Scalable Algorithms for Communication Networks

by

Li-yen Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering: Systems)
in The University of Michigan
2010

To my parents and teachers.

ACKNOWLEDGEMENTS

I am grateful for having the ideal advisor, Professor Petar Momcilovic. Working with Professor Momcilovic has brought me the greatest joy of learning. His industrious and rigorous attitudes made him the paragon of a researcher throughout my graduate study. Without his vision and guidance, this dissertation simply would not have been possible.

I would like to express my gratitude towards my dissertation committee. I thank Professor Demosthenis Tenekitzis for his comprehensive and valuable comments on my research. He has also been an inspiring teacher in my mathematical coursework. I am grateful to Professor Mingyan Liu for her attentive reviews on my thesis draft and her excellent teaching in my very first course on communication in Michigan. I also thank Professor Mark Van Oyen whose course intrigued my interest in queueing theory.

My gratitude extends to all my teachers in the Electrical Engineering: Systems, Mathematics, Statistics, Computer Science and Industrial & Operations Engineering Departments. In particular, I thank Professor Serap Savari for her warm and helpful advices. I am grateful for having the company of my classmates and colleagues. I thank my friends, for being consistently supportive and encouraging. They have made my graduate life cheerful and enjoyable.

This dissertation would not be complete without the mention of my respectful parents. Their dedication to my education is the primary reason for my love towards science and engineering. I sincerely thank them for their confidence in my capability of doing anything with determination.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

ABSTRACT

Network scalability has emerged as the essential problem in designing architectures and protocols for large-scale communication systems. Minor efficiencies, that can be tolerated in small networks, can accumulate and become a dominant factor determining the performance of large networks. In this thesis, we consider three problems that are related to scalability. First, we examine the size of routing tables as the number of nodes in the network increases. It is shown that the widely used shortest-path and straight-line routing algorithm can be implemented only when nodes' memory increases with the network size. On the other hand, it is established that there exist information-efficient algorithms, e.g., column-first routing protocol, that route packets correctly even if each node in the network is capable of storing information on a fixed number of destinations only. In the second part, we present a novel computational model utilizing time encoding, that enables a distributed scheduling mechanism. The scheduling algorithm we propose achieves performance comparable to centralized algorithms under uniform traffic. Exploiting a connection between switch scheduling and interval packing, we argue that the distributed nature of the algorithm limits the maximum relative load to $1 - e^{-2}$ under the worst-case scenario. The stability of the algorithm can be improved by enabling reversibility in distributed decision making. Finally, we discuss the bandwidth sharing problem for multi-hop networks. A buffer management policy that utilizes only simple packet attributes delivers a constant fraction of the maximum possible throughput. Moreover, the policy is robust to heavy traffic loads in the sense that the throughput does not degrade due to congestion.

CHAPTER 1

Introduction

Network scalability, a topic motivated by the growth of networking infrastructures, has emerged as a pivotal problem in designing architectures and protocols for large-scale multi-hop wireless networks. Indeed, the impact of large scales on the system performance can be acute: minor efficiencies, that can be tolerated in small networks, can accumulate and become a dominant factor determining the performance of large networks. As a result, the amount of local resources (e.g. buffer space, computational power, transmission bandwidth) provides a limitation on the network size that can be supported. Given that in the future, the growth of network size is only expected to continue, it is of interest to develop algorithms that support the operation of large networks.

Most of the scalability problems are impractical to be addressed experimentally due to the considerable cost of building large-scale prototypes. Even simulating such systems is often very difficult because of the computational limitations. Hence, our focus on mathematical analysis. In contrast to previous studies that overlook the limitations of individual nodes, our interest is in the impact of finiteness of resources at individual nodes on the overall network performance. To this end, in the rest of this dissertation, we consider three aspects of operation in large-scale networks: routing, scheduling and bandwidth sharing. The obtained analytical insight, e.g. scaling laws and optimal operations, of these problems is then utilized to design scalable algorithms, i.e., algorithms that require only negligible resources but achieve performance comparable to the performance delivered by centralized algorithms. The algorithmic aspect is particularly important given that some of widely considered algorithms require excessive resources at individual nodes and, hence, are not scalable. In these three aspects of the network operation, we improve the scalability of the algorithms from angles that were not considered in the literature, which in turn demonstrates the multidimensionality of the network scalability. Our study also address the tradeoff between the distributiveness and the performance of the algorithms, which has rarely been considered in the design of small systems. These results serve as the first-step analysis toward the other general network scalability problems, and indicate that completely new protocols are in need of establishment to support the operation of large-scales networks. The ultimate goal is to establish the existence and construction of algorithms that support efficient operation of large-scale networks of nodes with limited resources.

1.1 Routing Problem

In this section, we summarize the main results on the routing problem. We consider a network model with N nodes located in a unit square and grouped into *cells* based on their locations. It is appropriate to think of cells as nodes in a higher-level logical network. Our focus on cell-based algorithms is due to the following property: under the relaxed protocol model (a common model for interference) and a carefully chosen cell dimension, the number of cells that interfere with any given cell is bounded by a constant independent of the network size N [24]. In other words, the cell tessellation provides us a normalization of the network – a constant fraction of cells can transmit/receive packets simultaneously. For these cell-based algorithms, the routing table for each cell typically contains multiple *records*, that describe how packets to specific destination nodes need to be handled by this cell. Note that if one requires a node to hold a single packet, then the node memory increases at least logarithmically in the network size, since $\lceil \log_2 N \rceil$ bits are needed to uniquely identify a destination. Given that all nodes have identical capabilities (hardware), it is reasonable to make the assumption that the routing records are spread evenly among the nodes in a cell; hence it is sufficient to consider the amount of routing information stored in each cell. Each record contains a destination address, possibly along with adjacent cell identifiers, depending on the employed routing policy. In particular, nodes can maintain (possibly in compressed form) lists of records, each containing the address of a destination node, an input cell label and an output cell label. With this type of records, a cell forwards only packets with the destination address and input cell label that match a record in the list. Observe that in this case a cell can be treated as a switch/router with neighboring cells being input/output ports. Including the extra information (cell labels) in the records helps with compression of routing tables and is subject to only a small overhead.

We first consider a theoretic lower bound on the size of routing tables. In particular, for given forwarding and input cells, it is shown that the bound is proportional to the minimum number of routes utilizing the same output cell. The intuition behind this result is that if the packets forwarded by a cell are directed to all of the neighboring cells equally likely, the routing table cannot be compressed efficiently due to the high entropy of the content. On the other hand, if a large body of the packets forwarded by a cell is directed to a single neighbor, then the routing table can be effectively compressed. In particular, the majority of the packets are forwarded through this direction by default; only the records for those packets that are to be handled differently need to be stored in the table. The by-default forwarding allows us to eliminate the set of destination addresses (and output labels that are of negligible length) that are forwarded in the dominant direction, and form the routing table for destination addresses that are forwarded in the other directions only. We illustrate this result by applying the well-known longest-prefix compression algorithm [67] to two cases of forwarding directions: (i) uniform forwarding, and, (ii) biased forwarding. We demonstrate that the more biased the distribution of forwarding direction is, the better the performance of the longest-prefix compression. Therefore, it is desirable to design a routing protocol that routes packets on pre-constructed “highway system” – most of the packets (packets along the “highway”) travel on same routes and are handled in the same way by intermediate cells.

Next, we propose the main concept of this work: information-efficiency. A routing protocol in a flat-label network is said to be *information-efficient* if the amount of information at individual nodes required to route packets does not increase with the network size. We first examine the straight-line routing algorithms, considered in [33, 24, 25], (packets are forwarded along the cell intersected by the straight line connecting the source and destination cells). Such a routing scheme is shown to be *not* information-efficient. That is, this protocol can be implemented only when nodes’ memory increases with the network size. A closely-related algorithm is the shortest-path algorithm, which selects a route such that it contains the minimum number of cells (hops) at random. This algorithm is also information-inefficient. These results suggest that these two straightforward routing algorithms, which are aligned with many current real-world protocols, are of poor scalability and are not suitable for large-scale communication networks.

On the other hand, we argue that there exist information-efficient algorithms that route packets correctly even if each node in the network is capable of storing information on a fixed number of destinations only. To demonstrate the existence of such algorithms, we prove the information-efficiency for a particular routing scheme, i.e., column-first routing. We note that, in consistent with the discussion about compression of routing tables, column-first routing yields a single dominant forwarding direction in each cell. The notions of columns and rows (within the nature of square cells) serve as pre-existent routes; packets travel along these routes for most of the duration from the source to the destination. The information efficiency of column-first algorithm indicates that it is desirable to construct a "highway" system for routing, in order to maintain a constant size routing table at individual nodes. Our focus on column-first policy is due to the simplicity – it is not the only information-efficient routing algorithm.

In the last part of this work, we relax the assumption on the cell shape by proposing a heuristic distributed algorithm for constructing virtual rows and columns with Voroni cells (see [63] for a discussion of Voroni tessellation). Column-first-like routing can be operated given these pre-existent "highways". Our procedure starts with a single random cell identifying the furthest cell in the network by flooding a distance vector (counters of numbers of hops). Analogously, this identified cell can find a furthest cell, and the shortest path in between serve as an initial virtual row. This virtual row in turn flood a distance vector and establish other virtual rows in the network. Virtual columns can be established in a similar fashion. After the one-time construction, each cell in the network belongs to exactly one virtual row and one virtual column, and column-first-like algorithms can operate correctly utilizing these virtual rows/columns. The described procedure requires only negligible overhead, and provides only one way of constructing the virtual routes; it is by no means exclusive.

1.2 Scheduling Problem

This section outlines our research contribution in the scheduling problem. We design a fully distributed scheduling scheme with implicit information exchange by introducing a new model of computation based on *time encoding*. In our model, variables are represented with time intervals of possibly random lengths rather than deterministic bits. A time interval (and hence its length) can be defined by two impulses. Two operations are in particular enabled with simple implementation by the time-encoding model: minimum and inverse sum. The minimum of two intervals is given by the following procedure: define the new starting impulse as the beginning of the two aligned time intervals, and the new ending impulse as the earlier of the two ending impulses. The inverse sum can be implemented by applying the minimum operator to exponential random variables with means being the inverse of the input variable; this operator is only approximate, but can be implemented in a distributed fashion.

Enabled by the time-encoding model is a set of randomized scheduling algorithm that allows for distributed implementation. We focus on the scheduling of a crossbar switch due to both the simplicity and generality of the architecture. In the algorithm we propose, each time slot is divided into two phases: scheduling and cell transfer. In a scheduling phase, each input-output pair is represented by an exponentially distributed random variables with mean being the inverse of the corresponding weight; the weight of a pair can be computed through a selected weight function using only local attribute of this pair. A input-output pair with the minimum variable is selected in each iteration until no more pairs can be selected; then, the switch facilitates transmission to the selected pairs in the remaining of this time slot (cell transfer phase). In order to implement this algorithm distributively, suppose that the source of randomness is available at each input port, i.e., a unit mean exponential random time interval (timer) for each time slot. Adjusting the intensities of these time intervals is equivalent to setting timers based on the weight at input-output pairs. Upon the expiration of a timer, the pair is scheduled unless it conflicts with another scheduled pair. This process continues until all timers with finite length expire. The exponential timers can be implemented conveniently with our time-encoding computing model. Since only limited inter-port information exchange is performed, the implementation of this scheduling algorithm is distributed, and hence, scalable.

The described scheduling algorithm is shown to be rate stable for packet switches of size 2×2 under any admissible traffic pattern. Moreover, we show that it is rate stable for packet switches of an arbitrary size under uniform traffic patterns. The proof of these results follows the fluid model, i.e., a deterministic, continuous system of equations that parallels the evolution of the buffer occupancy process in packet switches. Our numerical comparison shows that this algorithm performs competitively (in terms of buffer occupancies/delay) under uniform traffic for a wide range of relative load, when compared to *centralized* switch scheduling algorithms, e.g., Maximum-Weight-Matching(MWM), Longest-Queue-First, and APSARA [29](a heuristic algorithm based on the MWM with a smaller complexity). For diagonal traffic, the algorithm fails to stabilize the switch for high values of relative load. We argue that the distributed nature of the algorithm limits the maximum relative load to $1 - e^{-2}$ under diagonal traffic by exploiting a connection between switch scheduling and interval packing [65]. An upper bound of the critical relative load for diagonal traffic pattern with arbitrary parameter can be evaluated numerically through an iterative procedure.

The stability of the algorithm can be improved by enabling reversibility in distributed decision making. The above algorithm makes irreversible scheduling decisions; once a pair is scheduled, the decision remains valid until the next scheduling phase, regardless of consequent scheduling decisions. In a reversible scheduling algorithm, all scheduled pairs are subject to disconnections based on the arrivals of a fixed rate Poisson processes until the end of the scheduling phase. Similar to the basic algorithm, each reconfiguration of the switching fabric, which either schedules or un-schedules a input-output pair, involves only this single pair – no inter-port communication is required. Therefore, it can be realized by a similar scheme as the one described above, with the additional ability to un-schedule. We make the observation that the state of the switching fabric is described by a continuous-time Markov chain. By deriving the product-form stationary distribution of this chain, we conclude that the probability of a matching being scheduled increases exponentially with respect to the number of packets in the matching. The expansion of stability region comes at the cost of a higher computational complexity.

1.3 Bandwidth Sharing Problem

In this section, we summarize our results in the bandwidth sharing problem. We do not study the problem under the fluid assumption of the network flows, i.e., treating the traffic as continuous medium; instead, the system is examined at packet-level granularity. Our approach to achieve the desired bandwidth sharing is based on employing buffer management policies. A buffer management mechanism consists of two parts: the service discipline and the dropping policy. In particular, the network throughput under three buffer management policies is considered and compared, namely, the first-come-first-serve (FCFS) discipline with tail dropping, the oldest-packet-first (OPF) policy and the newest-packet-first (NPF) policy. A network node operating under the FCFS discipline serves packet in its order of arrival to the node; it discards the newly arrival packets when the buffer is fully occupied under tail dropping. Under the OPF policy, the packets arriving to a node is served/enqueued with priority given by their age; the larger the number of hops it has traveled in the network, the higher the priority. In contrast, the NPF serves/buffers packets with a smaller number of hops traveled with higher priority.

We consider two cases of service time distribution for the network nodes: deterministic and exponentially distributed service times. In both cases, the throughput under FCFS discipline with tail dropping is maximized for a specific value of input rate, which is dependent of the network size. The throughput degrades exponentially with the increase of input rate beyond that value. In other words, in order to achieve the highest possible throughput, additional congestion control mechanism at end users is required to maintained the input rate at a certain level; otherwise the system throughput is subject to the decay due to congestion. On the other hand, the throughput under the OPF policy do not exhibit such behavior. As the input rate increases, the throughput approaches a constant fraction of the maximum throughput, suggesting that the OPF policy is robust to high traffic load, i.e., free of congestion. The throughput under the NPF policy behaves similar to the throughput under the FCFS discipline with tail dropping.

1.4 Organization of Dissertation

The dissertation is organized as follows. In Chapter 2 we analyze the scalability of routing problem. The scalability of scheduling problem the bandwidth sharing problem is studied in Chapter 3 and Chapter 4, respectively. Chapter 5 contains concluding remarks and reflections of our research. Subsidiary proofs and computation procedures omitted in the main text are located in the Appendices.

CHAPTER 2

Scalable Routing

2.1 Introduction

Mathematical study of large-scale wireless networks has been initiated in [33] where the authors established an upper bound on the network throughput as a function of the network size (number of nodes). The authors also constructed a protocol that achieves a slightly smaller throughput than the limiting one. Consequent studies focused on the throughput-delay trade-off [24, 25], achievability of the limiting throughput [27], and various extensions of the original model and their analysis [38, 49, 46]. The throughput capacity of wireless networks relates to that of lattice networks, e.g. see [6, 7] and the references therein. When the network nodes exhibit high mobility, i.e. discontinuous positions, the capacity is further improved [32].

It is recognized that design of scalable routing algorithms for wireless ad hoc networks is a challenging problem [68]. The difficulty is due to the fact that node identifiers (addresses) of adjacent nodes are not similar (flat-labeling), i.e., the address aggregation employed in the Internet is not directly applicable. Geographic routing [43, 50, 12] achieves aggregation by assigning additional identifiers to nodes based on their geographic location. However, we do not exploit this approach here since in that case an additional mechanism is needed to provide translation between the original and location-based identifiers; resource requirements of such a DNS-like mechanism can be significant. In this chapter, we focus on one particular aspect of routing scalability: the size of routing tables, i.e., the amount of information stored at network nodes required for routing packets between sources and respective destinations. In contrast with our interest in the required memory space of the control plane, the majority of existing studies that concerns limited memory space is dedicated to that of the data plane, i.e., buffer space used to store enqueued packets. Indeed, in [7] the authors obtained an approximation of the throughput in a large finite-buffer lattice network with deterministic transmission times. A wireless network with exponential transmission times and finite-buffer nodes was considered in [37]. Buffer requirements in a wireless network with mobile nodes were discussed in [34].

The size of routing tables is directly dependent on the adopted routing algorithm. To this end, we term a routing protocol *information-efficient* if it can operate correctly when each node stores information on a fixed number of destinations regardless of the network size. That is, routing tables at network nodes do not increase with the network size. We show that the straight-line routing algorithm, considered in [33, 24, 25], and the closely related shortest-path algorithm are not information-efficient. However, we demonstrate that the column-first routing scheme is information-efficient. Such an algorithm induces routing tables that can be efficiently compressed.

The remaining of this chapter is organized as follows. In the next section we describe a wireless network model and state preliminary results. Routing tables are discussed in Section 2.3. The following section contains scalability results on the straight-line and shortest-path routing algorithms. In Section 2.5 we demonstrate that there exists an information-efficient routing scheme for large wireless networks. A distributed algorithm is outlined in Section 2.6. Concluding remarks and technical proofs can be found in Section 2.7 and Appendix A, respectively.

2.2 Model and Preliminaries

Consider a random topology network model with N nodes placed uniformly at random in a unit-area square. Each node is identified by a unique address (label). Node locations and addresses are independent, i.e., the Internet model of address aggregation is not applicable; such an assumption holds often for ad hoc networks. A node sends a packet to some other node in the network by specifying the destination node's address to the network layer, i.e., a packet contains its destination address.

We consider the relaxed protocol model of interference [24, 25]: a transmission from node i to node j is successful if for some $\delta > 0$ and all nodes $k \neq i$ transmitting simultaneously with i

$$\Delta(k, j) \geq (1 + \delta)\Delta(i, j), \tag{2.1}$$

where $\Delta(i, j)$ is the Euclidian distance between nodes i and j. It is known [33] that under reasonable assumptions this model is equivalent to the popular physical model based on the signal-to-interference ratio.

In the following discussions, we focus on a cell-based algorithm for packet transmission. Namely, the unit square is divided into square cells of area a_N, where the subscript indicates the network size. The shape of cells is square just for the convenience of analysis; other ways to partition the unit square into cells can be considered, e.g., Voronoi tessellations [33] (see also Section 2.6). For simplicity, we assume that the quantity $1/\sqrt{a_N}$ is an integer so that all cells ($1/a_N$ in total) are of the same size and shape. Without loss of generality, let $\{a_N\}$ be a non-increasing sequence in N. A packet is delivered from its source to the destination by forwarding between cells, i.e., between the nodes located in those cells. A packet can be forwarded from a cell to only one of the four neighboring cells (north, east, south, west); no multi-cell hops are permitted. Under the relaxed protocol model the number of cells that interfere with any given cell is bounded by a constant independent of the network size N [24, Lemma 2]. In other words, a constant fraction of cells can transmit/receive packets simultaneously.

In order for the cell-based algorithm to deliver packets to their destinations, each cell that forwards packets should contain at least one node. Requiring that every cell in the unit square contains at least one node can satisfy this condition. To this end, let $k_N(i)$, $1 \le i \le 1/a_N$, be the number of nodes in the ith cell (the labeling of the cells is arbitrary) when the network consists of N nodes. Given the uniform distribution of node locations, there exists a minimum cell size a_N that probabilistically ensures $k_N(i) > 0$ for all $1 \le i \le 1/a_N$. A stronger version of the following lemma was stated in [24]; it indicates that is sufficient to have $a_N = \Omega(\ln N/N)$[1] as $N \to \infty$.

Lemma 2.1. If $a_N > a \ln N/N$ for some $a > 1$ then, as $N \to \infty$,

$$\mathbb{P}\left[\min_{1\le i\le 1/a_N} k_N(i) = 0\right] \to 0.$$

The next lemma provides an additional characterization of the minimum and maximum number of nodes in a single cell. Note that the expected number of nodes in a cell is given by $\mathbb{E}k_N(i) = a_N N$.

Lemma 2.2. Suppose that $a_N > a \ln N/N$ for some $a > 1$. There exist $0 < \gamma_1 < \gamma_2 < \infty$ such that, as $N \to \infty$,

$$\mathbb{P}\left[\min_{1\le i\le 1/a_N} k_N(i) \ge \gamma_1 a_N N\right] \to 1 \tag{2.2}$$

and

$$\mathbb{P}\left[\max_{1\le i\le 1/a_N} k_N(i) \le \gamma_2 a_N N\right] \to 1. \tag{2.3}$$

Proof. See Appendix A.1. □

Next, we describe the configuration of traffic patterns. Each node chooses a *point* on the unit square uniformly at random as its destination. All destinations are inde-

[1]We use the standard asymptotic notation, see e.g. [15, Section I.3].

pendent. Since at a chosen point there will be no node with probability 1, packets from the corresponding source will be delivered to one of the nodes that share the cell with the destination point. A similar assumption was made in [33] – it ensures the independence of traffic routes. A routing algorithm specifies a set of cells that forward packets between the sources and respective destinations. It this chapter we compare three different routing policies: straight-line, shortest-path and column-first. These policies operate as follows:

- *Straight-line:* The source node and destination point are connected by the straight line. Packets are forwarded along the cells intersected by the line. This policy was employed in [33, 24, 25] because it facilitates mathematical analyses. However, it cannot be implemented easily in practice without sophisticated node hardware and/or central authority.

- *Shortest-path:* The route is selected in such a way that it contains the minimum possible number of cells (hops). When a cell can achieve the same number of hops via multiple neighbors, the cell chooses one of these neighbors uniformly at random. The main advantage of this policy is that it can be implemented via the distributed asynchronous Bellman-Ford algorithm [10, Section 5.2].

