-SIAM SODA*, 2003.

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

[13] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley, 2005.

[14] L. Tassiulas and A. Ephremides, “Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks,” *IEEE Trans. on Automatic Control*, vol. 37, no. 12, pp. 1936–1948, 1992.

[15] B. Awerbuch and T. Leighton, “Improved Approximation Algorithms for the Multi-Commodity Flow Problem and Local Competitive Routing in Dynamic Networks,” in *ACM STOC*, 1994.

[16] L. Ying, S. Shakkottai, and A. Reddy, “On Combining Shortest Path and Back Pressure Routing Over Multihop Wireless Networks,” in *IEEE INFOCOM*, 2010.

[17] L. Bui, R. Srikant, and A. Stolyar, “Novel Architectures and Algorithms for Delay Reduction in Back Pressure Scheduling and Routing,” in *IEEE INFOCOM*, 2009.

[18] T. L. Pham, I. Lavallee, M. Bui, and S. H. Do, “A Distributed Algorithm for the Maximum Flow Problem,” in *IEEE ISPDC*, 2005.

[19] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Lu, “Dcell: A Scalable and Fault-Tolerant Network Structure for Data Centers,” in *ACM SIGCOMM*, 2008.

[20] N. Laoutaris, P. Rodriguez, and L. Massoulie, “ECHOS: Edge Capacity Hosting Overlays of Nano Data Centers,” *ACM Comp. Comm. Rev.*, vol. 38, no. 1, pp. 51–54, 2008.