- *Column-first:* Packets are delivered from the source cell to destination cell in two phases [48, Section 1.7]. First, they are forwarded along the column that contains the source cell until they reach the row that contains the destination cell. In the second phase packets are forwarded along the row to their destination.

We conclude the description of the routing policies with an observation that for the considered model all (unit square and square cells) three algorithms produce routes of the same hop (cell) length for any source-destination pair.

One more aspect of the network model needs to be specified: the amount of time required to correctly transmit a packet between two neighboring cells. Packet transmission times can be modeled as both deterministic [33, 24, 25, 7] and stochastic [37] quantities. In particular, in [37] it was assumed that the transmission times are exponentially distributed. The expected transmission time is a function of the packet size in bits. Since we assume that destination addresses are included in each packet's metadata field, the packet size has to increase at least logarithmically in the network size N, i.e., $\log_2 N$ bits are needed on average to uniquely identify all nodes in the network.

For simplicity we assume that each node has an infinite amount of buffer space to temporarily store packets in the course of the forwarding process. Network throughput ϑ_N is feasible if *all* N source-destination pairs can communicate at long-term rate ϑ_N. The cell-based algorithm is optimal in the sense that it achieves asymptotically the optimal trade-off between the throughput and delay [24, 25]. Studies [24, 25] assumed deterministic service times with a time-slotted transmission system and employed the straight-line routing scheme. It is straightforward to extend Theorem 1 of [24] for the case of the shortest-path and column-first routing policies:

Theorem 2.1. Consider network nodes capable of storing arbitrary amounts of routing information. Suppose that straight-line, shortest-path or column-first routing is employed. Let $a_N > a \ln N/N$ for some $a > 1$. The network throughput scales as $\vartheta_N = \Theta(a_N^{-1/2}/N)$, i.e., there exist constants $0 < \zeta_1 < \zeta_2 < \infty$ such that

$$\mathbb{P}\left[\vartheta_N < \frac{\zeta_1}{\sqrt{a_N}N} \text{ is feasible}\right] \to 1$$

as $N \to \infty$, and

$$\liminf_{N\to\infty} \mathbb{P}\left[\vartheta_N > \frac{\zeta_2}{\sqrt{a_N}N} \text{ is feasible}\right] < 1.$$

Since the throughput scaling is better understood for the model with deterministic transmission times and time-slotted system, we keep the discussion on the trade-off between the throughput and size of routing tables restricted to that model (see Corollary 2.1). However, we point out that Theorem 2.1 can be extended to cover the model with exponentially distributed transmission times (a time-slotted system is not applicable in that case). Namely, it can be shown that the network is equivalent to a queueing Kelly network [42, Ch. 3] for which stability results are available. When the transmission times are random but not exponential, the network stability might depend on the routing policy and could be hard to evaluate in general.

2.3 Routing Tables

In this section we discuss the size of routing tables at individual nodes required for the correct operation of routing protocols. We say that a routing algorithm operates correctly if packets from all destinations can be delivered to their respective destinations in the cell-based algorithm.

2.3.1 Types of Routing Records

Routing tables contain *records*, i.e., the information on how packets to specific destination nodes need to be handled. The amount of information required for correct routing depends on the type of a record, i.e., whether packets contain additional metadata besides the destination address. Two cases are of interest:

- *Destination address only.* Due to the broadcast nature of packet transmissions in wireless networks, a packet can be received by all nodes that satisfy the interference constraint (2.1). Each node keeps a list of destination addresses they forward packets to. A node examines all the packets it receives and either retransmits them (a some point of time) if they are on the list, or simply discards them. In this case a routing table consists of a list of destination addresses. A mechanism is needed to prevent a node from transmitting the same packet multiple times.

- *Destination address & adjacent cell identifiers.* All cells are assigned labels such that each cell can uniquely identify all of its interfering cells. Each packet contains cell labels of the transmitting and receiving nodes, i.e., a pair (η, ω); this is in addition to the address of the destination node. Nodes maintain (possibly in compressed form) lists of triples of the form $(\alpha,\ \iota,\ \omega)$, where α is the address of a destination node, ι is an input cell and ω is an output cell. A node is responsible for forwarding packets with destination α from cell ι to cell ω (hence, the term input and output cells). A node receives only packets that contain address α and transmitting-cell label ι such that $(\alpha, \iota, \cdot)$ is on the node's list. Before retransmitting a packet, a node updates the packet's meta-data field with its own cell label and the receiving-cell label ω from the triple (α, ι, ω). We observe that in this case a cell can be treated as a switch/router with neighboring cells being input/output ports.

We focus on the second type of records for two reasons. First, compressing routing tables in the former case is difficult because the randomness of address locations results in high entropy of the list. On the other hand, having extra information (cell labels) helps with compression. Second, the overhead due to cell labels is small. Since each cell has at limited number c of interfering cells (see Section 2.2), it is sufficient to have only $(c + 1)$ labels to have each cell identify its interfering cells uniquely [15, p. 1092], i.e., $\lceil \log_2(c + 1) \rceil$ bits per cell label are sufficient. In addition, such labels might be available at very little cost due to the layered architecture of current networks, e.g., one could exploit the already existing layer-2 addressing space. Finally, we remark that one could also design routing tables with other types of records. For example, records of type (α, ω) are feasible, i.e., the input cell identifier is eliminated in the triple (α, ι, ω). However, such a design does not substantially decrease the record size in bits while the ability to compress routing tables is compromised.

2.3.2 Size of Routing Tables

Next, we turn our attention to sizes of routing tables. Due to the model(flat-label), a prefix-based compression [62] of routing tables is not applicable. Let $C_N(i)$ be the set of routing records in cell i needed to correctly forward packets between the sources and destinations. All the information required in a cell has to be stored at nodes located in that cell, i.e.,

$$C_N(i) = \bigcup_{n \in i} T_N(n),$$

where $T_N(n)$ is the set of records stored at node n. By $\|C_N(i)\|$ and $\|T_N(n)\|$ we denote the sizes of routing tables $C_N(i)$ and $T_N(n)$, respectively, measured in number of address length, i.e., if $\|C_N(i)\| = x$ then the size of compressed $C_N(i)$ in bits is given by $x\lceil \log_2 N \rceil$. Effectively, we assume that each node can hold a packet in its memory (since the packet size is lower bounded by $\lfloor \log_2 N \rfloor$ bits). Given that all nodes have identical capabilities (hardware), it is reasonable to assume that the routing information is spread evenly among nodes in a cell. This implies, for all $i \in n$,

$$\|T_N(n)\| \geq \left\lfloor \frac{\|C_N(i)\|}{k_N(i)} \right\rfloor \tag{2.4}$$

(recall that $k_N(i)$ is the number of nodes in the ith cell). Data stored at each node should contain not only a compressed routing table but also an decompression algorithm. Thus, it would perhaps be more appropriate to consider the Kolmogorov-Chaitin complexity [16, Ch. 7] of the routing table $C_N(i)$. However, estimating the Kolmogorov-Chaitin complexity of the routing table is difficult in general. Therefore, we will establish a lower bound by considering the traditional information-theoretic compression. In that case the compressed data do not have to contain a decompression algorithm (which is not true in the case of the Kolmogorov-Chaitin complexity). Note that considering specific algorithms for compression of general tables (such as [8]) will not provide a lower bound on $\|C_N(i)\|$.

Let $\psi_d(i)$ be the number of flows forwarded by cell i in direction d. Direction d indicates the neighbors of cell i between which it forwards packets (i.e. the input and output cells), e.g., south-to-north, north-to-east, etc. In other words, for a given cell, direction d can be represented by a pair of neighboring cell labels (ι, ω) with $\iota \neq \omega$. For all non-edge cells in the unit-area square, d can take 12 possible values; denote the set of these values by $\mathcal{D}$.

A routing table $C_N(i)$ can be thought of as a discrete-valued function[2] R that has two inputs and one output:

$$\omega = R(\alpha, \iota), \tag{2.5}$$

where ω is the output cell label computed based on the destination address α and the input-cell label ι.

The following lemma establishes a lower bound on the amount of routing information stored at individual nodes. The bound is based on the minimum value of $\psi_d(i)$ over $d \in \mathcal{D}$. The intuition behind the lemma is as follows. For a given cell i, consider the function $R(\cdot, \iota)$ for some specific ι. Let α be a destination chosen uniformly at random among destinations that have packets forwarded by cells ι and i. Then the randomness of $R(\alpha, \iota)$ is determined by $\psi_d(i)$ for $d = (\iota, \cdot)$, since $\psi_d(i)$ for $d = (\iota, \cdot)$ determines the likelihood of possible values of $R(\alpha, \cdot)$. The higher the randomness of $R(\alpha, \iota)$ (or entropy of $R(\cdot, \iota)$) the more information is need to correctly route packets. In the extreme case when $R(\alpha, \iota)$ is deterministic no information about individual destination needs to be stored – routing can be performed solely based on the input cell labels (in this case, all packets arriving from the input cell ι are forwarded to a single neighboring cell regardless of the destination address α.)

Lemma 2.3. Let $\psi := \min_{d \in \mathcal{D}} \psi_d(i)$. There exists $\zeta > 0$ such that for all $n \in i$

$$\|T_N(n)\| \geq \frac{\zeta \psi}{k_N(i) \ln N}.$$

[2]For compression of functions in communication contexts see [64, 45].

Proof. For notational simplicity let $\hat{\psi} := \sum_{d \in \mathcal{D}} \psi_d(i)$. It is sufficient to consider the case $\psi > 0$; otherwise the lemma holds trivially. The proof is based on a counting argument. For $d = (\iota, \omega)$ define

$$A_d(i) := \{\alpha : \omega = R(\alpha, \iota)\} \tag{2.6}$$

as a set of destination addresses forwarded by cell i in direction d. Because of the uniform distribution of addresses in the unit square, the minimum number of *bits* required to describe $R(\cdot, \cdot)$ is lower bounded by the number of bits required to describe $A_d(i)$ with the smallest cardinality (across $d \in \mathcal{D}$) so that $\{\alpha \in A_d(i)\}$ is verifiable. This number of bits is further lower bounded by $\log_2 \binom{\hat{\psi}}{\psi}$, since ψ is the cardinality of smallest of sets $A_d(i)$, $d \in \mathcal{D}$, and $\hat{\psi}$ is the total number of destination addresses forwarded by cell i; for a given set of cardinality $\hat{\psi}$, the number of subsets with cardinality ψ is given by $\binom{\hat{\psi}}{\psi}$. Furthermore, for $\psi > 0$,

$$\binom{\hat{\psi}}{\psi} \geq \frac{(\hat{\psi} - \psi)^{\psi}}{\psi!} \geq \left(\frac{\hat{\psi}}{\psi} - 1\right)^{\psi},$$

and hence, for some $\zeta > 0$,

$$\|C_N(i)\| \geq \zeta \frac{\psi}{\ln N};$$

we used the fact the address length is equal to $\lceil \log_2 N \rceil$ bits. Recalling (2.4) completes the proof. □

2.3.3 Compression

In this subsection we discuss a practical compression scheme that, in certain cases, can achieve performance that is close to the lower bound from Lemma 2.3. The algorithm exploits the difference in cardinalities of sets $A_d(i)$ for fixed i and different d's (see (2.6) for the definition), i.e., the compression is based on the existence of dominant routing directions. Consider a forwarding cell i and one of its neighbors ι. A single-argument routing/forwarding function $R(\cdot,\iota)$ (see (2.5)) can be expressed in the following form:

$$R(\alpha,\iota) = \begin{cases} \omega_2, & \text{if } \alpha \in A_{d_2}(i),\ d_2 = (\iota,\omega_2), \\ \omega_3, & \text{if } \alpha \in A_{d_3}(i),\ d_3 = (\iota,\omega_3), \\ \omega_1, & \text{otherwise,} \end{cases} \tag{2.7}$$

where $\iota \neq \omega_1 \neq \omega_2 \neq \omega_3$ are the four neighbors of cell i. Then, the function $R(\cdot,\iota)$ can be described by the two sets $A_{d_2}(i)$ and $A_{d_3}(i)$. Upon a reception of a packet, cell i examines whether the destination address α belongs to either A_{d_2} or A_{d_3}. If it does, the packet is forwarded to either cell ω_2 or cell ω_3; otherwise to cell ω_1. The number of bits required to describe $R(\cdot,\iota)$ in the form (2.7) is upper bounded by

$$(\|A_{d_2}(i)\| + \|A_{d_3}(i)\| + 1)\lceil \log_2 N \rceil,$$

for N large enough, where $\|A_d(i)\|$ is the number of destination addresses in set $A_d(i)$. Selecting d_1 in such a way that it satisfies

$$A_{d_1}(i) = \max_{d \in \{d_1,d_2,d_3\}} \|A_d(i)\|,$$

results in the minimum number of bits required (for this particular scheme). The algorithm is particularly effective when $\|A_{d_1}(i)\| \gg \|A_{d_2}(i)\| + \|A_{d_3}(i)\|$. In particular, for each input node ι, the largest set $A_{d_1}(i)$ can be eliminated from the its routing table. Instead, a packet from ι is directed according to d_1 by default if it carries an destination address that does not belong to either $A_{d_2}(i)$ or $A_{d_3}(i)$. We further exploit this idea in Section 2.5.

2.3.4 Example: Longest-Prefix Compression

The idea of biasing routing algorithms in order to reduce the size of forwarding tables is not limited to a specific compression scheme employed at a cell. In the following, we illustrate such a point by examining the well-known longest-prefix compression algorithm [67]. Namely, we compare the performance of compression scheme under various cardinalities of sets A_{d_i}'s (e.g. with or without dominant forwarding directions). For simplicity, suppose that packets to a given destination are forwarded by cell c from cell ι in the direction d_i with probability p_i for $1 \leq i \leq 3$. In a given cell c, forwarding directions for different destinations are independent. Then the total number flows forwarded by cell c from cell ι is binomially distributed with parameter (N, p), where $p = p_1 + p_2 + p_3$ is the probability that a given destination address has its packets forwarded by cell c from cell ι. An example of an uncompressed routing table is shown in the left in Fig. 2.1. Upon receiving a packet from cell ι, cell c looks up the destination address in the table and forwards the packet to the appropriate output cell (indexed by 1, 2 and 3).

Next, we describe a binary tree representation of the routing table in order to illustrate the procedure of longest-prefix compression. For convenience, assume that the number of nodes N is a power of 2; the addresses in the network correspond to the leaves of a complete binary tree of depth $\log_2 N$. For each address $\alpha \in \{0,1\}^{\log_2 N}$ we store the output cell identifier at a leaf node corresponding to α as shown on the top in Fig. 2.2. If packets to destination α are not forwarded by cell c from cell ι, then the leaf node corresponding to α is empty. Applying longest-prefix compression to the set of addresses is equivalent to iterating the following merging procedure on the tree. We say that a set of leaf nodes have non-conflicting indices if they are either empty or contain cell index i for some $1 \leq i \leq 3$. If a leaf and its sibling have non-conflicting indices, they are merged into a new node (either empty or with cell index i). We iterate the procedure until no more leaves can be merged. An example of a tree before and after

address		output cell
α_3	0010	1
α_4	0011	1
α_5	0100	1
α_6	0101	2
α_7	0110	3
α_8	0111	3
α_9	1000	1
α_{10}	1001	1
α_{11}	1010	2
α_{12}	1011	3
α_{13}	1100	1
α_{15}	1110	2
α_{16}	1111	1

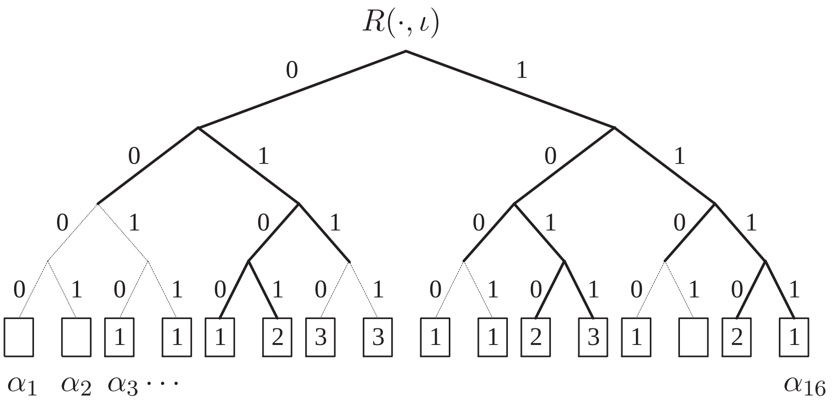
prefix	output cell
00	1
0100	1
0101	2
011	3
100	1
1010	2
1011	3
110	1
1110	2
1111	1

prefix	output cell
*	1
0101	2
011	3
1010	2
1011	3
1110	2

Figure 2.1: An example of $R(\cdot, \iota)$ for a fixed ι and its reduced form by applying longest-prefix compression. The compressing operation in the merging process of the tree representation is shown in Fig. 2.2. The table on the left (corresponding to the tree representation on the top in Fig. 2.2) is the routing table before compression; the table in the center corresponds to the tree on the bottom in Fig. 2.2, which is the routing table after the merging procedure; on the right is a fully compressed version of routing table.

this operation are shown in Fig. 2.2; the corresponding routing tables can be found in Fig. 2.1. After the merging process, each leaf corresponds to a prefix stored in the routing table. Note that a node x in the binary tree has its prefix stored in the routing table if and only if (i) all leaf descendants of x have non-conflicting indices with at least one leaf being non-empty, and (ii) there is a pair of leaf descendants of the parent of x such that they have conflicting indices. Observe that one of the output cell indices can be removed from the routing table without producing ambiguity. Without loss of generality, assume that $p_1 \geq \max\{p_2, p_3\}$. Then, prefixes with output cell indices 1 can be replaced by a single entry $(*, 1)$ in the table, as shown in the right in Fig. 2.1 (the symbol "$*$" represents a zero-length prefix). Whenever a packet arrives, it is forwarded to an output cell according to the longest-prefix in the routing table that matches the destination address.

We now estimate the expected number of bits needed to store all the prefixes. Clearly, the performance of longest-prefix compression depends on the content stored



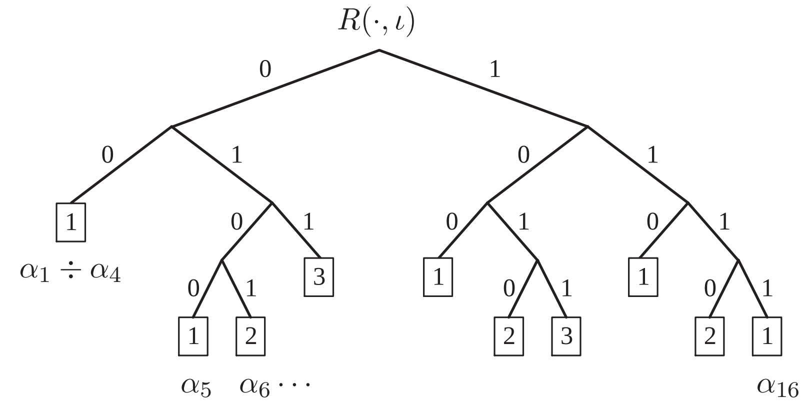


Figure 2.2: The example of $R(\cdot, \iota)$ in Fig. 2.1, represented by a binary tree. Each leaf of the tree corresponds to an address of length $\log_2 N$, specified by the unique path from the root. The tree can be compressed by merging leaves non-conflicting indices.

at each leaf. Define B_x^j as the event that the children of x can be merged to a node with index j, for $1 \le j \le 3$, and B_x^0 as the event that all of the descendant leaves of x are empty. Then, if node x is of height $l(x)$, the probabilities of B_x^j are as the follows:

$$\mathbb{P}\left[B_x^j\right] = \begin{cases} p_0^{2^{l(x)}}, \text{ if } j = 0; \\ (p_0 + p_j)^{2^{l(x)}} - p_0^{2^{l(x)}}, \text{ if } 1 \le j \le 3, \end{cases} \tag{2.8}$$

where $p_0 = 1 - p$ is the probability that a leaf node is empty. Let B_x be the event that x is a prefix leaf with cell index $j \neq 1$ (i.e. a leaf in the compressed (merged) tree with cell index $j \neq 1$). This event indicates that the prefix corresponding to x is stored in the compressed version of routing table. Then,

$$B_x = \bigcup_{j \in \{2,3\}} \left\{ B_x^j \cap \overline{\left\{ B_{\sigma(x)}^j \cup B_{\sigma(x)}^0 \right\}} \right\},$$

where $\sigma(x)$ is the sibling of x in the original tree, and $\overline{B}$ denotes the complement of B. Given the height $l(x)$ of x, the conditional probability of B_x, due to (2.8), satisfies

$$\begin{aligned} \mathbb{P}[B_x | l(x) = i] &= \sum_{j \in \{2,3\}} \mathbb{P}\left[B_x^j\right] \left(1 - \mathbb{P}[B_{\sigma(x)}^j] - \mathbb{P}[B_{\sigma(x)}^0]\right) \\ &= \sum_{j \in \{2,3\}} \left((p_0 + p_j)^{2^i} - p_0^{2^i}\right) \left(1 - (p_0 + p_j)^{2^i}\right). \end{aligned} \tag{2.9}$$

The expected number of bits needed for the routing table stored in cell c is given by

$$\begin{aligned} \mathbb{E}\|C_N(n)\| &= \mathbb{E}\left[\sum_{i=0}^{\log_2 N} \sum_{x : l(x) = i} (\log_2 N - i) \cdot 1_{\{B_x\}}\right] \\ &= \sum_{i=0}^{\log_2 N} 2^{\log_2 N - i} (\log_2 N - i) \mathbb{P}[B_x | l(x) = i]. \end{aligned} \tag{2.10}$$

Next, we examine two contrasting cases that illustrate the impact of the routing algorithm on the size of routing tables:

- *Uniform forwarding directions:* $p_1 = p_2 = p_3 = p/3$.

  From (2.8), it follows that

$$\mathbb{P}[B_x | l(x) = i] = 2\left((1-2p/3)^{2^i} - (1-p)^{2^i}\right)\left(1-(1-2p/3)^{2^i}\right). \tag{2.11}$$

  Note that a throughput-efficient routing algorithm requires that $p = p(N) \to 0$ as $N \to \infty$, i.e. each node forwards packets for only a negligible fraction of all nodes. The summation in (2.10) can be lower bounded by a single term corresponding to $i = \lfloor -\log_2 p \rfloor$; in that case, $\mathbb{P}[B_x|l(x) = i]$ is $\Theta(1)$, as $N \to \infty$. Thus, (2.10) and (2.11) yield

$$\begin{aligned}\mathbb{E}\|C_N(n)\| &\geq Np\ \log_2(Np)\ \mathbb{P}[B_x|l(x) = \lfloor -\log_2 p \rfloor] \\ &= c_1(Np \ln N)(1+o(1)),\end{aligned} \tag{2.12}$$

  as $N \to \infty$, for some constant $c_1 > 0$. The last equality is due to the fact that p must dominate $O(1/N)$; indeed, a packet with a given destination address is typically forwarded by at most a negligible fraction of nodes in the network. Note that the expected number of bits needed to store the routing table before compression is $Np \ln N$, which is of the same order as the right-hand side of (2.12) (the empty leaf nodes need not be stored). Hence, the longest-prefix algorithm results in only a constant factor reduction of the size of routing tables when there the forwarding directions is uniform.

- *Dominant forwarding direction:* $p_1 \gg p_2 = p_3$. Note that this corresponds to $\|A_{d_1}\| \gg \|A_{d_2}\| + \|A_{d_3}\|$ (See the previous subsection).

  In this case, the conditional probability in (2.9) evaluates to

$$\mathbb{P}[B_x|l(x) = i] = 2\left((1-p+p_2)^{2^i} - (1-p)^{2^i}\right)\left(1-(1-p+p_2)^{2^i}\right).$$

Thus, as $N \to \infty$, the expected number of bits needed for the routing table is upper bounded by

$$\begin{aligned} \mathbb{E}\|C_N(n)\| &\leq \log_2 N \left(Np \log_2(Np) \mathbb{P}[B_x | l(x) = \lceil -\log_2 p \rceil]\right) \\ &= c_2 \, \frac{p_2 \ln N}{p_1} \, (Np \ln N) \left(1 + o(1)\right), \end{aligned}$$

for some constant $c_2 < \infty$. That is, if $p_1 \gg p_2 \ln N$, then the size of routing table is compressed with reduction of factor proportional to $p_1/(p_2 \ln N) \gg 1$. Therefore, the more biasing the dominant forwarding direction is, the better the performance of the longest-prefix compression.

2.4 Shortest-Path Routing

In this section we focus on two routing algorithms: straight-line and shortest-path. The straight-line algorithm is deterministic, i.e., for a fixed pair of cells it always produces the same route. On the other hand, the shortest-path algorithm is randomized.

In addition to referring to cells by a single index $i \in \{1, \ldots, N\}$, we also refer to cells by a pair of indices (i, j), $i, j \in \{1, \ldots, 1/\sqrt{a_N}\}$ when convenient. These indices specify the relative position of cells with respect to the lower-left corner of the unit square. The cells in the lower-left and upper-right corners are labeled by $(1, 1)$ and $(1/\sqrt{a_N}, 1/\sqrt{a_N})$, respectively. When cell locations are of significance we will use the two-dimensional indexing.

Consider a pair of source-destination cells as shown in Fig. 2.3. The shortest path (in the number of hops) between these cells is not unique. In fact, given that the source and destination nodes are located in cells (m, n) and (u, v), respectively, the number of shortest paths is given by $\binom{|u-m|+|v-n|}{|u-m|}$; each of these paths is $|u - m| + |v - n|$ hops long. Unless the source and destination cells are located in the same row or column, the destination cell can be reached via two different neighbors of the source. Given that the

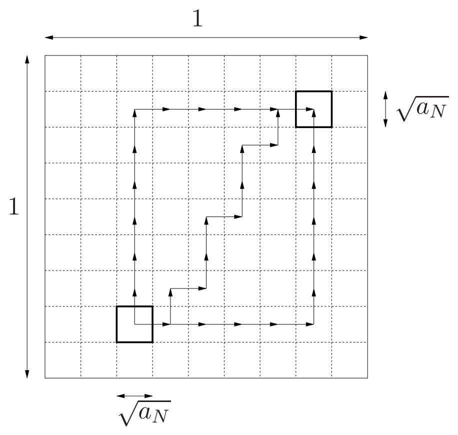


Figure 2.3: There exist multiple shortest paths (in the number of hops) between two cells located in different rows and columns. Three different paths are shown in this example. When a cell can achieve the same number of hops to the destination cell via 2 different neighbors, it chooses one at random.

sole goal is to minimize the number of hops, neither of the neighbors is preferred over the other one. Thus, our assumption that cells choose uniformly at random between the alternatives (neighbors). Furthermore, in the absence of geographic information at individual cells, random choice of neighbors appears to be a natural solution.

The next lemma establishes a lower bound on the number of forwarded flows in all directions by a large fraction of cells in the network. This result coupled with the fact that most cells forward packets to $\Theta(\sqrt{a_N}N)$ destinations (because a typical distance between a source and destination is $\Theta(1/\sqrt{a_N})$ cells) implies that the forwarded flows are evenly spread among the set $\mathcal{D}$ (modulo a pre-factor). For a set of nodes or cells $\mathcal{S}$, let $\|\mathcal{S}\|$ be the cardinality of the set.

Lemma 2.4. Consider either the straight-line or shortest-path routing algorithm. Let $a_N \geq 1/N$ such that $a_N \to 0$ as $N \to \infty$. For arbitrary $\delta \in (0,1)$ there exists a sequence of sets $\mathcal{C}_N \subseteq \{1, 2, \ldots, a_N^{-1}\}$ such that $a_N\|\mathcal{C}_N\| \geq 1 - \delta$ for all N large enough and

$$N\mathbb{P}[\psi_d(i) < \varepsilon\sqrt{a_N}N] \to 0,$$

as $N \to \infty$, for some $\varepsilon > 0$, all $i \in \mathcal{C}_N$ and all $d \in \mathcal{D}$.

Proof. See Appendix A.2. □

Remark 2.1. The conditions of the lemma on the cell size are not restrictive: (i) The case $a_N < 1/N$ is not relevant to the cell-based algorithm in view of Lemma 2.1; (ii) When $\lim a_N > 0$ only $\Theta(1/N)$ throughput per source-destination pair can be achieved, cf. Theorem 2.1. However, $\Theta(1/N)$ throughput can be achieved without multi-hop forwarding.

The following theorem is the first main result of this chapter. It characterizes the size of routing tables at individual nodes under the straight-line and shortest-path routing algorithms. Effectively, the theorem states that within our model these algorithms are not scalable in the sense that they require excessively large routing tables when throughput close to the limiting one is desired ($a_N = o(\ln^2 N)$ as $N \to \infty$, cf. Fig. 2.3).

Theorem 2.2. Consider either the straight-line or shortest-path routing scheme. Let $a_N > a \ln N/N$, for some $a > 1$, such that $a_N \to 0$ as $N \to \infty$. For arbitrary $\delta \in (0, 1)$, routing tables at $(1 - \delta)N$ nodes scale as $\Omega(a_N^{-1/2}/\ln N)$ as $N \to \infty$. That is, there exists a sequence of sets $\mathcal{N}_N \subseteq \{1, 2, \ldots, N\}$ with $\|\mathcal{N}_N\| \geq (1 - \delta)N$, and $\varepsilon > 0$ such that, as $N \to \infty$,

$$\mathbb{P}\left[\min_{n \in \mathcal{N}_N} \|T_N(n)\| \geq \frac{\varepsilon}{\sqrt{a_N} \ln N}\right] \to 1,$$

i.e., the size of the routing tables increases without a bound as the network size $N \to \infty$, since $a_N > a \ln N/N$ implies $\varepsilon/(\sqrt{a_N} \ln N) \to \infty$, as $N \to \infty$.

Remark 2.2. The theorem also holds for a wide set of network topologies that satisfy Lemma 2.4. Effectively, the theorem holds for topologies under which there is no significant correlation between input and output cells (see Section 2.3).

Proof. We start with obtaining an estimate on the number of nodes that forward a large number of flows in all 12 possible directions. To this end, let $\mathcal{C}_N$ be a set of cells with $\|\mathcal{C}_N\| \geq (1 - \zeta)a_N^{-1}$ for some $\zeta \in (0, 1)$; recall that the total number of cells is a_N^{-1}.

It is appropriate to think of $\mathcal{C}_N$ as the set of cells that are not "too close" to the edges of the unit square. De Morgan's law and the union bound yield

$$\begin{aligned}
\mathbb{P}\left[\min_{i\in\mathcal{C}_N}\min_{d\in\mathcal{D}}\psi_d(i) \geq \xi\sqrt{a_N}N\right] &= \mathbb{P}\left[\bigcap_{i\in\mathcal{C}_N}\bigcap_{d\in\mathcal{D}}\{\psi_d(i) \geq \xi\sqrt{a_N}N\}\right] \\
&= 1-\mathbb{P}\left[\bigcup_{i\in\mathcal{C}_N}\bigcup_{d\in\mathcal{D}}\{\psi_d(i) < \xi\sqrt{a_N}N\}\right] \\
&\geq 1-\sum_{i\in\mathcal{C}_N}\sum_{d\in\mathcal{D}}\mathbb{P}\left[\psi_d(i) < \xi\sqrt{a_N}N\right] \\
&\geq 1-\|\mathcal{D}\|\|\mathcal{C}_N\|\max_{i\in\mathcal{C}_N}\max_{d\in\mathcal{D}}\mathbb{P}[\psi_d(i) < \xi\sqrt{a_N}N] \\
&> 1-12N\max_{i\in\mathcal{C}_N}\max_{d\in\mathcal{D}}\mathbb{P}[\psi_d(i) < \xi\sqrt{a_N}N],
\end{aligned}$$

where the last inequality follows from $\|\mathcal{D}\| = 12$ and $\|\mathcal{C}_N\| \leq N$. The preceding inequality and Lemma 2.4 imply that there exists set of cells $\mathcal{C}_N$ that which contains $(1-\zeta)$ fraction of cells such that, as $N \to \infty$,

$$\mathbb{P}\left[\min_{i\in\mathcal{C}_N}\min_{d\in\mathcal{D}}\psi_d(i) \geq \xi\sqrt{a_N}N\right] \to 1, \tag{2.13}$$

i.e., $(1-\zeta)$ fraction of cells forward at least $\xi\sqrt{a_N}N$ flows in *all* directions.

Now consider a cell $i \in \mathcal{C}_N$. According to Lemma 2.3 the following bound holds for all nodes n located within the cell i:

$$\|T_N(n)\| \geq \zeta\frac{\min_{d\in\mathcal{D}}\psi_d(i)}{k_N(i)\ln N}, \tag{2.14}$$

for some $\zeta > 0$. Define a set $\mathcal{N}_N$ of nodes located within cells in the set $\mathcal{C}_N$

$$\mathcal{N}_N := \{n \in \{1,2,\ldots,N\} : n \in i \in \mathcal{C}_N\},$$

and note that by the strong law of large numbers

$$\liminf_{N\to\infty} \frac{\|\mathcal{N}_N\|}{N} > 1-\delta, \tag{2.15}$$

where $\delta \in (0,1)$ is such that $\delta \to 0$ as $\zeta \to 0$. By applying the minimum operator on both sides of (2.14), we obtain a lower bound on the size of routing tables at all nodes in set $\mathcal{N}_N$:

$$\begin{aligned}\min_{n\in\mathcal{N}_N} \|T_N(n)\| &\geq \zeta \min_{n\in\mathcal{N}_N} \left\{ \frac{\min_{d\in\mathcal{D}} \psi_d(i(n))}{k_N(i(n)) \ln N} \right\} \\ &= \zeta \min_{i\in\mathcal{C}_N} \left\{ \frac{\min_{d\in\mathcal{D}} \psi_d(i)}{k_N(i) \ln N} \right\} \\ &\geq \zeta \frac{\min_{i\in\mathcal{C}_N} \min_{d\in\mathcal{D}} \psi_d(i)}{\max_{i\in\mathcal{C}_N} k_N(i) \ln N},\end{aligned}$$

where $i(n)$ is the cell that contains the node n. Finally, the preceding inequality, (2.13), Lemma 2.2 and (2.15) imply the statement of the theorem. $\square$

Theorem 2.2 allows one to quantify the required memory at network elements for a given cell tessellation parameter a_N. Since this parameter a_N also determines the throughput for the model with deterministic transmission times (see Theorem 2.1), one has that the amount of available memory at individual nodes limits the maximum throughput. The following Corollary establishes the dependency of the throughput on the memory size at individual nodes.

Corollary 2.1. Consider either the shortest-path or straight-line routing scheme. Let each node be capable of storing b_N addresses in its routing table when the network consists of N nodes. If $b_N = O(\sqrt{N/\ln N})$ as $N \to \infty$, then there exist $0 < \zeta_1 < \zeta_2 < \infty$ such that

$$\mathbb{P}\left[\vartheta_N < \zeta_1 b_N/N \text{ is feasible}\right] \to 1, \tag{2.16}$$

as $N \to \infty$, and

$$\liminf_{N\to\infty} \mathbb{P}\left[\vartheta_N > \zeta_2 b_N \ln N/N \text{ is feasible}\right] < 1. \tag{2.17}$$

Remark 2.3. If $b_N = \Theta(N/\ln N)$ then the fraction that determines ϑ_N satisfies $b_N/N = \Theta(1/\sqrt{N \ln N})$. Given that the maximum achievable throughput under the cell-based algorithm and column-first routing is $\Theta(1/\sqrt{N \ln N})$, increasing b_N beyond $\Theta(N/\ln N)$ does not improve the throughput.

Proof. We prove (2.16) first. Set $\sqrt{a_N} = \lceil c/b_N \rceil$ for some constant $c > 0$. Due to the assumption $b_N = O(\sqrt{\ln N/N})$ it is possible to choose c such that $a_N > a \ln N/N$ with $a > 1$, i.e., the assumption of Theorem 2.1 is satisfied. Hence, there exists $\xi_1 > 0$ such that with no limit on the size of routing tables we have

$$\mathbb{P}\left[\vartheta_N < \frac{\xi_1}{\sqrt{a_N}N} \text{ is feasible}\right] \to 1, \tag{2.18}$$

as $N \to \infty$. Next we establish an upper bound on the size of a routing tables in all cells. The size of the routing table in cell i is upper bounded by the number of flows traversing the cell, $\sum_{d\in\mathcal{D}} \psi_d(i)$. If throughput ϑ_N is achievable, then there exists a fixed $\delta > 0$ such that for all $1 \le i \le 1/a_N$

$$\sum_{d\in\mathcal{D}} \psi_d(i) < \frac{\delta}{\vartheta_N},$$

where δ can be interpreted as the maximum transmission rate of a cell. Limit (2.18) and the preceding inequality yield, as $N \to \infty$,

$$\mathbb{P}\left[\max_{1\le i\le 1/a_N} \sum_{d\in\mathcal{D}} \psi_d(i) < \frac{\delta\sqrt{a_N}N}{\xi_1}\right] \to 1. \tag{2.19}$$

On the other hand, the minimum number of routing records that can be stored in a cell can be estimated by Lemma 2.2:

$$\mathbb{P}\left[b_N \min_{1\leq i\leq 1/a_N} k_N(i) > \varepsilon\sqrt{a_N}N\right] \to 1, \tag{2.20}$$

as $N \to \infty$, for some $\varepsilon > 0$; we also used the relationship between a_N and b_N. Combining (2.19) and (2.20) yields that for ξ_1 large enough each cell has enough memory space to store all the required routing records in the limit as $N \to \infty$:

$$\mathbb{P}\left[\max_{1\leq i\leq 1/a_N} \sum_{d\in\mathcal{D}} \psi_d(i) < b_N \min_{1\leq i\leq 1/a_N} k_N(i)\right] \to 1.$$

Therefore, (2.18) holds also for the case when each node can store b_N addresses rather than an infinite number of them. Recalling that $\sqrt{a_N} = \lceil c/b_N \rceil$ renders (2.16).

Now we consider (2.17). It suffices to consider only the case $a_N > a \ln N/N$ with $a > 1$ since otherwise (2.17) holds due to the fact that $\Theta(1/\sqrt{N \log N})$ is the highest throughput achievable with the cell-based algorithm [33, 24, 25]. For the routing protocol to operate correctly all the required routing records need to be stored in nodes' memory, implying that b_N has to satisfy

$$\begin{aligned} b_N &\geq \max_{1\leq n\leq N} \|T_N(n)\| \\ &\geq \max\left\{1, \min_{n\in\mathcal{N}_N} \|T_N(n)\|\right\}, \end{aligned}$$

where $\mathcal{N}_N \subseteq \{1, \ldots, N\}$; if $b_N = 0$ than there is no memory available for routing tables at all, and, thus, one requires $b_N \geq 1$. When $a_N > a \ln N/N$, $a > 1$, the preceding relationship and Theorem 2.2 yield, for some $\varepsilon > 0$,

$$\mathbb{P}\left[b_N > \frac{\varepsilon}{\sqrt{a_N}\ln N}\right] \to 1 \tag{2.21}$$

as $N \to \infty$. Theorem 2.1 guarantees an existence of $\xi_2 < \infty$ such that

$$\liminf_{N\to\infty} \mathbb{P}\left[\vartheta_N > \frac{\xi_2}{\sqrt{a_N}N} \text{ is feasible}\right] < 1.$$

Combining this limit with (2.21) concludes the proof. □

2.5 Information-Efficient Routing

In this section we establish that there exists an information-efficient routing algorithm. In particular, we show that column-first routing (see Section 2.2) has such a property. That is, the scheme can operate correctly in large networks without sacrificing the throughput even when each node has enough memory to store only a finite number of routing records (addresses). The compression scheme described in Section 2.3.3 can be used to efficiently represent routing tables when column-first policy is employed. We stress that the column-first policy is only one of possible information-efficient policies; we focus on this particular policy due to its simplicity. Also, it is possible to devise an information-efficient policy that is distributed and does not require global node coordinates (see Section 2.6).

Consider a non-edge cell and a packet to be forwarded through this cell. If the packet is forwarded along a row, then the packet will remain in the same row until it reaches its destination. This implies that a packet arriving from the west or east neighboring cell has to be forwarded to the east and west neighbor, respectively. In other words, no packet is forwarded in the east-to-south, east-to-north, west-to-south and west-to-north direction, i.e., $\psi_d(i) = 0$ for these directions. For most of the cells, if a packet arrives from the south neighbor, it is more likely to be forwarded to the north neighbor, since

$$\mathbb{E}\psi_d(i) = a_N(l-1)(1 - l\sqrt{a_N}),$$

where l is the row the cell is located at and d is the south-to-north direction, and

$$\mathbb{E}\left[\psi_{d_1}(i) + \psi_{d_2}(i)\right] = a_N(l-1)(\sqrt{a_N} - a_N),$$

where d_1 and d_2 are south-to-west and south-to-east directions, respectively[3]. Therefore, a cell can perform correct routing by maintaining only 4 sets of destination addresses $\{A_{d_i}\}$. Packets destined to addresses in $\{A_{d_i}\}$ are forwarded from the south or north cell to either the east or west cell, depending to which A_{d_i} the address belongs. All other packets are forwarded straight across the cell (north-to-south, south-to-north, east-to-west, west-to-east). With this kind of representation of the routing functions $R(\cdot,\cdot)$ each destination contributes at most one routing record to one cell in the network. This follows from the fact that each route experiences at most one $90°$-change of direction.

The next proposition is the second main result of the chapter.

Proposition 2.1. Let Π_b the column-first routing algorithm in a network of nodes capable storing b routing records. If $a_N > a \ln N/N$ for some $a > 1$, then there exists $b < \infty$ such that $\mathbb{P}[\Pi_b \text{ operates correctly}] \to 1$ as $N \to \infty$.

Proof. Let $\nu_N(i)$ be the number of routes that undergo a change in direction in a cell i when the network is of size N. Then, the maximum of $\nu_N(i)$ across all cells can be estimated using the union bound:

$$\mathbb{P}\left[\max_{1\le i\le 1/a_N} \nu_N(i) \le \gamma a_N N\right] \ge 1 - \sum_{i=1}^{a_N^{-1}} \mathbb{P}[\nu_N(i) > \gamma a_N N].$$

For a fixed cell i, a route changes its direction at this cell only if the source is in the same column and the destination is in the same row (but not in the cell i). For $j = 1, \ldots, N$, let $I_j = 1$ if node j originates a flow that changes direction at cell i, and

[3]The disparity of $\psi_d(i)$ for direction i's that allows for an efficient representation of routing tables was exploited in [7] to obtain an approximation of the throughput of lattice networks with deterministic service times and finite buffers.

$I_j = 0$ otherwise. Then, we have

$$\nu_N(i) = \sum_{j=1}^{N} I_j. \tag{2.22}$$

Due to our assumption of independence of traffic routes (see Section 2.2), the sequence $\{I_j\}_{j=1}^{N}$ is an i.i.d. sequence of Bernoulli random variables with $\mathbb{E}I_j = (\sqrt{a_N} - a_N)^2 < a_N$; the probability that the source node is in the same column as cell i (but not in cell i) is $(\sqrt{a_N} - a_N)$, the same applies to the destination. Then, (2.22), Markov's inequality and the independence of source-destination pairs yield, for arbitrary $s > 0$,

$$\begin{aligned}
\mathbb{P}[\nu_N(i) > \gamma a_N N] &= \mathbb{P}\left[\prod_{j=1}^{N} e^{sI_j} > e^{s\gamma a_N N}\right] \\
&\leq e^{-s\gamma a_N N}\mathbb{E}\left[e^{sI_1}\right]^N \\
&\leq e^{-s\gamma a_N N}\left(1 + a_N(e^s - 1)\right)^N \\
&= e^{-s\gamma a_N N} e^{N\ln(1+a_N(e^s-1))} \\
&\leq e^{-a_N N(\gamma s - e^s + 1)},
\end{aligned}$$

where the last inequality is due to $\ln(1+x) \leq x$ for $x \geq 0$. Given the assumptions of the proposition, it is possible to select s and γ such that, as $N \to \infty$,

$$\mathbb{P}\left[\max_{1\leq i\leq 1/a_N} \nu_N(i) \leq \gamma a_N N\right] \to 1. \tag{2.23}$$

The required amount of storage at individual nodes for correct operation of the routing algorithm in cell i is upper bounded by $\|T_N(n)\| \leq \zeta\lceil \nu_N(i)/k_N(i)\rceil$, where $\zeta > 1$ accounts for bits needed to represent cell labels. This bound is based on the assumption that all the records can be stored in a distributed fashion within the cell.

Finally, (2.23) and Lemma 2.2 result in

$$\mathbb{P}\left[\max_{1\le i\le 1/a_N} \lceil \nu_N(i)/k_N(i)\rceil \le \frac{\gamma}{\varepsilon}\right] \ge \mathbb{P}\left[\max_{1\le i\le 1/a_N} \nu_N(i) \le \gamma a_N N,\ \min_{1\le i\le 1/a_N} k_N(i) \ge \varepsilon a_N N\right] \to 1,$$

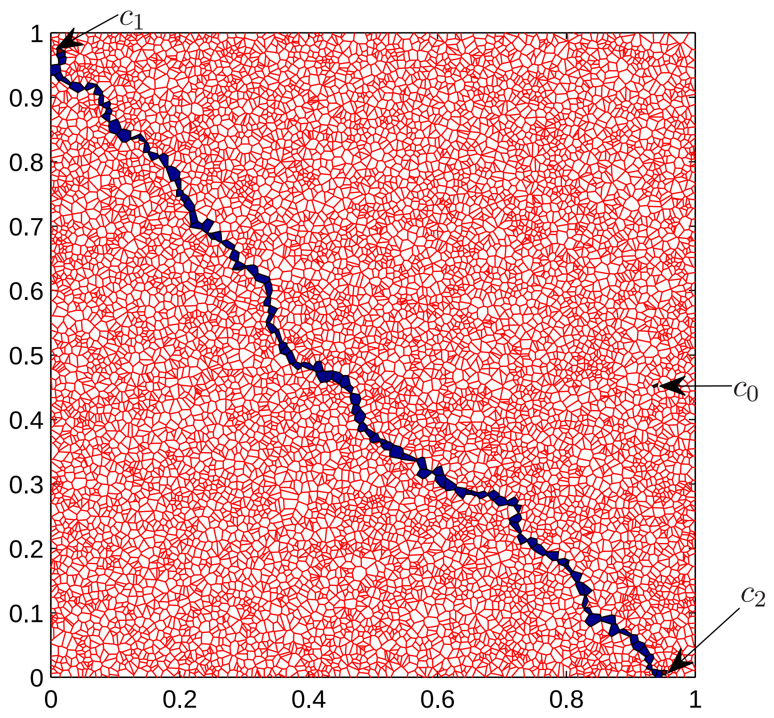
as $N \to \infty$. Choosing $b = \lceil \zeta\gamma/\varepsilon \rceil$ completes the proof. □

Proposition 2.1 and Theorem 1 from [37] suggest that nodes with limited memory can be used to create large multi-hop networks, at least when the transmission times are exponentially distributed. The protocol constructed in [37] utilized the column-first routing strategy. These two results indicate that memory bottlenecks can be avoided both in the data and control planes.

2.6 Distributed Approach

In the previous sections, the discussion focused on a unit square partitioned into square cells. It is of interest to consider information-efficient routing algorithms with more general cell shapes. Moreover, high-level notions of rows and columns do not usually pre-exist in distributed networks. The purpose of this section is to illustrate that information efficiency of routing protocols is not tied to regular cell shapes and pre-existing notions of rows and columns; detailed design and analysis of efficient distributed algorithms is beyond the scope of the discussion. Namely, we provide an example on how virtual rows and columns can be distributively constructed with dense Voronoi tessellations. Cells in the network do not have any information about the network topology besides the identity of their neighboring cells.

Consider the Voronoi tessellation of the unit square (see [63] for a discussion of Voronoi tessellation). Let $\{x_1, x_2, \ldots, x_{1/a_N}\}$ be a set of $1/a_N$ points on the unit square; each point represents a center of a cell, and hence there are $1/a_N$ cells in total. The set of



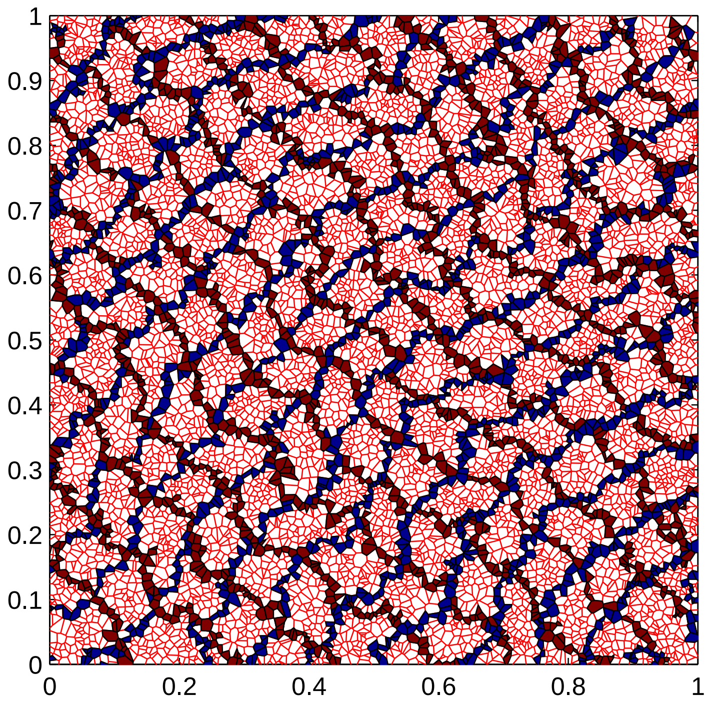


Figure 2.4: This figure provides an example on how virtual rows and columns for a Voronoi tessellated unit square are constructed by the distributed algorithm. A set of 10^4 points are selected uniformly at random as centers of the Voronoi cells, which satisfies the condition that rules out too eccentrically shapes. The figure on the top shows cells c_0, c_1, c_2, and the initial path P; in the figure on the bottom, only the virtual rows (columns) with indices of multiples of 6 are colored for clarity.

all points that are closer to x_i than to any of the other x_j's, for each i, i.e., $\{x \in (0,1)^2 : |x - x_i| \leq |x - x_j|, j \neq i\}$, is called a Voronoi cell. Such tessellation provides a method of grouping nodes that are physically close to each other and was considered in previous work on cell-based forwarding algorithms (e.g., see [33]). Following [33] (see Section IV. in [33]), we require that each cell contains a disk of area $\Theta(a_N)$ as $N \to \infty$, and is contained by a disk of area $\Theta(a_N)$; this condition rules out too eccentrically shaped cells. As in the previous sections, packets are forwarded between neighboring cells. The following algorithm constructs virtual rows and columns for a Voronoi tessellated unit square (an instance is shown in Fig. 2.4):

1. A randomly selected cell c_0 floods a distance counter that increases by hops over the network so that every cell is aware of the minimum distance (in hops) to c_0.

2. A cell c_1 with the longest hop distance to c_0 is identified. Cell c_1 performs another distance-counter flood to identify a furthest cell c_2 from c_1.

3. An initial path (a set of connected cells) P from c_1 to c_2 is created; random decisions are made whenever there is a tie.

4. Path P floods a distance counter; all of the cells in P have distance counters set to zero. The distance to P is equal to the minimum distance to a cell belonging to P. Cells with the same distance to P (and on the same side) form virtual rows.

5. A cell furthest away from P is identified as c_3; a cell furthest away from c_3 is identified as c_4. As in steps 3 and 4, cells c_3 and c_4 can be used to construct a path and thus the virtual columns.

Each time a distance counter is flooded, all cells receive the counter. Hence, all cells belongs to exactly one virtual row and one virtual column. Note that neighboring cells in a virtual row (column) can be separated by at most one cell with a different distance counter, due to the nature of distance counters. Hence, a cell needs to interact with at most all the two-hop neighbors. When a packet is forwarded along a virtual row or column, intermediate cells may need to forward the packet to multiple neighbors (i.e. the neighbors that have not yet received the packet). However, the number of forwarding actions taken remains constant. This is due to a constant number of neighbors for each cell.

After the one-time construction of virtual rows and columns, a column-first-like routing algorithm can be performed on the unit square. Given a transmission pair, the source and destination nodes forward a request along the column and row they belong to. The two requests intersect at some cell, which in turn keeps the record of the forwarding destination address in the routing table. It is straightforward to argue that routing by the virtual rows and columns requires the same (order of magnitude) amount of routing information as column-first routing with square cells of the similar size. The above construction of virtual rows (columns) can be generalized to a larger class of tessellation on convex regions. We stress that the above algorithm provides one way of constructing the virtual rows and columns, but is by no means exclusive.

2.7 Concluding Remarks

The amount of information required to route packets in a large wireless network can have a significant impact on the achievable network throughput. The widely considered shortest-path and straight-line algorithms require routing tables of excessive size when non-negligible throughput is desired (in comparison with the maximum throughput). On the other hand, when the column-first scheme is employed, each node has to store routing information on only a fixed number of destinations regardless of the network size.

CHAPTER 3

Scalable Scheduling

3.1 Introduction

In this chapter, we address the scheduling problem within the framework of a particular network topology, namely, a full bipartite graph that corresponds to the packet switch. High performance routers and switches, being the core of various existing and proposed architectures, are likely to remain indispensable parts of future networks. We develops a novel model of computation that enables efficient operation of packet switches with distributed scheduling; the model yields a new architecture and scheduling algorithms. Scalability of packet switches has been considered in the literature. In short, there exists a tradeoff between the switch performance and hardware complexity, as well as between the complexity of data-plane hardware (crossbars, memory banks, etc.) and control plane hardware (control unit). However, the studies in this domain are dominated by architectures with either a significant number of switching components and/or a centralized controller. The assumption of system coordination is typically implied in those models. Our algorithms are not based on this assumption. Namely, we design a fully distributed scheduling scheme for packet switches. Distributed operation is enabled by introducing a novel model of computation, and thus eliminating the standard model based on bits, registers, machine cycles and all related hardware. The new model of computation utilizing time encoding is capable of drastically reducing the complexity of the control plane, since inter-port communication on a single piece of switching fabric is not required. While a number of alternative models of computation have been proposed (molecular, quantum, membrane, etc.), at present devices based on these models do not outperform those using the standard model. In contrast, the architecture we propose provides performance that is comparable with known centralized architectures.

Our model of computation is inspired by neurons. In particular, the relevant quantities for determining a feasible schedule are represented by the length of time intervals instead of conventional bits in registers; hence, the term time encoding. Computation required to determine a state of the switching fabric can be performed efficiently under such representation. We point out that the model is potentially extendable to other inherently decentralized systems such as wireless networks. Our focus on the switch design problem is dictated by two factors: (i) switch scheduling problem is well-defined, structured and important in practice, and (ii) the resulting architecture can not only be implemented physically, but also enables switch operation at speeds that are comparable to switches based on conventional technology.

The remaining of this chapter is organized as follows. In the next section, we outline the basic switch model and briefly review relevant literature. Section 3.3 describes a novel model of computation based on time encoding, where time intervals are used to perform computations. In Section 3.4 we discuss a distributed scheduling algorithm utilizing this model and its implementation. A reversible version of our algorithm is presented in Section 3.5. Finally, concluding remarks can be found in Section 3.6.

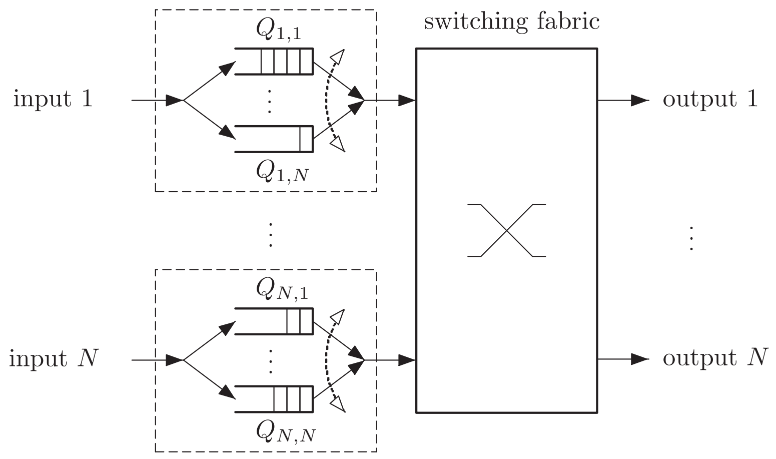


Figure 3.1: The structure of a Virtual-Output-Queued switch. Packets are enqueued at inputs based on their destinations. Each input/output can be connected to at most one output/input in every time slot.

3.2 Model

3.2.1 Virtual-Output-Queued switch

The Virtual-Output-Queued (VOQ) switch architecture has the following desirable properties: (i) it is based on a switching fabric operating at line speeds, while output-queued switches require hardware speedup; (ii) it does not suffer from performance degradation due to the head-of-line (HOL) blocking as in the case of input-queued switches; (iii) it can be implemented on a single switching fabric (crossbar); (iv) no reordering of packets is required. The reduced hardware requirements and improved performance come at the cost of increased control complexity. Hence, the key performance metrics (throughput, delay, etc.) crucially depend on the employed scheduling algorithm. In particular, it has been shown that a greedy algorithm performs poorly under certain traffic scenarios [47].

The structure of an $N \times N$ VOQ switch is shown in Fig. 3.1. Packets arriving to input i with destination j are enqueued in the virtual output buffer (i, j). We assume that the buffer is of infinite size, that time is slotted, and that every packet is of uniform size such that it takes exactly one time slot to transmit a packet. The number of packets in queue (i, j) at time slot t is denoted by $Q_{i,j}(t)$. Let $A_{i,j}(t)$ be the number of packets arriving to queue (i, j) at time t. The queue occupancies evolve in time according to

$$Q_{i,j}(t+1) = Q_{i,j}(t) - S_{i,j}(t) + A_{i,j}(t), \tag{3.1}$$

where $S_{i,j}(t) = 1$ if queue (i, j) is non-empty and is served at time t, i.e., a packet is switched from input i to output j; otherwise $S_{i,j}(t) = 0$. Event $\{S_{i,j}(t) = 1\}$ indicates that the switching fabric is configured to a state such that one packet in queue (i, j) is switched, and, thus, $S_{i,j}(t)$ depends on the scheduling algorithm. The switching fabric is physically constrained to transmit at most one packet for each input/output port in a time slot, i.e.,

$$\sum_{i=1}^{N} S_{i,j}(t) \leq 1 \quad \text{and} \quad \sum_{j=1}^{N} S_{i,j}(t) \leq 1. \tag{3.2}$$

A scheduling algorithm determines the value of $\{S_{i,j}(t)\}$ subject to (3.2). Hence, in view of (3.1), the impact of scheduling algorithm on the evolution of queue occupancy processes is only through the values of $\{S_{i,j}\}$.

In the rest of the discussion, we assume that the arrival processes satisfy a strong law of large numbers, i.e., with probability 1,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} A_{i,j}(t) = \lambda_{i,j}, \quad i, j = 1, \ldots, N, \tag{3.3}$$

where $\lambda_{i,j}$ is the traffic intensity at virtual queue (i, j). We say that a scheduling algorithm is *rate stable* if, with probability 1,

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} S_{i,j}(t) = \lambda_{i,j}, \quad i, j = 1, \ldots, N,$$

for any arrival process satisfying (3.3). The input traffic is called *admissible* if

$$\lambda_{\cdot,j} = \sum_{i=1}^{N} \lambda_{i,j}(t) < 1 \quad \text{and} \quad \lambda_{i,\cdot} = \sum_{j=1}^{N} \lambda_{i,j}(t) < 1. \tag{3.4}$$

It is straightforward to show that no scheduling algorithm is rate stable under non-admissible traffic ($\lambda_{\cdot j} > 1$ or $\lambda_{i\cdot} > 1$ for some i, j). In that case, $Q_{i,j}(t)$ increases over time without a bound regardless of the scheduling algorithm for some i and j.

3.2.2 Maximum-Weight-Matching scheduling

Maximum-Weight-Matching (MWM) algorithm is rate stable under any admissible traffic pattern and is known to provide high throughput and low delay [20, 17, 35, 31, 59]. We sketch the operation of MWM below. The state of the switch can be described by a bipartite graph, with vertices representing input and output ports, and edges representing the corresponding virtual buffers. In each time slot t, edge (i, j) is assigned a weight $W_{i,j}(t)$ that measures the congestion level of the corresponding queue, e.g., $W_{i,j}(t) = Q^{\beta}_{i,j}(t)$, $\beta > 0$ [3, 54]. In the context where delay is concerned, the weight function is oftentimes associated with the age of the packets in the queue. (Note that an ideal weight function for our scheduling algorithm in Section 3.4 uses only local information from individual queues, in order to achieve distributiveness.) A discussion on the selection of weight functions can be found in [44]. The MWM algorithm selects a matching (set of independent input-output pairs) with the highest sum of weight and schedules transmissions according to the matching. When $W_{i,j}(t) = 1_{\{Q_{i,j}(t)>0\}}$, the algorithm reduces to the Maximum-Size-Matching (MSM) algorithm. In [79, 54] it was shown that the MWM algorithm is rate stable with Bernoulli arrival processes when no input or output port is overloaded (admissible traffic); the result was extended for more general arrival processes in [20]. MWM-like algorithms were also argued to perform well when applied in the networks of constrained queues [77] and switches [53].

Furthermore, the MWM algorithm with appropriately chosen weight functions performs optimally under the heavy load scenario not only in switches [71, 75] (the cross-bar switch considered here is a special case of the generalized switch examined in [75]), but also in stochastic processing networks [18, 4] (optimality is considered with respect to holding cost).

Nonetheless, computing the maximum weight matching at line speeds is a challenge. Existing algorithms [17, 35, 31] are either of significant complexity or poor scalability, i.e., requiring a large number of arithmetic operations for a large number of input/output ports. Hence, a number of practical algorithms were developed, including iSLIP [52], MUCS [19] and RPA [1]. These algorithms do not attempt to approximate the MWM algorithm explicitly and are inferior to the MWM when the input traffic is not uniform [30]. In particular, iSLIP is a maximal matching scheduling algorithm; no unconnected pair of input and output do not have packets for each other in a maximal matching. It was shown in [20] that all maximal matching algorithms are stable at speedup 2 when the traffic is Bernoulli i.i.d. and the delay performance is investigated in [70]. On the other hand, algorithms based on MWM approximations were also considered, see e.g. [78, 29, 21]. They utilized the observation that queue occupancies exhibit correlation in time, and hence, the weight of a matching does not change significantly over a small time interval. Recently, an algorithm based on an auction is proposed in [13].

Distributed switch scheduling on a single crossbar poses additional challenges. Iterative algorithms such as iSLIP [52] and the bidding algorithm [13] can be considered distributed but they require multiple rounds of "information exchange". Namely, input and output ports communicate between each other in order to establish a feasible schedule. Note that the problem of switch scheduling can be viewed as an instance of a general problem of scheduling on interference graphs. Interference graphs were also considered in the context of scheduling in wireless networks. In such networks information exchange (message passing) is even costlier since control packets utilize the same resource (bandwidth) that needs to be allocated (scheduled). Algorithms that require a large number of rounds of message exchanges are capable of supporting switch operation at limited speeds only since a non-negligible amount of time is needed for rounds to take place. On the other hand, schemes with simple implementations such as maximal scheduling (single round of scheduling) deliver only a fraction of the possible throughput [14, 72]. Distributed algorithms with low-complexity considered in the literature include local back-pressure policy [77] and the Longest-Queue-First (LQF) policy [23]. For further studies of distributed scheduling in wireless networks see [59, 26, 74] and references therein. The extension of our basic algorithm, the reversible algorithm described in Sect. 3.5, is conceptually closest to the distributed CSMA algorithm proposed in [39]; in particular, even though the algorithms are different, the structures of underlying Markov chains bear resemblance.

3.2.3 Algorithm comparisons

Although it is desirable to have a scheduling algorithm with a higher relative throughput, this oftentimes comes at the cost of larger time complexity. When comparing the performance of scheduling policies, both the quality and running time of decision making need to be taken into account. In particular, if an algorithm achieves relative throughput $\rho \in [0, 1]$ and takes τ seconds to determine a schedule, then the maximum theoretic throughput is at most $\rho\tau$ cells/s. For example, consider two scheduling algorithms $\mathcal{C}$ and $\mathcal{D}$, achieving 100% and 85% of the relative throughput, respectively. If it takes 10^{-8} s for $\mathcal{C}$ and 10^{-10} s for $\mathcal{D}$ to compute a schedule, then the maximum theoretic throughput is at most 10^8 cells/s for $\mathcal{C}$ and $85 \cdot 10^8$ cells/s for $\mathcal{D}$. Therefore $\mathcal{D}$ significantly outperforms $\mathcal{C}$ due to faster decision making.

3.3 Time-encoding computation

In this section, we discuss a model of computation based on time encoding. The standard model of computation is based on bits and elementary operations (e.g., summation comparison and multiplication) and is implemented in digital logic. Although some early computing machines were based on analog components, today digital circuits dominate the implementation of specialized and general-purpose processors. Nonetheless, alternative models of computation have received considerable attention (e.g., see [2, 60, 66]), most of which are yet to be physically implemented. On the other hand, the model of computation we propose can be integrated into the existing solid-state technology.

3.3.1 Time-encoding operations

In our model of computation, variables are represented with time intervals of possibly random lengths rather than deterministic bits. A time interval (and hence its length) can be defined by two impulses. Let X and Y be two variables. The following two basic operations can be utilized to implement time-encoding algorithms:

- *Minimum* ($\min\{X, Y\}$). Variables X and Y are represented by two constant time intervals with lengths X and Y, respectively. Define the new starting impulse as the beginning of the two aligned time intervals, and the new ending impulse as the earlier of the two ending impulses. Clearly, the length of the new time interval is $\min\{X, Y\}$.

- *Inverse sum* ($1/(X + Y)$). Variables X and Y are represented by two random-length time intervals. In particular, the length of the intervals are given by exponential random variables with means $1/X$ and $1/Y$; denote the lengths of these intervals by T_X and T_Y, respectively. The minimum of the two, $\min\{T_X, T_Y\}$, can be obtained by applying the minimum operator. Random variable $\min\{T_X, T_Y\}$

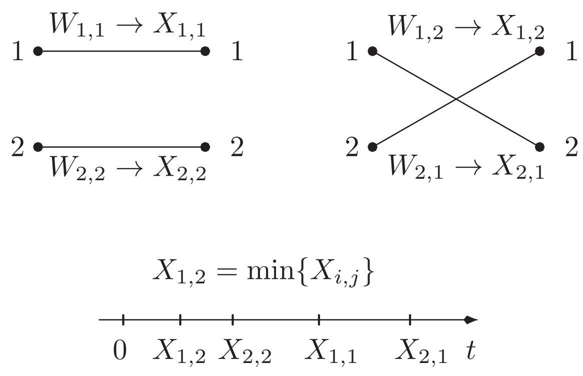


Figure 3.2: Illustration for the motivating example. The switching fabric has only 2 possible states shown on the top. On the bottom is the time line that indicates the expiration of the timers. In this example, the timers are set at $t = 0$. At $t = X_{1,2}$, the first timer to expire is the one associated with input-output pair $(1, 2)$. Thus, in this instance the switch is configured to the matching with weight M_2.

is exponentially distributed with mean $1/(X+Y)$. Hence, the output of the operator, $\min\{T_X, T_Y\}$, is only an approximation of the inverse sum, $1/(X + Y)$, but can be implemented in a distributed fashion.

3.3.2 A motivating example

Next, we illustrate how the basic time-encoding operations can be used with a motivating example. For simplicity, consider a 2×2 switch with input-output weights given by $[W_{i,j}]_{i,j=1,2}$. In this case there exist two matchings (switch configurations) with weights $M_1 = W_{1,1} + W_{2,2}$ and $M_2 = W_{1,2} + W_{2,1}$ (with corresponding connectivity patterns $\{(1, 1), (2, 2)\}$ and $\{(1, 2), (2, 1)\}$ as shown in Fig. 3.2). The MWM algorithm schedules a matching with the maximum weight. Given the standard model of computation based on digital logic (i.e., bits, registers, summations, comparisons, etc.), effectively a single entity is required in order to determine which matching has a higher weight. In particular, the entity needs to be aware of the values of $[W_{i,j}]_{i,j=1,2}$ so that M_1 and M_2 can be computed and compared.

Our distributed scheme using information encoded in timing is described next.

The scheme is randomized with the following intuition: the heavier the matching the more likely it is scheduled. In the first step each input-output weight is randomized. In particular, $W_{i,j}$ is replaced with $X_{i,j}$ where $X_{i,j}$ is exponentially distributed with $\mathbb{E}X_{i,j} = 1/W_{i,j}$. This is the step that enables distributed summation. Observe that the randomization can be implemented in a distributed fashion since each $X_{i,j}$ depends only on the corresponding value of $W_{i,j}$. In the second step, at some fixed time, say $t = 0$, each input-output pair (i, j) (virtual buffer) sets a timer with an expiration time $X_{i,j}$. Upon the expiration of timer (i, j), an attempt is made to connect input i and output j. Such a connection is feasible only if both input i and output j have not yet been connected to some other ports. The algorithm terminates when all finite timers expire. We call the algorithm distributed since each virtual queue attempts a transmission based on local information only. Note that in the 2×2 case the matching is scheduled whenever one of its two input-output pairs is scheduled first (equivalently, a timer expires first). For example, matching with weight M_1 is scheduled if and only if $\min\{X_{11}, X_{22}\} < \min\{X_{12}, X_{21}\}$. The key fact is that $\min\{X_{11}, X_{22}\}$ and $\min\{X_{12}, X_{21}\}$ are exponentially distributed with means $1/M_1$ and $1/M_2$, respectively. Effectively, we use the comparison of $\min\{X_{11}, X_{22}\}$ and $\min\{X_{12}, X_{21}\}$ (fully distributed based on timers) as a proxy for the comparison between M_1 and M_2. A straightforward computation shows that the two matchings are chosen with probabilities $M_1/(M_1 + M_2)$ and $M_2/(M_1 + M_2)$.

3.4 Distributed scheduling

The centralized implementation of the MWM algorithm requires the presence of a control unit in the switch architecture. Previous approaches to eliminate the control unit come at cost of increasing the complexity of the switching hardware. Informally, the idea is to shape the input processes (make the input traffic uniform) so that a simple scheduling algorithm can be employed, e.g. see the load-balanced router architecture [40, 41]. The increased complexity stems from the additional hardware required to reorder the packets (in order to restore the original sequencing) prior to their transmission to output ports. In contrast, our approach aims at developing a distributed scheduling algorithm (i.e. no control unit is required) that maintains the original order of packets within the switch. The key idea behind our architecture is to enable distributed summations so that matching weights can be computed without explicit information exchange between ports.

3.4.1 Basic algorithm

In this subsection, we describe a randomized algorithm $\mathcal{A}(W)$. The argument W refers to a function that is used to assign weights to input-output pairs. For example, if $W = Q^{\beta}$ (as proposed originally in [3]), then at time slot t the weight of the edge (i, j) in the corresponding input-output bipartite graph is given by $Q^{\beta}_{i,j}(t)$. We use $[T_{i,j}(t)]$ to denote an $N \times N$ matrix that is independent of the arrival processes. Elements of each matrix are independent and exponentially distributed random variables with unit mean; the matrices for different values of t are independent. The algorithm operates as follows.

Algorithm $\mathcal{A}(W)$ [55]

For each time slot t:

1. Define X as an $N \times N$ matrix with elements

$$X_{i,j} := \frac{T_{i,j}(t)}{W_{i,j}}.$$

2. Match input i to output j such that

$$(i, j) = \arg\min X_{k,l}; \tag{3.5}$$

   delete the ith row and jth column from X.

3. Repeat the preceding step until X is empty or $X_{i,j} = \infty$ for all i, j.

It is clear that $\mathcal{A}(W)$ implements LQF scheduling with "randomized queue length" given by $X_{i,j}^{-1} = W_{i,j}/T_{i,j}$. The reason for not defining $\mathcal{A}(W)$ as simply a randomized LQF algorithm is that the above description of the algorithm allows for a distributed implementation (see the next subsection). Such an implementation is feasible due to the arg min operator in (3.5) – the LQF algorithm requires the arg max operator. The algorithm $\mathcal{A}(W)$ with an appropriately chosen weight function has the following desired characteristics [52]:

- High throughput: in the following subsections we argue that the algorithm remains stable under relatively high traffic loads.
- Starvation free: $\mathcal{A}(W)$ eventually serves all virtual output queues as long as the total backlog is not infinite and W is proportional to the queue lengths; there exists a positive probability for any of the non-empty queues to be scheduled.
- Simple implementation: aside from the source of randomness, the algorithm is straightforward to implement. Conceptually, operation of each input-output pair (virtual queue) remains the same, regardless of the size of the switch.

3.4.2 Distributed implementation

Suppose that the source of randomness is available at each input port; that is, for each time slot, unit mean exponential random variables $[T_{i,j}]$ are available at the input

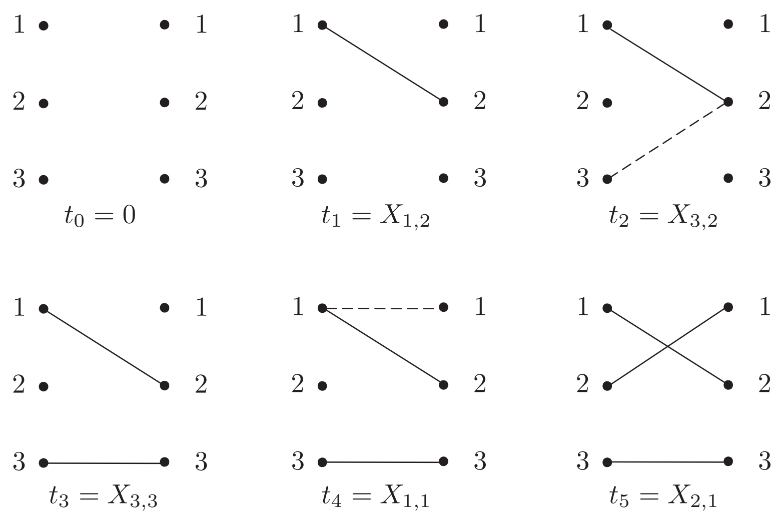


Figure 3.3: An example of how a matching is built in the case $X_{1,2} < X_{3,2} < X_{3,3} < X_{1,1} < X_{2,1} < \infty$ and all other $X_{i,j}$'s are equal to ∞. The first timer expires at $t_1 = X_{1,2}$, the second at $t_2 = X_{3,2}$ and so on. When a timer expires, the input is connected to the output only if such a connection is feasible. Non-feasible connections are shown with dashed lines.

ports. By adjusting the intensities of these random variables based on $W_{i,j}$ (e.g., based on $Q_{i,j}(t)$), one obtains $[X_{i,j}]$. At a fixed time, each input-output pair (i, j) sets a timer that expires in $X_{i,j}$ time units; note that these units do *not* correspond to time slots needed to switch a packet (cell) from an input to an output. Upon the expiration of timer (i, j), input i is matched to output j unless either of them is already matched. This process continues until all timers with $X_{i,j} < \infty$ expire. An example of how the algorithm operates for a 3×3 switch is shown in Fig. 3.3. The timers are set at time $t = 0$; as they expire, attempts are made to connect the corresponding input and output ports. Upon the completion of the scheduling phase (in this case the input ports 1, 2 and 3 are connected to the output ports 2, 1 and 3, respectively), the cell transfer phase begins.

Note that the implementation is distributed whenever $W_{i,j}$ is only a function of the state of the virtual queue (i, j), i.e., the timers are set with local parameters only. However, the complexity of implementing the required source of randomness needs to be

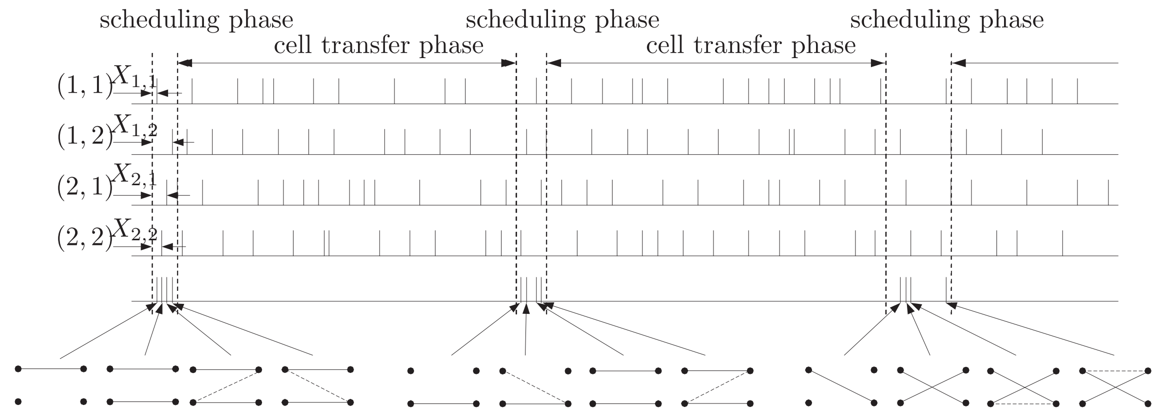


Figure 3.4: Conceptual operation of a 2×2 switch under $\mathcal{A}(W)$. Poisson processes are generated at each virtual buffer (input-output pair) with rates proportional to the occupancy continuously over time; these processes serve as the required timers during scheduling phases. At the beginning of each scheduling phase, all input-output ports are disconnected. As impulses arrive (timers expire), the corresponding feasible connections are established. The configuration of the switching fabric after each arrival of the Poisson processes is shown on the bottom of the graph. Once the fabric is configured, a cell transfer phase starts.

considered. There exists at least two possible approaches to obtain randomness: (i) external – a physical device (e.g. photon-based) auxiliary to the switching fabric can serve as a source of randomness (Poisson process); these devices can potentially operate at very high speeds; and (ii) internal – randomness can be extracted from packet payloads, e.g., see [22]. We note that lasers and photon-counting semiconductor detectors can be used to implement Poisson timers [82]. Intensities of external Poisson processes (sequences of impulses) at virtual buffers are modulated according to buffer occupancies. By the memoryless property of Poisson processes, an exponentially-distributed time interval can be defined by a single impulse, namely, the ending impulse. Then, the arg min of a set of variables is determined by the first impulse from the corresponding set of Poisson processes. Thus, the algorithm can be implemented through the two operations described in Sect. 3.3.1. Moreover, in order to simplify the source of randomness, such impulses are generated continuously as shown in Fig. 3.4. The switch operates in two alternating phases: scheduling and cell transfer; the length of the cell transfer phase is relatively longer. Note that in practice, each scheduling phase is of fixed length, and there is a positive probability that the some impulses do not arrive before the scheduling phase ends. We adjust the arrival rate to be large compared to the length of the scheduling phase, hence approximating algorithm $\mathcal{A}$. In Fig. 3.4 we illustrated the conceptual operation of the 2×2 switch. At the beginning of a scheduling phase, a sequence of impulses is generated at each virtual buffer. As impulses are generated (equivalently timers expire) from the Poisson processes, the switching fabric is configured accordingly. The resulting configurations are shown on the bottom of the figure. Once the configuration is determined, the cell transfer phase starts. Upon completion of a cell transfer phase, the switching fabric is reset, i.e., all input and output ports are disconnected, and the scheduling phase for the next time slot begins.

Finally, we point out that one can potentially implement the LQF algorithm by setting $T_{i,j}$'s in the definition of $\mathcal{A}(W)$ to constants rather than of unit-rate random variables. However, in that case one will need to implement timers that allow for synchronization of initial impulses. In addition, producing time intervals of precise lengths poses technical challenges.

3.4.3 Performance

The distributed nature of the proposed architecture and the scheduling algorithm results in a low complexity of the control plane. In this subsection, we evaluate the performance of $\mathcal{A}$. Inefficiencies of $\mathcal{A}$ are due to the fact that inverse summation is performed only approximately (see Sect. 3.3.1), and, thus, the maximum weight matching is evaluated approximately as well. First, we consider the uniform traffic pattern defined by arrival rates $\lambda_{i,j} = \rho/N$, $1 \leq i, j \leq N$, where $\rho \in [0, 1)$ is the relative traffic load. A preliminary version of the following result appeared in [55].

Theorem 3.1. Algorithm $\mathcal{A}(Q^\beta)$, $\beta > 0$, is rate stable under any admissible traffic in a 2×2 switch. Moreover, $\mathcal{A}(Q^\beta)$, $\beta > 0$, is rate stable under uniform admissible input traffic in a switch of an arbitrary size.

Proof. See Appendix B.1. $\square$

Next, we considered the diagonal traffic pattern parameterized by $\alpha \in [0,1]$, and defined by

$$\lambda_{i,j} = \begin{cases} \alpha\rho, & j = i, \\ (1-\alpha)\rho, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise}; \end{cases} \tag{3.6}$$

parameter $\rho \in (0,1)$ is the relative load for each input and output. Under this traffic pattern, the queue occupancy process is unstable under $\mathcal{A}(Q^\beta)$ when $\rho > \rho^*$, where ρ^* denotes the value of the critical relative load. The inefficiency arises due to the distributed nature of the algorithm. Note that $\rho^* < 1$ when empty queues are served with positive probabilities in the long run. We illustrate this point with the following example. Consider a 3×3 switch under diagonal traffic with $\alpha = 1/2$ and $\rho = 1$. In the long run, the switch should be configured into one of the two following service matrices:

$$m_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } m_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{3.7}$$

with probability 1. Any other configuration inevitably "wastes" one unit of service. For example, if connection $(1,1)$ is made, it must follow that queues $(2,2)$ and $(3,3)$ will also be served in the same time slot. However, since $\mathcal{A}(Q^\beta)$ is distributed, the configuration for the remaining input/output ports (2 and 3) are not constrained by the established connection that corresponds to $(1,1)$. Hence, when all queues with positive arrival rates are large, the scheduling algorithm selects a service matrix other than m_1 and m_2, such as

$$m_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \tag{3.8}$$

with a positive probability, where only two units of service occur. Note that when the switching fabric is configured according to the above service matrix, no other input-output pair with positive arrival rate can be scheduled concurrently. Although the input-output pair $(3, 2)$ can be scheduled in addition to the scheduled pair $(1, 1)$ and $(2, 3)$, it has zero arrival rate and hence corresponds to an empty queue for the given traffic pattern – service of queue $(3, 2)$ does not contribute to an actual departure of a packet.

The following proposition establishes a performance limit of $\mathcal{A}(Q^\beta)$ under the diagonal traffic pattern.

Proposition 3.1. Consider algorithm $\mathcal{A}(Q^\beta)$, $\beta > 0$, under diagonal traffic with $\alpha = 1/2$. The algorithm is not rate stable for

$$\rho > 1 - \left(\sum_{r=0}^{2N-1} \frac{(-2)^r}{r!} + \frac{1}{2} \frac{(-2)^{2N}}{(2N)!} \right) \to 1 - e^{-2}, \tag{3.9}$$

as the number of input/output ports $N \to \infty$.

Proof. See Appendix B.2. □

Results of numerical comparisons of $\mathcal{A}(Q)$ and several other centralized algorithms: MWM, LQF, and APSARA [29] under uniform and diagonal traffic are shown in Fig. 3.5 and Fig. 3.6, respectively. The MWM scheduling is discussed in Section 3.2.2 in details. The other two algorithms in comparisons are considered in the literature and briefly described below.

LQF scheduling chooses queues to serve in a slot iteratively, starting from the queue with the largest weight (or the longest queue, as the name indicates) and proceeding in

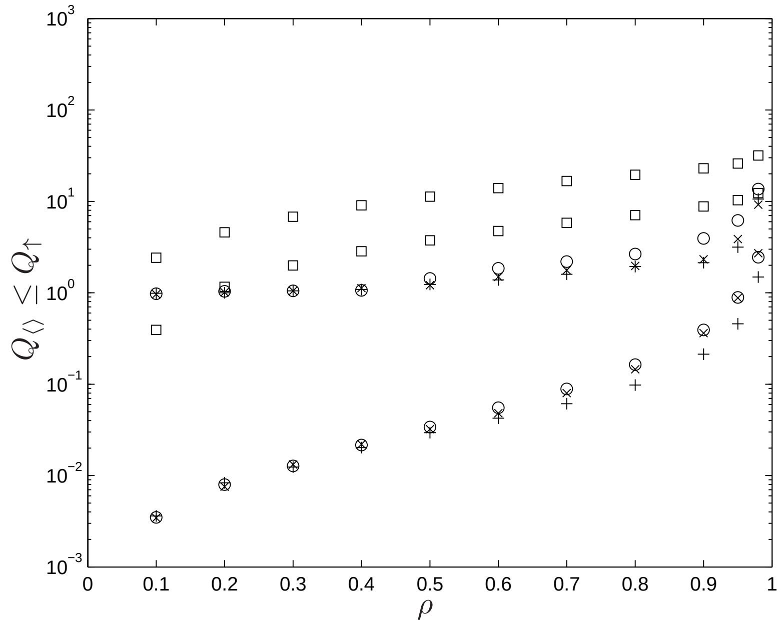
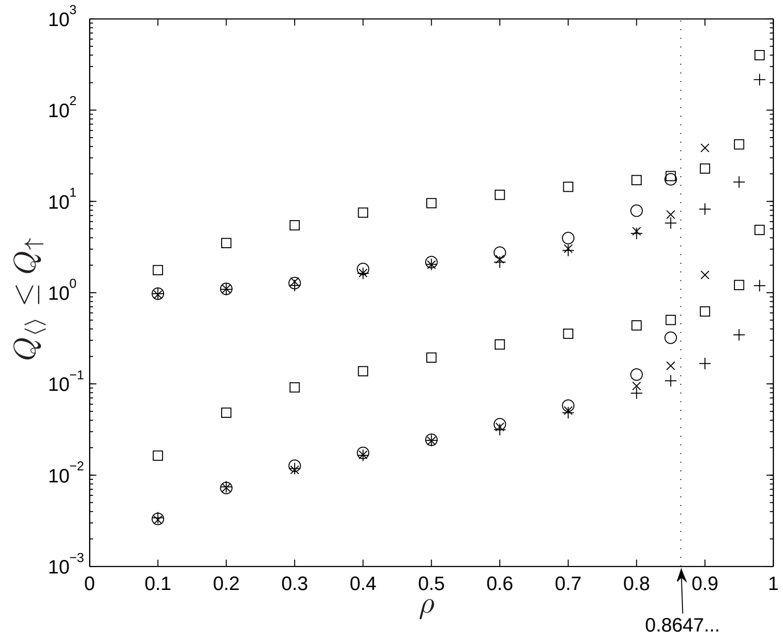


Figure 3.5: For a switch with $N = 32$, the expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for the MWM (+), APSARA (□), LQF (×), and $\mathcal{A}(Q)$ (○) algorithms under uniform traffic.

a decreasing manner. Queues that have one of their neighbors in the interference graph selected are not considered in the next iteration step. When two or more queues have equal backlog, a tie-breaking rule must be specified. This procedure is stopped until no further (nonempty) queue can be included. At each time slot, the served queues form a maximal independent set of the subgraph consisting of the nonempty queues. It is shown that the stability of LQF depends not only on the average intensities of the arrival processes in addition to their average intensities. A set of sufficient conditions for stability can be found in [23].

APSARA [29] is a heuristic algorithm based on MWM with a smaller complexity. In each time slot, it selects between the matching served in previous time slot and a matching from a Hamiltonian walk on the graph of all matchings, i.e., a matching randomly drawn from all possible matchings. APSARA utilizes the fact that the MWM matchings of contiguous time slots are correlated, since individual queue lengths can differ by one at most and, thus, the weight of a matching changes by a bounded amount.



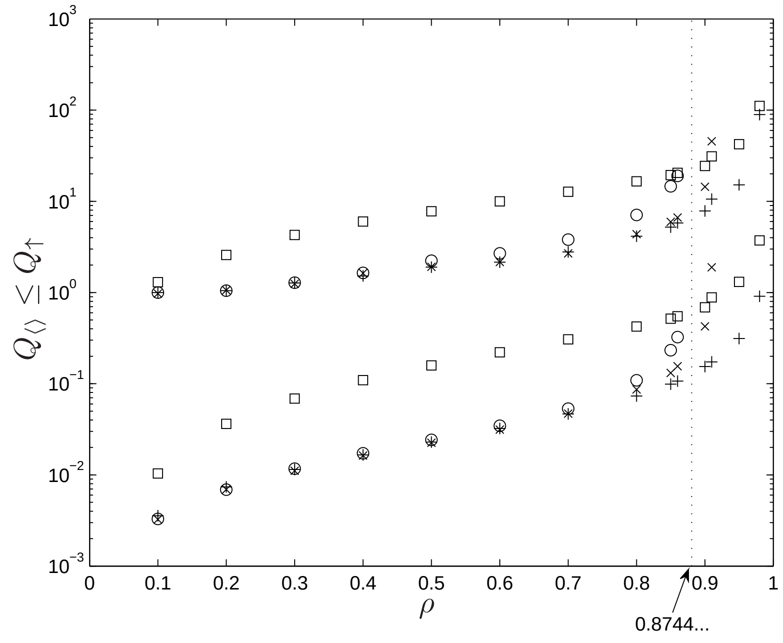


Figure 3.6: For a switch with $N = 32$, the expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for the MWM (+), APSARA ($\Box$), LQF ($\times$), and $\mathcal{A}(Q)$ ($\circ$) algorithms under diagonal traffic with $\alpha = 1/2$ (top) and $\alpha = 2/3$ (bottom). The corresponding maximum ρ's for $\mathcal{A}(Q)$ are also shown.

Namely, a heavy matching is likely to stay heavy over several time slots. APSARA is shown to be rate stable under any admissible i.i.d. Bernoulli packet arrival processes.

In our simulation, the arrivals form independent Bernoulli processes, and the number of input/output ports is $N = 32$. For a range of values of ρ, two quantities,

$$Q_{\langle\rangle} = \frac{1}{N^2} \sum_{i,j=1}^{N} \mathbb{E} Q_{i,j}(t) \tag{3.10}$$

and

$$Q_{\uparrow} = \mathbb{E}\left[\max_{i,j} Q_{i,j}(t)\right],$$

are estimated under the four algorithms. It is clear that $Q_{\langle\rangle} \le Q_{\uparrow}$, and, thus, we do not label data points in the figures. Note that for the diagonal traffic pattern one can alternately normalize $Q_{\langle\rangle}$ by $2N$ instead of N^2 as in (3.10), since only $2N$ virtual queues receive packets. Given that we consider only relative performance of algorithms, the normalization does not effect our comparison.

As seen in Fig. 3.5, the algorithm $\mathcal{A}(Q)$ performs competitively under uniform traffic for a wide range of ρ, when compared to *centralized* switch scheduling algorithms. Numerical results for diagonal traffic with $\alpha = 1/2$ and $\alpha = 2/3$ are shown in Fig. 3.6. As observed, the algorithm fails to stabilize the switch for high values of relative load ρ as indicated in Proposition 3.1. The LQF algorithm also fails to stabilize the switch under diagonal traffic as argued in [23] (the sufficient local pooling condition is not satisfied in this case). For $\alpha = 1/2$ an estimate of the critical ρ^* is given in Proposition 3.1. An upper bound of the critical relative load for $\alpha = 2/3$ can be evaluated numerically with as described in Appendix B.3. In particular, $\rho^* < 0.8744\ldots$ for $\alpha = 2/3$ and $\mathcal{A}(Q)$. The following section describes a scheduling algorithm that addresses the instability.

3.5 Reversible algorithm

Algorithm $\mathcal{A}$ makes irreversible scheduling decisions; once a connection between two ports is established, these ports remain connected until the next scheduling phase, regardless of consequent scheduling decisions. In this section, we demonstrate that enabling reversibility in decision making potentially improves the scheduling performance.

A scheduling algorithm $\mathcal{B}_\gamma(W)$, where W is the set of weights, operates during a fixed length (τ time units) scheduling phase. Similar to algorithm $\mathcal{A}(W)$, $\mathcal{B}_\gamma(W)$ attempts to connect eligible input-output pairs upon arrivals in the corresponding Poisson processes. However, all connected pairs are subject to disconnections based on arrivals in rate-μ Poisson processes. In particular, for an unmatched pair (i,j), a connection is attempted according to a Poisson process with rate $\lambda W_{i,j}$; a matched pair (k,l) is disconnected after an exponentially distributed time interval with rate μ. When $\mu = 0$ and $\lambda \to \infty$, the algorithm reduces to $\mathcal{A}(W)$ (since τ is finite). The parameter $\gamma = \lambda/\mu$ controls the relative rate of connecting/disconnecting ports. The state of the switching fabric is fixed at the end of the scheduling phase for the duration of the cell transfer phase. The operation of $\mathcal{B}_\gamma$ is formally described below. Here, for $t \in [0,\tau]$, $[T_{i,j}(t)]$ is an $N \times N$ matrix with elements being i.i.d. unit-rate exponentially distributed random variables; the matrices are independent for different values of t. Let $\mathcal{X}$ be the set of all feasible fabric configurations:

$$\mathcal{X} = \left\{ \mathcal{I} \subseteq [1,N]^2 : \sum_{i=1}^{N} 1_{\{(i,j)\in\mathcal{I}\}} \leq 1,\ \sum_{j=1}^{N} 1_{\{(i,j)\in\mathcal{I}\}} \leq 1 \right\}.$$

Without loss of generality, assume that the scheduling phase begins at $t = 0$.

Algorithm $\mathcal{B}_\gamma(W)$

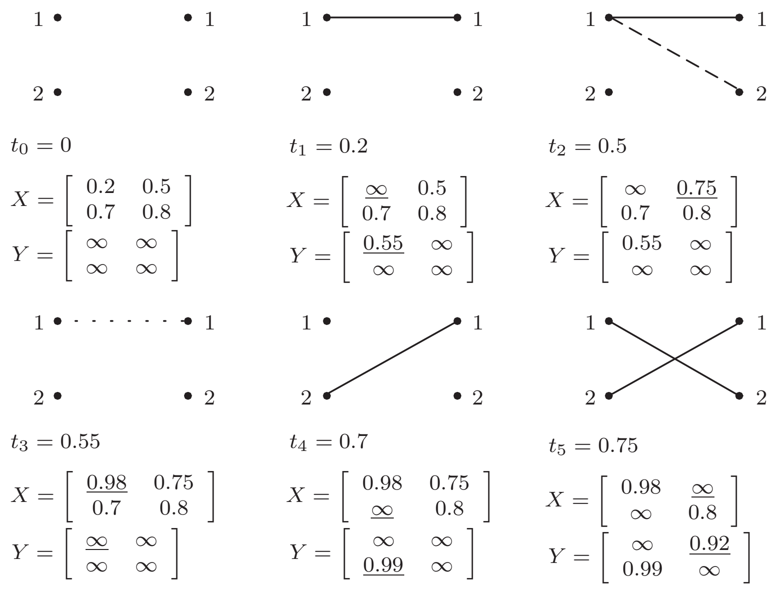
```
1. Initialization:
       $X_{i,j} := T_{i,j}(0)/(\lambda W_{i,j}),\ 1 \le i,j \le N,$    (3.11)
       $Y_{i,j} := \infty,\ 1 \le i,j \le N,$
       $\mathcal{S} := \emptyset.$

2. While $t \le \tau$:
       $(i,j) := \arg\min X_{n,m},$
       $(k,l) := \arg\min Y_{n,m}.$

   Case 1: $X_{i,j} < Y_{k,l}$.
       If $\mathcal{S} \cup \{(i,j)\} \in \mathcal{X}$,
       then $X_{i,j} := \infty$ and $Y_{i,j} := T_{i,j}(t)/\mu + t$,
       otherwise $X_{i,j} := T_{i,j}(t)/(\lambda W_{i,j}) + t$.
   Case 2: $X_{i,j} > Y_{k,l}$.
       Set $\mathcal{S} := \mathcal{S} \setminus \{(k,l)\}$,
       $X_{k,l} := T_{k,l}(t)/(\lambda W_{k,l}) + t$ and $Y_{k,l} := \infty$.
   Update $t$:
       $t := t + \min\{X_{i,j}, Y_{k,l}\}.$

3. Configure the switching fabric according to $\mathcal{S}$.
```

Elements of X and Y represent times when the corresponding connections are attempted and disconnections are preformed, respectively. An example of how the algorithm evolves over time in a 2×2 switch is shown in Fig. 3.7. Given the time-encoding computation model, $\mathcal{B}_\gamma$ can be implemented in a distributed fashion. Each reconfigura-



$t_0 = 0$

$$X = \begin{bmatrix} 0.2 & 0.5 \\ 0.7 & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} \infty & \infty \\ \infty & \infty \end{bmatrix}$$

$t_1 = 0.2$

$$X = \begin{bmatrix} \underline{\infty} & 0.5 \\ 0.7 & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} \underline{0.55} & \infty \\ \infty & \infty \end{bmatrix}$$

$t_2 = 0.5$

$$X = \begin{bmatrix} \infty & \underline{0.75} \\ 0.7 & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} 0.55 & \infty \\ \infty & \infty \end{bmatrix}$$

$t_3 = 0.55$

$$X = \begin{bmatrix} \underline{0.98} & 0.75 \\ 0.7 & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} \underline{\infty} & \infty \\ \infty & \infty \end{bmatrix}$$

$t_4 = 0.7$

$$X = \begin{bmatrix} 0.98 & 0.75 \\ \underline{\infty} & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} \infty & \infty \\ \underline{0.99} & \infty \end{bmatrix}$$

$t_5 = 0.75$

$$X = \begin{bmatrix} 0.98 & \underline{\infty} \\ \infty & 0.8 \end{bmatrix} \quad Y = \begin{bmatrix} \infty & \underline{0.92} \\ 0.99 & \infty \end{bmatrix}$$

Figure 3.7: An example of how the state of the switching fabric evolves over time. The minimum elements in both X and Y at the time instances when changes of state occur are $X_{1,1}$, $X_{1,2}$, $Y_{1,1}$, $X_{2,1}$, $X_{2,2}$. At these time instances, the algorithm changes the matching correspondingly and updates X and Y. The values of X and Y after updates are shown under the corresponding bipartite graphs; the updated elements are underlined. A dashed edge indicates that the connection fails to be established due to a conflict with the current state; a dotted edge indicates that an existing connection is eliminated.

tion of the switching fabric, which either establishes or removes a connection, involves only a single input-output pair – no inter-port communication is required. Therefore, $\mathcal{B}_\gamma$ can be realized by a similar scheme as the one described in Sect. 3.4.2, with the additional ability to disconnect two matched ports.

Since the elements of X and Y are exponential random variables, the state of the switching fabric can be described by the continuous-time Markov chain $\{\mathcal{S}(t), 0 \leq t \leq \tau\}$; a state of the switching fabric $\mathcal{I}$ is uniquely defined by the set of edges (matched input-output pairs) in the corresponding bipartite graph. Edges in $\mathcal{I}$, $\mathcal{I} \in \mathcal{X}$, are non-conflicting, i.e., they do not share an input or output node. The state $\emptyset$ denotes the empty configuration. For every $\mathcal{I} \in \mathcal{X}$, define the following two sets:

$$\begin{aligned} K_{\mathcal{I}}^{+} &= \{\mathcal{J} \in \mathcal{X} : \mathcal{J} = \mathcal{I} \cup \{(k,l)\} \text{ for some } (k,l)\}, \\ K_{\mathcal{I}}^{-} &= \{\mathcal{J} \in \mathcal{X} : \mathcal{I} = \mathcal{J} \cup \{(k,l)\} \text{ for some } (k,l)\}. \end{aligned}$$

In short, $K_{\mathcal{I}}^{+}$ is the set of states that $\mathcal{I}$ can augment into with an extra connection, and $K_{\mathcal{I}}^{-}$ is the set of states that $\mathcal{I}$ can reduce into with one removal of a connection. For example, in a 3×3 switch, for $\mathcal{I} = \{(1,1),(2,2)\}$, we have $K_{\mathcal{I}}^{+} = \{\{(1,1),(2,2),(3,3)\}\}$ and $K_{\mathcal{I}}^{-} = \{\{(1,1)\},\{(2,2)\}\}$. The process $\{\mathcal{S}(t), 0 \leq t \leq \tau\}$ is described by the initial state $\mathcal{S}(0) = \emptyset$ and the rate matrix $[p_{\mathcal{I},\mathcal{J}}]$ with elements

$$p_{\mathcal{I},\mathcal{J}} = \begin{cases} \lambda W_{i,j}, & \mathcal{J} \in K_{\mathcal{I}}^{+} \text{ and } (i,j) \in \mathcal{J} \setminus \mathcal{I}, \\ \mu, & \mathcal{J} \in K_{\mathcal{I}}^{-}, \\ 0, & \text{otherwise.} \end{cases}$$

The state of the switching fabric at the end of a scheduling phase $\mathcal{S}(\tau)$ is fixed for the duration of the cell transfer phase that follows immediately. As λ, $\mu \to \infty$, the distribution of $\mathcal{S}(\tau)$ converges to the stationary distribution π of the process with rate matrix given by $[p_{\mathcal{I},\mathcal{J}}]$. Let $n_{\mathcal{I}}(W)$ denote the number of edges in $\mathcal{I}$ that have a nonzero weight, i.e.,

$$n_{\mathcal{I}}(W) = \sum_{(i,j)\in\mathcal{I}} 1_{\{W_{i,j}>0\}}.$$

Then, the stationary distribution π is given by

$$\pi_{\mathcal{I}} = \pi_{\emptyset}\, \gamma^{n_{\mathcal{I}}(W)} \prod_{(i,j)\in\mathcal{I}} W_{i,j},$$

where

$$\pi_{\emptyset}^{-1} = \sum_{\mathcal{I}\in\mathcal{X}} \gamma^{n_{\mathcal{I}}(W)} \prod_{(i,j)\in\mathcal{I}} W_{i,j}.$$

Indeed, it is straightforward to verify that π is a solution to the set of balance equations, and it satisfies $\sum_{\mathcal{I}\in\mathcal{X}} \pi_{\mathcal{I}} = 1$.

Observe that as $\gamma \to \infty$ and $\mu \to \infty$, algorithm $\mathcal{B}_\gamma$ selects only matchings with the highest size. That is, only matchings from the set $\mathcal{X}^* = \{\mathcal{I} : n_{\mathcal{I}}(W) = \max_{\mathcal{J}\in\mathcal{X}} n_{\mathcal{J}}(W)\}$ are selected with positive probability. In particular, for $\mathcal{I} \in \mathcal{X}^*$,

$$\mathbb{P}[\mathcal{S}(\tau) = \mathcal{I}] \to \frac{\prod_{(i,j)\in\mathcal{I}} W_{i,j}}{\sum_{\mathcal{J}\in\mathcal{X}^*} \prod_{(i,j)\in\mathcal{J}} W_{i,j}},$$

and $\sum_{\mathcal{I}\in\mathcal{X}^*} \pi_{\mathcal{I}} \to 1$, as both $\gamma \to \infty$ and $\mu \to \infty$. Hence, algorithm $\mathcal{B}_\gamma$ implements a weighted version of the MSM algorithm in the limit as $\gamma \to \infty$ and $\mu \to \infty$. For example, when $W_{i,j} = 1_{\{Q_{i,j}>0\}}$, algorithm $\mathcal{B}_\gamma(W)$ selects uniformly among all maximum size matchings. On the other hand, when $W_{i,j}(t) = \exp\{Q_{i,j}(t)\}$ and γ, $\mu \to \infty$, the algorithm configures the switching fabric to state $\mathcal{I}$ with probability

$$\pi_{\mathcal{I}} = \frac{\exp\left\{\sum_{(i,j)\in\mathcal{I}} Q_{i,j}(t)\right\}}{\sum_{\mathcal{J}\in\mathcal{X}} \exp\left\{\sum_{(i,j)\in\mathcal{J}} Q_{i,j}(t)\right\}},$$

i.e., the probability of a matching being scheduled increases exponentially with respect to the number of packets in the matching. In [76] the authors conjecture that a delay-optimal algorithm is the one that selects a max-weight matching among the maximum size matchings (the weight is a logarithmic function). Therefore, by setting $W_{i,j} = Q_{i,j}$, algorithm $\mathcal{B}_\gamma$ approximates (in the limit) this algorithm in [76] since it selects a matching based on $\prod_{(i,j)\in\mathcal{I}} Q_{i,j}$ and $\log \prod_{(i,j)\in\mathcal{I}} Q_{i,j} = \sum_{(i,j)\in\mathcal{I}} \log Q_{i,j}$. Finally, we point out that the structure of stationary distribution π is identical to the stationary distribution of a chain that arises in the analysis of a CSMA scheduling algorithm proposed in [39].

Next we provide some intuition on why reversibility improves switch stability. To this end, consider the example discussed in Sect. 3.4.3 of a 3×3 switch under diagonal traffic with $\alpha = 1/2$. Given that all queues with positive arrival rates are equal and large, algorithm $\mathcal{A}$ selects a matching that serves 2 rather than 3 queues (such as m_3, see (3.8)) with probability (approximately) 1/3, i.e., with probability 1/3 one unit of service is “wasted”. This effect results in $\rho^* < 1$ (see Proposition 3.1). On the other hand, when $\lambda \gg \mu \gg 1/\tau$, algorithm $\mathcal{B}_\gamma$ can configure the switching fabric to a state corresponding to m_3 only at the beginning of the scheduling phase. Given that the switch is in this state, one of the connected pairs will be disconnected (since $\mu \gg 1/\tau$), and, due to $\lambda \gg \mu$, a feasible pair of ports will be connected. This process continues until 3 pairs are connected, say m_1 or m_2 (see (3.7)). Once the fabric is in one of these states, it is not configured to a state corresponding to m_3 due to the fact that as soon as one of the pairs is disconnected, the same pair is connected right away ($\lambda \gg \mu$) because it remains the only feasible pair. That is, the limiting algorithm $\mathcal{B}$ selects only among maximum size matchings.

A numerical comparison of $\mathcal{B}_\gamma(Q)$ with parameters $(\lambda, \mu, \tau) = (1, 10, 1)$, $(1, 10, 10)$ and an algorithm approximating $\mathcal{A}(Q)$ ($\mathcal{B}_\gamma(Q)$ with $(\lambda, \mu, \tau) = (1, 0, 1)$) under diagonal traffic is shown in Fig. 3.8. We point out that while $\mathcal{B}_\gamma(Q)$ for finite values of λ, μ, τ can provide a larger admissible region than $\mathcal{A}(Q)$, the corresponding queues sizes under

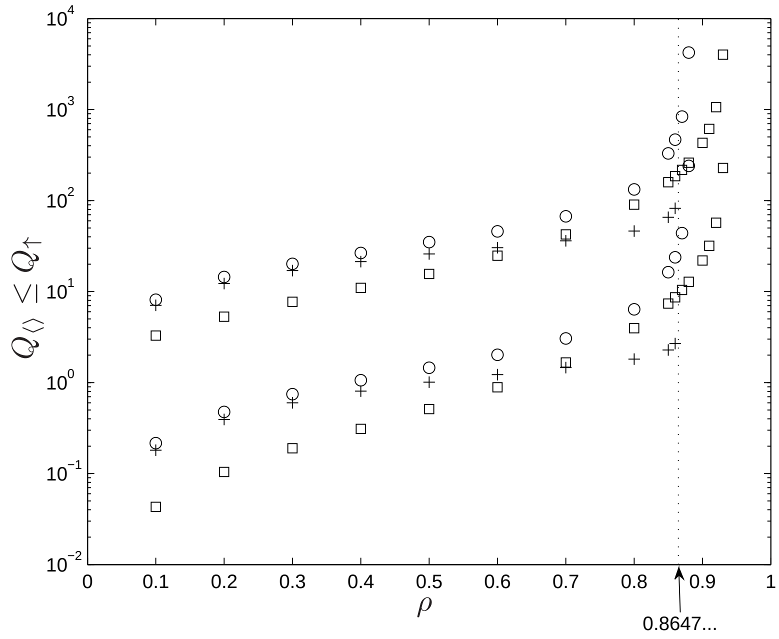


Figure 3.8: The expected average virtual output queue length $Q_{\langle\rangle}$ and expected maximum virtual output queue length $Q_{\uparrow}$ for $\mathcal{B}_\gamma(Q)$ (+,□) and an approximation of $\mathcal{A}(Q)$ (∘) under diagonal traffic with $\alpha = 1/2$ ($N = 32$). Here $\mathcal{A}(Q)$ is approximated by $\mathcal{B}_\gamma(Q)$ with $(\lambda, \mu, \tau) = (1, 0, 1)$, and $\mathcal{B}_\gamma(Q)$ uses parameters $(\lambda, \mu, \tau) = (1, 10, 1)(+)$ and $(1, 10, 10)(\square)$. The upper bound of the critical ρ for $\mathcal{A}(Q)$ is also shown; note that $\mathcal{B}_\gamma(Q)$ stabilizes the switch under diagonal traffic with values of ρ that exceed this bound.

$\mathcal{B}_\gamma(Q)$ can be larger than under $\mathcal{A}(Q)$ for moderate values of ρ. This is due to the fact that $\mathcal{B}_\gamma(Q)$ is not efficient when $\lambda \gg \mu$ does not hold, since some feasible input-output pairs might not be connected at the end of the scheduling phase. However, given that $\mathcal{B}_\gamma(Q)$ is determined by the product $\lambda Q_{i,j}$ (see (3.11)), increasing queue lengths uniformly and multiplicatively is equivalent to increasing λ. Thus, when queue lengths increase, the scheduling performance improves.

Finally, note that the performance of $\mathcal{B}_\gamma(W)$ can be improved by running the algorithm in background during the cell transfer phase, i.e., effectively increasing the value of τ and, thus, allowing for better convergence to the stationary distribution π. Recomputing the schedule for each time slot is equivalent to independently sampling from π in each time slot in the limit as $\gamma, \mu \to \infty$. On the other hand, the algorithm in [39] produces schedules that are strongly correlated in time, and, thus, different behavior. For example, under diagonal traffic with $\alpha = 1/2$ and $Q_{i,j} = q1_{\{\lambda_{i,j}>0\}}$ for large q, $\mathcal{B}_\gamma(W)$ serves each matching with probability 1/2; on the other hand, if the algorithm is not reset (all ports disconnected) for each time slot, then a matching is selected with probability 1/2 and served in the following time slots until its weight becomes small relative to the weight of the other matching.

3.6 Concluding remarks

In this chapter we introduce a time-encoding model of computation, which enables a set of new architectures and algorithms for packet switches. We developed distributed algorithms for single cross-bar packet switch that achieves performance comparable to that of centralized algorithms. The proposed scheduling algorithms can be applied to scheduling in wireless networks modeled by general interference graphs; in an interference graph, two nodes can not transmit concurrently if they share an edge, see, e.g., [14, 61, 56]. The crossbar switch has a well-structured interference graph (the dual graph of a full bipartite graph) with each node representing an input-output pair. In the case of wireless networks, nodes in the graph represent network nodes and edges typically indicate physical proximity of network noes. Algorithm $\mathcal{A}$ operates on a general graph as follows. Each network node sets possibly multiple timers according to the corresponding weights. A connection is established if the timer expires before any of the interfering pair is scheduled for transmission. The resulting schedule is an approximation of the maximum weight matching on the general graph. Algorithm $\mathcal{B}$ can be applied to general interference graphs analogously by enabling reversibility of scheduling decisions.

CHAPTER 4

Bandwidth Sharing

4.1 Introduction

The performance of a multi-hop network depends on the bandwidth sharing among individual flows. Various bandwidth sharing policies have been proposed to achieve throughput maximization and/or user fairness [57, 58]. It was shown [11, 81] that fair bandwidth sharing stabilizes the network whenever possible, while the bias introduced by discriminatory scheduling policies may lead to loss of capacity. These studies are mostly conducted under the fluid assumption of network flows, i.e., the traffic is treated as continuous medium and the packet-level granularity is ignored. In practice, the information on flow rates, oftentimes critical to bandwidth sharing mechanisms implemented distributively at nodes (either intermediate network nodes or end users), is generally not explicitly available. Instead, the observation of the network traffic at a node is limited to packet-level events, e.g. arrivals, departures and packet drops due to lack of buffer space. While it might be reasonable to use the number of packets as a proxy for the flow rates at intermediate nodes when the number of packets per flow in a buffer is large, such an approximation can be erroneous when the amount of memory at intermediate nodes is limited. For example, suppose that two flows with different rates utilize an intermediate node. With a large buffer space, the node may buffer 100 packets and 10 packets from the first and the second flows, respectively, and thus, the node can differentiate flows based on their rates. However, when the node has very limited buffer space, it is likely that it holds only one packet from each flow, and, hence the node's ability to make effective differentiation solely based on the packet count is compromised. The invalidity of fluid assumption in this case is due to the discrete nature of packets. Therefore, the analysis of the network on packet-level bandwidth sharing and its implementation is of interest. A number of studies that concern bandwidth allocation in terms of packet-based implementation adopt the framework of the TCP connection game [5, 84]. However, these results are commonly restricted to the assumption of network model with a single bottleneck link. On the other hand, a number of studies are devoted to congestion control. A hop-by-hop congestion control algorithm is proposed in [83]. In [9, 5, 28], the necessity of presence and level of congestion control is investigated. In [73], the authors propose the shortest remaining processing time preemptive resume scheduling algorithm. A duality between congestion control and queue management is discovered by [51].

Our approach to achieve the desired bandwidth sharing is based on employing buffer management policies in intermediate nodes that utilize additional attributes of the packets. In particular, we focus on improving the network throughput. The buffer management mechanisms that implement a bandwidth allocation consists of two aspects: service discipline and packet dropping. That is, given a collection of packets in the buffer, a node needs to select the next packet for transmission; when the buffer is full, the node chooses a packet to drop upon a packet arrival.

The rest of this chapter is organized as follows. In Section 4.2, we describe the cyclic network model and three buffer management policies. The throughput performance of each policy is analyzed in Section 4.3 under two service time distributions: deterministic and exponential. Section 4.4 contains numerical simulations of these algorithms. Finally, concluding remarks can be found in Section 4.5.

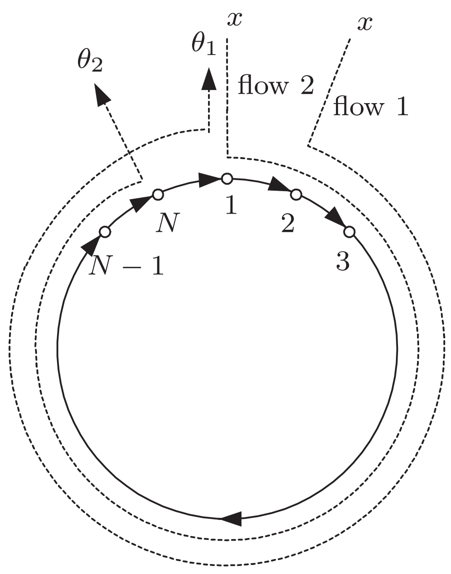


Figure 4.1: The cyclic network contains N nodes and N flows. Each flow traverses all nodes in the network sequentially. In the figure, only flow 1 and flow 2 are shown with dotted lines.

4.2 Model

4.2.1 Cyclic network

We focus on a cyclic network model with N identical nodes and N flows. A node serves packets according to the given service discipline; without loss of generality, the mean service time is taken to be unity. A flow is indexed by the first node on its route; in particular, flow i enters the network at node i, traverses all nodes in the network in the order of $(i+1) \mod N, (i+2) \mod N, \ldots, (i+N-1) \mod N$, and exits the system as shown in Fig. 4.1. Flows generate packets to be injected into the network according to i.i.d. Poisson processes with rate x. Each node is capable of storing $b-1$ packets (b is the maximum number of packets that a node is capable of holding, including possibly the one in service). When a packet arrives at a node in service, it can be enqueued if the buffer is not full; otherwise, the node discards a packet according to the packet dropping policy. We denote the throughput of flow i, i.e., the expected rate that packets complete service and exit the system, by θ_i. In particular, let $\phi_i(t)$ be the cumulative number of packets from flow i that complete service (traverse all nodes) up to time t, starting from an empty system; the throughput is defined as

$$\theta_i = \liminf_{T\to\infty} \frac{\mathbb{E}\phi_i(T)}{T},$$

for i, and the average throughput of the system is $\theta = \sum_{i=1}^{N} \theta_i/N$. Note that regardless of the arrival processes and service time distribution, the maximum throughput of the cyclic network model cannot exceed $1/N$. In fact, this is only achieved under the condition that all arrivals and service times are perfectly paced, so that utilization of each node is 1.

4.2.2 Buffer management policies

The specification of a buffer management mechanism consists of two parts: service discipline and dropping policy. The service discipline determines the priority of packets in receiving service. We assume that the service is operated in a work-conserving fashion throughput the chapter. On the other hand, the dropping policy selects the packet to be dropped upon a packet arrival at a node with full buffer. We assume that all nodes in the network are identical, i.e., they adopt the same buffer management policy. In the following, three different buffer management mechanisms are discussed.

- First-come-first-serve (FCFS)and tail dropping

  The FCFS service discipline with tail dropping policy is a simple and commonly considered buffer management policy. As the names suggest, the packets are served in the order of their arrivals to the node; under tail dropping policy, the newly arrived packet is rejected when the buffer is full. Both policies do not utilize any packet attributes besides arrival times.

- Oldest-packet-first (OPF) policy

  This policy prioritizes packets according to its age in the system; the longer (in the number of hops) a packet stays in the network, the higher its priority of receiving service and being enqueued. Under this regime, each packet maintains a counter that is incremented every time the packet completes service in a node, a piece of information similar to the time-to-live (TTL) field in the standard packet format provided by TCP/IP protocol. Whenever a service is completed, the node selects the packet with the highest priority and serves for this packet, before forwarding it to the next node (or the packet exiting the system). In addition, when the arriving packet has a strictly higher priority than the one in service, the service is withdrawn and given to the arriving packet. If a packet arrives to a node with full buffer, the packet (among the enqueued and the new arrival) with the smallest number of hops traveled is discarded. Each time the node begins serving a different packet, the service time is regenerated from the given distribution. Aside from the additional time information stored in the packets, the oldest-packet-first policy required each node to perform a sorting function to the enqueued packets, which is of complexity $O(\log b)$. However, the operations of these algorithms remain fully distributed.

- Newest-packet-first (NPF) policy

  This scheme serves as a comparison to the OPF policy. Packets with smaller ages in the system are served/enqueued) with higher priority. Similar to OPF, each packet is required to keep record of the number of hops that it has traveled in the network. The service is received by the newest packet and the oldest packet is dropped upon an arrival to a full buffer.

4.3 Throughput analysis

In this section, we consider two distributions of service times: deterministic and exponential. It is demonstrated that, unlike the FCFS with tail dropping, the OPF policy performs well regardless of the offered load, i.e., the network does not experience throughput degradation due to congestion.

4.3.1 Deterministic service time

First, we consider the case of deterministic service times and nodes with no buffer space ($b = 1$). Under these assumptions, a packet arriving to a node is either served (at least until another arrival occurs) or rejected; if another packet with strictly higher priority arrives while the packet is in service, the packet is dropped immediately. The conditional probability that a packet is dropped at the ith node on its route given that it is served by the first $(i-1)$ nodes on its route is denoted by p_i; and the conditional probability that a packet completes service at the ith node on its route, given that it is served by all previous nodes is denoted by $q_i = 1 - p_i$. Due to symmetry of the network topology, these conditional probabilities are independent of the flow index. Note that if two packets have both completed service at the same node, then their potential arrivals at all downstream nodes differ by at least by one time unit, i.e., neither of the packets will be discarded at a node because the other packet is receiving service or arriving at that node. Hence once a packet completes service at the first node, it can only be dropped due to an exogenous arrival. Clearly, the throughput is uniform across network nodes and is given by

$$\theta = x \prod_{i=1}^{N} q_i. \tag{4.1}$$

The following theorem characterizes the limiting behavior of the throughput performance under the OPF policy. This result indicates that the OPF policy is free of throughput degradation due to congestion.

Theorem 4.1. Consider the N-node cyclic network with $b = 1$. Under the OPF policy, the throughput θ_{OPF} is satisfies

$$\frac{x}{1+Nx} \geq \theta_{OPF} \geq \frac{x}{1+(2N-1)x},$$

for all $N > 0$ and any $x > 0$.

Proof. We make the observation that a packet can not be dropped once it completes service at least one node. This is because under the OPF policy, exogenous arrivals are always served/enqueued with the lowest priority, and other forwarded packets do not arrive at the same node within one time unit. Thus, given that a packet completes service at the first node, the dropping probability is zero, i.e., $p_i = 0$ and $q_i = 1$ for $i > 1$. Then, due to (4.1), one has $\theta_{OPF} = q_1 x$.

Since packets arrive to the system according to Poisson processes, we will utilize the PASTA property to evaluate p_1. Two exclusive and exhaustive scenarios that result in a packet drop at the first node are discussed individually:

1. The server of the first node busy when the packet arrives. This event occurs with probability Nq_1x, sice the server is busy Nq_1x fraction of the time.

2. A packet is forwarded from an upstream neighboring node while the newly arrived packet is in service at the first node on its route. A sufficient condition for this event to occur is that the packet arrives in one unit prior to the arrival of the forwarded packet. Thus, the probability that a new arrival will be dropped due to this reason is in $[0, (N-1)q_1x]$, since this condition is satisfied $(N-1)q_1x$ fraction of time.

Taking into account of these two cases, we obtain the relation

$$Nq_1x \leq p_1 \leq (2N-1)q_1x.$$

Solving the above inequality allow us to upper and lower bound q_1 as follows

$$\frac{1}{1+Nx} \geq q_1 \geq \frac{1}{1+(2N-1)x},$$

and, hence, the throughput can be bounded by

$$\frac{x}{1+Nx} \geq \theta_{OPF} \geq \frac{x}{1+(2N-1)x}.$$

$\square$

Due to the nature of this policy, the first node for each flow exhibits a congestion control behavior that restricts the number of packets injected into the network. Note that both bounds increases with x, suggesting that increasing the input rate does not result in congestion that diminishes the network performance. In the limiting case when $x \to \infty$, the throughput $\theta^*_{OPF} = \lim_{x\to\infty} \theta_{OPF} \in [1/(2N), 1/N)$, and thus, $\theta^*_{OPF} = \Theta(1/N)$ as $N \to \infty$. Therefore, the OPF policy achieves at least one half of the maximum throughput. Clearly, in the case $x = o(1/N)$, the throughput approaches x as $N \to \infty$.

In order to complete the comparison, we also compute the throughput under the NPF policy.

Theorem 4.2. Consider the N-node cyclic network with $b = 1$. The throughput θ_{NPF} under the NPF policy is given by

$$\theta_{NPF} = \frac{e^{-2(N-1)x}x}{1+x},$$

for all $N > 0$ and any $x > 0$. Hence, the maximum throughput under the NPF policy is achieved when the input rate is given by

$$x^*_{NPF} = \frac{1}{2}\left(\sqrt{\frac{N+1}{N-1}} - 1\right).$$

Proof. In contrast with the OPF policy, a packet arriving to the network is dropped upon arrival only if the first node on its route is serving a packet from the same flow. By the PASTA property, the drop probability satisfies $p_1 = q_1 x$, since $q_1 x$ represents the fraction of time that a node is busy serving packets from the flow that enters the network at that node. Thus, at the first node,

$$p_1 = \frac{x}{1+x}. \tag{4.2}$$

Once a packet is forwarded to the second node, it will only be dropped due to exogenous arrivals at the second node, since all other packets forwarded from the previous node do not arrive within one time unit. The same applies to all downstream node, i.e., a forwarded packet can be dropped due to exogenous arrivals. In particular, suppose that the packet arrives at the ith node at time t_i, $i > 1$, the service is completed if and only if there are no exogenous arrivals on this node between $(t_i - 1, t_i + 1)$. Since the exogenous arrival processes are mutually independent and independent of the state of the network, using the property of Poisson processes, we obtain

$$q_i = e^{-2x} \tag{4.3}$$

for $i > 1$; q_i is the probability that there are no arrivals from an exogenous arrival process in two units of time. Combining (4.3) with (4.1) and (4.2) renders

$$\theta_{NPF} = \frac{e^{-2(N-1)x}x}{1+x}. \tag{4.4}$$

The input rate x^*_{NPF} that achieves the optimal (highest) throughput can be obtained by taking the derivative of (4.4) with respect to x:

$$\frac{d\theta}{dx} = \frac{e^{-2(N-1)x}}{(1+x)^2} \left(2(N-1)x^2 + 2(N-1)x - 1\right).$$

Setting the derivative to 0 yields

$$x^*_{NPF} = \frac{1}{2}\left(\sqrt{\frac{N+1}{N-1}} - 1\right).$$

□

When network size grows to infinity, we have $\lim_{N\to\infty} N x^*_{NPF} = 1/2$, and the limiting throughput is given

$$\lim_{N\to\infty} N\theta^*_{NPF} = \frac{1}{2e},$$

i.e., the NPF policy can potentially provide a $1/(2e) \simeq 0.1839...$ fraction of the maximum possible throughput at best. However, the throughput θ^*_{NPF} is achieved under the very specific input rate x^*_{NPF}; as input rate increases beyond x^*_{NPF}, the system is subject to a performance degradation due to congestion. Moreover, this degradation is approximately exponential in x. In order to achieve the optimal throughput, the input rate at each node needs to be set to a value that is dependent of the network size, i.e., some congestion control mechanisms that utilizes global information is required.

Throughput degradation due to congestion also occurs under the FCFS with tail dropping policy. The dependency between input rate and throughput is similar to the dependency under NPF as demonstrated by numerical simulation in Section 4.4 (See Fig. 4.2).

4.3.2 Exponential service time

Next, we consider the case when the service times are i.i.d. and exponentially distributed with rate 1, and $b \geq 1$. Similar to Section 4.3.1, the arrival/service/drop dynamic, is identical for all nodes, due to the symmetry of the cyclic network. In order to gain analytical insight, we approximate the arrival process formed by packets from individual flows by independent Poisson processes. This approximation allows us to apply the PASTA property and focus our computation on the rate of the process. In fact, the processes are more regular than Poisson processes – burst of packets are likely to be eliminated when they go through nodes. It is suggested in [80] that, similar the PASTA property, periodic arrival processes also observe the system in steady state distribution in many general stochastic systems. Under the FCFS discipline with tail dropping, each node is approximated by a $M/M/1/b$ queueing system. Recall that the drop probability for the $M/M/1/b$ queueing system is π_b, where

$$\pi_i = \frac{1-\lambda}{1-\lambda^{b+1}}\lambda^i, \tag{4.5}$$

$i = 0, \ldots, b$, is the steady state distribution of the buffer occupancy and λ is the total arrival rate to the node, e.g., see [10]. The arrival process to a node consists of N components: $(N-1)$ forwarded flows of the corresponding rates $x\prod_{i=1}^{j} q_i$, $j = 1, \ldots, N-1$ and one exogenous arrival process of rate x; recall that q_i's are the conditional pass probabilities defined in Section 4.3.1. By the Poisson assumption for the arrival process, q_i is independent of the index i of the node (hence we omit the index and write $q = q_i$), and is given by

$$\begin{aligned} q &= 1 - \pi_b \\ &= \frac{1-\lambda^b}{1-\lambda^{b+1}}, \end{aligned} \tag{4.6}$$

where the second equality follows from (4.5). The total arrival rate λ can be expressed in terms of x and q as the follows,

$$\begin{aligned}\lambda &= x + qx + q^2x + \cdots + xq^{N-1} \\ &= x\frac{1-q^N}{1-q},\end{aligned} \tag{4.7}$$

and the expression for the throughput in (4.1) yields an approximation $\tilde{\theta}_{FCFS}$ for the throughput θ_{FCFS}:

$$\tilde{\theta}_{FCFS} = q^N x. \tag{4.8}$$

The set of equations (4.6), (4.7) and (4.8) fully characterizes the relationship between input rate x and throughput approximation $\tilde{\theta}_{FCFS}$.

Now, we consider the limiting case $N \to \infty$. It is sufficient to consider two cases: $\lambda = \Omega(1)$ and $\lambda = o(1)$, as $N \to \infty$. In the case when $\lambda = \Omega(1)$, since the conditional pass probability q is strictly less than 1, the throughput $\tilde{\theta}_{FCFS}$ decreases exponentially with $N \to \infty$. On the other hand, when $\lambda = o(1)$, (4.5) yields, as $N \to \infty$,

$$\pi_b = \Theta\left(\lambda^b\right),$$

and (4.7) results in, as $N \to \infty$

$$\begin{aligned}x &= \frac{1-q}{1-q^N}\lambda \\ &= \Theta\left(\frac{\lambda^{b+1}}{1-e^{-N\lambda^b}}\right).\end{aligned} \tag{4.9}$$

Substituting the (4.9) into (4.8), we obtain

$$\tilde{\theta}_{FCFS} = \Theta\left(\frac{\lambda^{b+1}e^{-N\lambda^b}}{1-e^{-N\lambda^b}}\right),$$

as $N \to \infty$, which, with a change of variable $y = n\lambda^b$, can be written as

$$\tilde{\theta}_{FCFS} = N^{-\frac{b+1}{b}}\Theta\left(g(y)\right).$$

The function $g(y)$ is defined as

$$g(y) = \frac{e^{-y}}{1-e^{-y}}y^{\frac{b+1}{b}},$$

and for all large enough y,

$$\frac{\partial g}{\partial y} = \frac{\exp(-y)y^{\frac{b+1}{b}}\left[(-1+\frac{b+1}{b}y^{-1})(1-e^{-y})-e^{-y}\right]}{(1-e^{-y})^2}$$

is negative, i.e., $g(y)$ is maximized with a finite y. Thus the quantity $\tilde{\theta}_{FCFS}$ is maximized with $\lambda = \Theta(N^{-1/b})$, implying that

$$\tilde{\theta}^*_{FCFS} = \Theta(N^{-\frac{b+1}{b}}),$$

as $N \to \infty$. As seen in the following section, the throughput θ_{FCFS} is subject to an exponential decay in input rate x, as approximated by $\tilde{\theta}_{FCFS}$.

4.4 Numerical comparisons

This section contains numerical comparisons of the throughput under the three buffer management mechanisms discussed in Section 4.2.2. The simulations are conducted with network size $N = 100$. In Fig. 4.2, we plot the throughput θ as a function

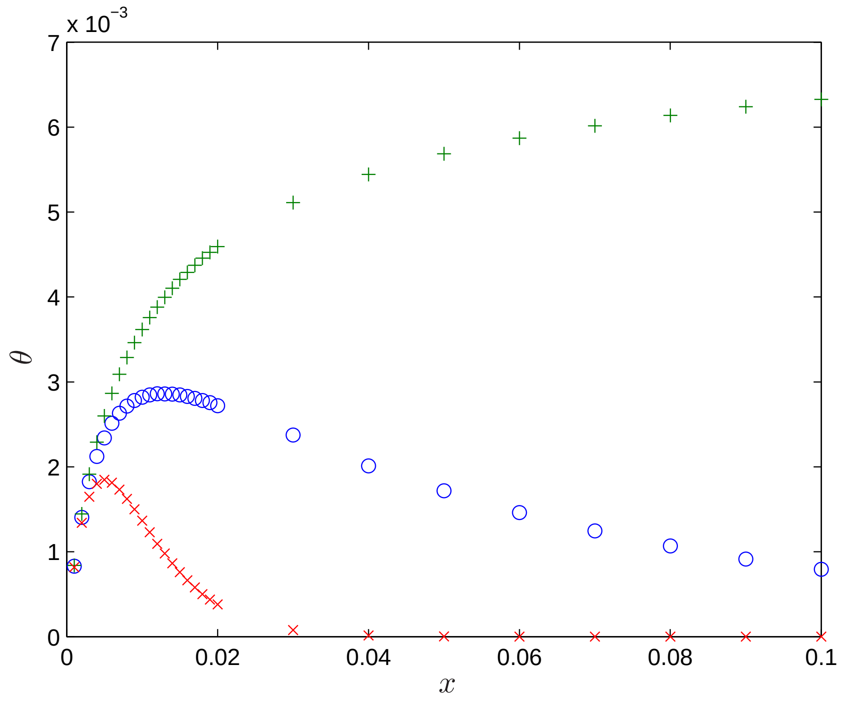
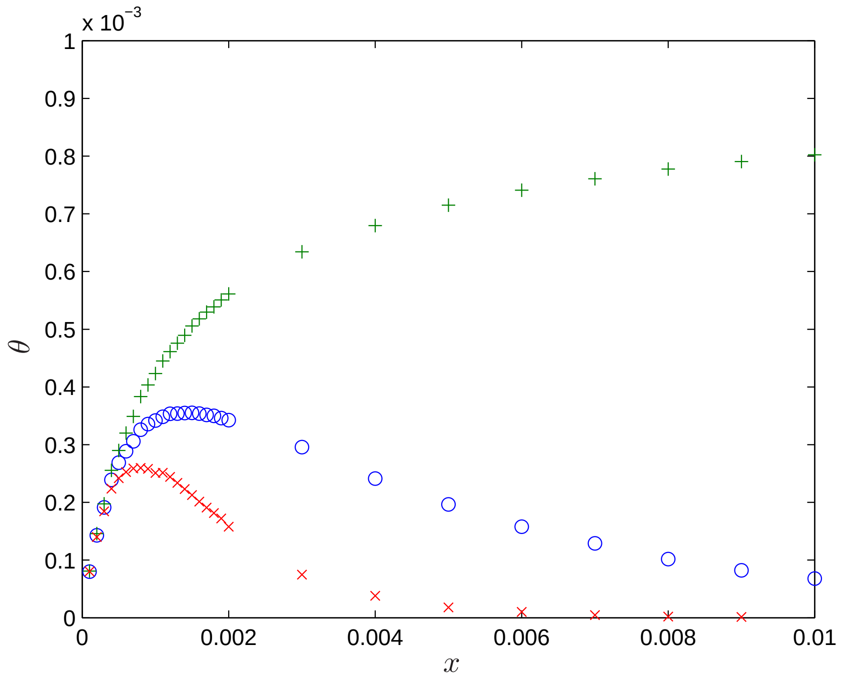


Figure 4.2: The throughput θ as a function of input rate x under the OPF policy (+), NPF policy ($\times$) and FCFS service with tail dropping ($\circ$). The cyclic network consists of $N = 100$ nodes and there are no buffer spaces ($b = 1$) for all nodes. Service times are deterministic with unit length.

of the input rate x under the OPF policy, NPF policy and FCFS discipline with tail dropping. Both the NPF policy and FCFS discipline with tail dropping suffer from performance degradation due to congestion; under high traffic load, the probability that a packet completes service decreases exponentially with the number of hops traveled. In contrast, the throughput under the OPF policy increases with input rate x.

A comparison for the system of exponentially distributed (unit rate) service times (the specification for network size and buffer space remains the same) is shown on the top of Fig. 4.3; on the bottom plot we present the simulation result for the case $b = 5$. Observe that a small increase in the buffer space significantly boosts the throughput, especially for OPF and FCFS, rather than for NPF. Nonetheless, in both cases, the throughput performance exhibits similar behavior as in Fig. 4.2 – under the the NPF policy and FCFS discipline with tail dropping it decreases exponentially for large input



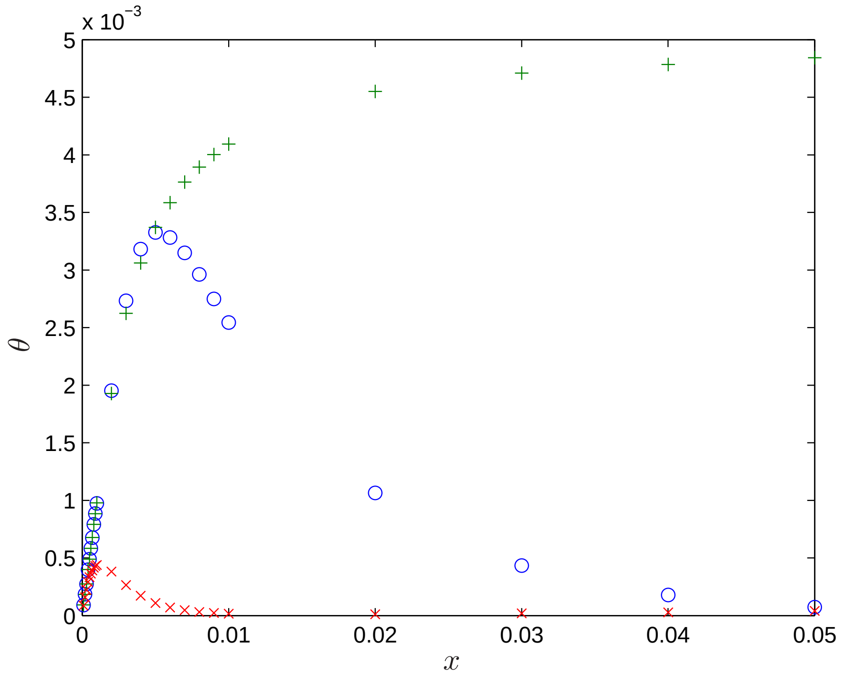


Figure 4.3: The throughput θ as a function of input rate x under the OPF policy ($+$), NPF policy ($\times$) and FCFS service with tail dropping ($\circ$). The cyclic network consists of $N = 100$ nodes and $b = 1$ (top) or $b = 5$ (bottom) for all nodes. Service times are exponentially distributed.

rate x while under the OPF policy it increases with x and approaches a constant fraction of the maximum throughput.

4.5 Concluding Remarks

This chapter examines the bandwidth sharing problem in multi-hop networks. In contrast with approaches in the literature that rely on the fluid assumption of the flows, our discussion focus on the packet-level granularity. We propose a buffer management profile, the OPF policy, that achieves a constant fraction of the throughput and the robustness under heavy traffic load, i.e., the throughput performance does not degrade with congestion as observed under the FCFS with tail dropping policy. Furthermore, our buffer management mechanism is distributed and can be easily implemented on current queueing systems without additional hardware. The implementation is identical for all network nodes and does not require global information or any time-correlated estimation (e.g. flow rate)of the system.

CHAPTER 5

Conclusions

Advances in information technology enabled the development of large-scale communication networks, giving rise to a new set of research problems on scalability studied in this dissertation. The basic idea is to analyze the impact of the network scale on its operation and use these insights to design a set of algorithms that support efficient operation of large-scale networks. In particular, we propose a set of scalable algorithms that facilitate communication effectively when the network size increases while individual network nodes are equipped with only limited resources. We start by an investigation of the size of routing tables under various routing algorithms as the number of nodes in the network increases. We term a routing protocol in a flat-label network information-efficient if the amount of information at individual nodes required to route packets does not increases with the network size. It is shown that the popular shortest-path and straight-line routing algorithms are not information-efficient, i.e., these protocols can be implemented only when nodes' memory increases with the network size. On the other hand, it is established that there exist information-efficient algorithms, e.g., column-first routing protocol, that route packets correctly even if each node in the network is capable of storing information on a fixed number of destinations only. Next, we develop a novel scheduling mechanism for distributed systems, and packet switches in particular, based on a minimal set of hardware components. The standard model of computation implemented by digital logic is replaced by a model based on time encoding. This new model allows for distributed computation with low hardware complexity. A distributed switch scheduling algorithm utilizing time encoding is shown to deliver performance comparable to centralized algorithms under uniform traffic. Exploiting a connection between switch scheduling and interval packing, we argue that the distributed nature of the algorithm limits the maximum relative load to $1 - e^{-2}$ under the worst-case scenario. We also point out that the stability of the algorithm can be improved by enabling reversibility in distributed decision making. Finally, we discuss the bandwidth sharing problem for multi-hop networks. Instead of adopting the fluid assumption of the network traffic, we examine the system at the packet-level granularity. A set of buffer management mechanisms that delivers a constant fraction of the maximum throughput is considered. In contrast to the widely-adopted FCFS service discipline with tail-dropping policy, this mechanism is robust to heavy traffic loads – the throughput does not degrade due to congestion. Furthermore, the buffer management is minimal, since it utilizes only simple packet attributes and requires no estimation of flow rate. The implementation is identical for all network nodes and does not require global information of the network.

The reflections on our research is threefold. First, the results in this dissertation indicate that improving the scalability of algorithms is a multidimensional problem. Network scalability is a relatively new metric of assessing the system performance; analytical studies on this topic have received considerable attention only recently. The results are far from complete and a number of issues remain to be resolved. In this dissertation, we examine the network scalability from various aspects that have been overlooked in the literature. In particular, while theoretical studies on the impact of finite memory focused on the data plane (buffer space), our discussion in Chapter 2 direct the attention to the control plane, i.e., the routing information. In Chapter 3, it is suggested that an entirely different model of computation can be effectively used to perform distributed scheduling, i.e., implicit information exchange can be utilized to replace the traditional message-passing approaches.

Second, our research provides evidences that a set of completely new protocols might be required for the operation of large-scale communication networks – some of the protocols employed by the current systems either suffer from inefficiencies that exacerbate with the network scale, or require an impractically large amount of computing resources. This is best demonstrated by drawing a contrast between traditional algorithms and our proposed algorithms that aim to improve scalability. For example, the information-efficient/inefficient routing protocols in Chapter 2, and the buffer managements with/without packet attributes in Chapter 4. The mathematical insights and proposed algorithms can serve as first steps toward the evaluation of other network scalability problems. A uniform theoretical framework remains to be established.

Third, a tradeoff between the distributiveness and the performance of algorithms is exhibited. The design of scalable systems is recognized to be a challenging problem. While the algorithmic complexity is well-defined (e.g., time or space complexity), the notion of distributiveness is yet to be quantified fully. The ambiguity of the distributiveness of algorithms stems from its multidimensionality. For example, the ideal distributed algorithm has the feature of symmetric implementation, i.e., it can be implemented on a homogenous network (a set of identical computing devices). Moreover, the devices are expected to utilize only local information, since, in order to reduce the communication bandwidth, it is desirable to have a minimal amount of information exchange between the devices. This is due to the fact that the exchange of information between network elements is very often more expensive than information retrieval and storage on a single centralized system. These features of an algorithm oftentimes come at the cost of reduced system performance. Therefore, in contrast to centralized algorithms, the distributed algorithms discussed in this dissertation do not always achieve the maximum performance, i.e., these algorithms operate suboptimally. In particular, in Chapter 3, the price for enabling distributed decision making (scheduling) is the reduction in the stability region. In the past, this tradeoff between distributiveness and the performance of algorithms has been overlooked by the majority of designers that consider small systems. This aspect is both essential for the design of large-scale systems and difficult to evaluate/quantify.

APPENDICES

APPENDIX A

Appendix for Chapter 2

A.1 Proof of Lemma 2.2

1) Statement (2.2): The union bound and the fact that all $\{k_N(i)\}_{i=1}^N$ are identically distributed yield

$$\mathbb{P}\left[\min_{1\le i\le 1/a_N} k_N(i) \ge \varepsilon a_N N\right] \ge 1 - a_N^{-1}\mathbb{P}[k_N(i) < \varepsilon a_N N].$$

The last term in the preceding inequality can be bounded by applying Markov's inequality and invoking the independence of node locations on the unit square:

$$\begin{aligned}\mathbb{P}[k_N(i) < \varepsilon a_N N] &= \mathbb{P}[e^{-sk_N(i)} > e^{-s\varepsilon a_N N}]\\ &\le e^{s\varepsilon a_N N}\mathbb{E}e^{-sk_N(i)}\\ &\le e^{s\varepsilon a_N N}\left[1 - a_N(1-e^{-s})\right]^N\\ &= e^{s\varepsilon a_N N + N\ln(1-a_N(1-e^{-s}))},\end{aligned} \tag{A.1}$$

for some $s > 0$. Combining (A.1) with relationship $\ln(1-x) \le -x$, for all $x \in (0,1)$, results in

$$a_N^{-1}\mathbb{P}[k_N(i) < \varepsilon a_N N] \le e^{-a_N N(1-e^{-s}-\varepsilon s)-\ln a_N}.$$

Therefore, the lemma holds provided that the exponent on the right-hand side grows without bound as $N \to \infty$:

$$a_N N(1 - e^{-s} - \varepsilon s) + \ln a_N \to \infty. \tag{A.2}$$

The function in (A.2) is monotonic in a_N, implying that it is lower bounded by

$$a(1 - e^{-s} - \varepsilon s) \ln N - \ln N + \ln \ln N + \ln a \tag{A.3}$$

due to the assumption on a_N. Now, under the assumptions of the lemma, it is possible to select $s > 0$ and $\varepsilon > 0$ such that the function (A.3) tends to infinity as $N \to \infty$, and, hence, (A.2) holds. This concludes this part of the proof.

2) Statement (2.3): The proof is very similar to the one of the first part of the lemma. Namely, we have

$$\mathbb{P}\left[\max_{1 \le i \le 1/a_N} k_N(i) \le \gamma a_N N\right] \ge 1 - a_N^{-1}\mathbb{P}[k_N(i) > \gamma a_N N]$$

and

$$\begin{aligned} a_N^{-1}\mathbb{P}[k_N(i) > \gamma a_N N] &\le a_N^{-1} e^{-s\gamma a_N N + N \ln(1 + a_N(e^s - 1))} \\ &\le e^{-a_N N(s\gamma + 1 - e^s) - \ln a_N} \end{aligned} \tag{A.4}$$

for some $s > 0$. Because of the assumptions of the lemma on a_N it is possible to select $\gamma < \infty$ such that the right-hand side of (A.4) tends to 0 as $N \to \infty$. Hence, the statement holds. ■

A.2 Proof of Lemma 2.4

1) Straight-line routing: In order to avoid repetition, we provide a detailed proof of the lemma statement for only one specific direction d – south-to-east. All other values of d can be treated in a similar manner.

Let $\mathcal{C}_N$ be the set of cells located at least δ units away from the nearest edge of the unit square. The cardinality of such a set is at least $(1-\delta)a_N^{-1}$ due to the assumption $a_N \to 0$ as $N \to \infty$. Consider an arbitrary cell $i \in \mathcal{C}_N$.

First, we estimate probability p that cell i forwards packets originating at an arbitrary node in the south-to-east direction. This can be achieved by conditioning on the location of the source node. To this end, suppose that the source node is located at distance r and angle φ relative to the upper-right corner of cell i as illustrated in Fig. A.1. Denote the conditional probability by $p_{r,\varphi}$. The given cell forwards packets in the south-to-east direction if the straight line connecting the source and the respective destination crosses the lower and right edges of the cell (the bold edges in Fig. A.1). For $\sqrt{2a_N} \le r \le \delta$ and $0 \le \varphi \le \pi/4$, this occurs when the the destination is located in the shaded disc sector of angular size ϕ shown in Fig. A.1. The fact that the radial size of the sector is δ and the definition of set $\mathcal{C}_N$ ensures that the sector is within the unit square for N large enough. The uniform distribution of destinations in the unit square yields that $p_{r,\varphi}$ is lower bounded by the area of the shaded sector, i.e.,

$$\begin{aligned} p_{r,\varphi} &\ge \phi[(\delta+r)^2 - r^2]/2 \\ &\ge \phi\delta^2/2. \end{aligned} \tag{A.5}$$

Geometry of the considered case yields that the angle $\phi \le \varphi$ is related to the pair of polar coordinates (r, φ) via

$$\tan(\pi/2 - \varphi - \phi) = \tan(\pi/2 - \varphi) - \sqrt{a_N}/(r \sin\varphi),$$

which renders

$$r \sin\phi = \sqrt{a_N}\sin(\varphi - \phi), \tag{A.6}$$

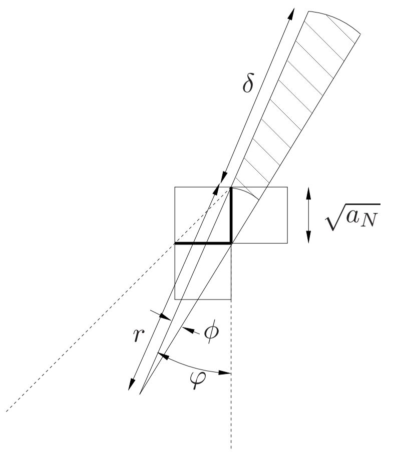


Figure A.1: A cell forwards packets in the south-to-east direction if the straight line connecting the source and destination crosses the two bold lines. Provided that the source is located at (r, φ) in polar coordinates relative to the cell's upper-right corner, the destination can be anywhere inside the shaded disc sector. The figure is plotted for the case $0 \leq \varphi \leq \pi/4$.

for $0 < \varphi \leq \pi/4$. By further restricting values of r and φ to $(\delta/2, \delta)$ and $(\pi/8, \pi/4)$, respectively, it is straightforward to obtain a lower bound on ϕ from (A.6):

$$\phi \geq \zeta \sqrt{a_N}/r, \tag{A.7}$$

for some $\zeta > 0$ and all N large enough; the bound is based on the expansion of the sin function around the origin and $a_N \to 0$ as $N \to \infty$.

Now, integrating the conditional probability $p_{r,\varphi}$ over a restricted range of (r, ϕ) results in the following lower bound

$$\begin{aligned} p &\geq \int_{2\sqrt{a_N}}^{\delta} \int_{0}^{\pi/4} p_{r,\varphi}\, r \, d\phi \, dr \\ &\geq \int_{\delta/2}^{\delta} \int_{\pi/8}^{\pi/4} \frac{\phi\delta^2}{2} r \, d\phi \, dr, \end{aligned}$$

where the second inequality is due to (A.5). The preceding relationship and (A.7) yield a desired bound for some $\xi > 0$

$$p \geq \xi\sqrt{a_N}. \tag{A.8}$$

Second, consider an estimate of $\psi_d(i)$. The independence of node locations and (A.8) imply

$$\mathbb{P}[\psi_d(i) < \varepsilon\sqrt{a_N}N] \leq \mathbb{P}\left[\sum_{j=1}^{N} I_j < \varepsilon\sqrt{a_N}N\right], \tag{A.9}$$

where $\{I_j\}_{j=1}^N$ is an i.i.d. sequence of Bernoulli random variables with $\mathbb{E}I_j = \xi\sqrt{a_N}$; variable I_j lower bounds the indicator function of whether node i forwards packets with source node j in direction d.

Applying Markov's inequality to (A.9) results in, for $s > 0$,

$$\begin{aligned}\mathbb{P}[\psi_d(i) < \varepsilon\sqrt{a_N}N] &\leq e^{s\varepsilon\sqrt{a_N}N}\left[1 - \xi\sqrt{a_N}(1 - e^{-s})\right]^N \\ &\leq e^{s\varepsilon\sqrt{a_N}N + N\ln(1-\xi\sqrt{a_N}(1-e^{-s}))} \\ &\leq e^{-\sqrt{a_N}N(\xi(1-e^{-s})-\varepsilon s)},\end{aligned}$$

where the last inequality follows from $\ln(1-x) \leq -x$ for all $x \geq 0$ small enough. For a given value of ξ it is feasible to select s and $\varepsilon > 0$ such that $\xi(1 - e^{-s}) - \varepsilon s > 0$. This fact, coupled with the assumption $\sqrt{a_N} \geq 1/\sqrt{N}$, yields the statement of the lemma.

2) Shortest-path routing: As in the proof of the statement for the straight-line scheme, we focus on one direction d – south-to-east; all other direction can be treated similarly.

Define $\mathcal{C}_N$ as the set of all cells that are at least $(\lceil\delta/\sqrt{a_N}\rceil + 1)$ *cells* away from the boundary of the unit square. The cardinality of such a set is at least $(1-\delta)a_N^{-1}$ for N large enough. Let i be an arbitrary cell in $\mathcal{C}_N$.

Next, we lower bound the probability p that cell i forwards packets with the source at an arbitrary node in direction d. To this end, conditioning on the source and destination cells with relative locations (j, l) and (u, v), respectively (see Fig. A.2), and a path-

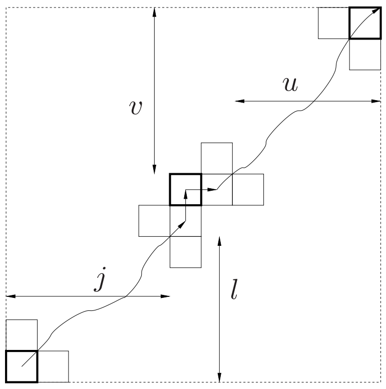


Figure A.2: Probability p of forwarding packets of a flow in the south-east direction by the cell in the center of the figure can be estimated by conditioning on the location of source and destination cells (lower-bottom and upper-left, respectively). All the distances shown are measured in cells.

counting argument yield

$$\begin{aligned} p &\geq \frac{a_N^2}{4} \sum_{j,l=1}^{\lceil \delta/\sqrt{a_N} \rceil} \sum_{u,v=1}^{\lceil \delta/\sqrt{a_N} \rceil} \binom{j+l}{j} 2^{-(j+l)} \\ &\geq \frac{a_N \delta^2}{4} \sum_{j=\lfloor \delta/(2\sqrt{a_N}) \rfloor}^{\lceil \delta/\sqrt{a_N} \rceil} \sum_{l=\lfloor j-\sqrt{j} \rfloor}^{\lceil j \rceil} \binom{j+l}{j} 2^{-(j+l)}. \end{aligned}$$

Since the relative values of j and l are large when N is large, Stirling's approximation is applicable and renders an estimate on the summands in the preceding inequality:

$$\begin{aligned} \binom{j+l}{j} 2^{-(j+l)} &\geq \frac{1}{2\pi} \sqrt{\frac{j+l}{jl}} \frac{(\frac{j+l}{2})^{j+l}}{j^j l^l} \\ &\geq \zeta/\sqrt{j}. \end{aligned}$$

for some $\zeta > 0$. Therefore, the following holds for N large enough

$$\begin{aligned} p &\geq \frac{a_N \delta^2}{4} \sum_{j=\lfloor \delta/(2\sqrt{a_N}) \rfloor}^{\lceil \delta/\sqrt{a_N} \rceil} \sum_{l=\lfloor j-\sqrt{j} \rfloor}^{\lceil j \rceil} \zeta \left\lfloor \frac{\delta}{2\sqrt{a_N}} \right\rfloor^{-1/2} \\ &\geq \zeta \sqrt{a_N} \delta^3/16, \end{aligned}$$

i.e., $p \geq \xi\sqrt{a_N}$, for some $\xi > 0$. The rest of the proof is identical to the second part of the proof for the straight-line scheme – see the part that follows (A.8). ■

APPENDIX B

Appendix for Chapter 3

B.1 Proof of Theorem 3.1

In order to prove Theorem 3.1, we introduce the fluid model of a switch. The fluid model is a deterministic, continuous system of equations that parallels (3.1). Recall that $S_{i,j}(t)$ is the service provided to queue (i,j) in the time slot t. We can rewrite (3.1) as the following:

$$Q_{i,j}(t) = Q_{i,j}(0) + \sum_{\tau=0}^{t} A_{i,j}(t) - D_{i,j}(t), \tag{B.1}$$

where $D_{i,j}(t) = \sum_{\tau=0}^{t} S_{i,j}(\tau)$ is the cumulative number of departures from queue (i,j) up to the time slot t; $D_{i,j}(0) = 0$. For each $t \geq 0$, the evolution of the fluid model is governed by

$$\bar{Q}_{i,j}(t) = \bar{Q}_{i,j}(0) + \lambda_{i,j} t - \bar{D}_{i,j}(t) \geq 0, \tag{B.2}$$

where $\bar{Q}_{i,j}(t) = \lim_{r\to\infty} Q(rt)/r$ and $\bar{D}_{i,j}(t) = \lim_{r\to\infty} D(rt)/r$ are the fluid limits of $Q_{i,j}$ and $D_{i,j}$, respectively. A proof of convergence to a unique limit can be found in [20].

Note that $\bar{D}$ depends on the scheduling algorithm. We specify the rate of departure $\dot{\bar{D}}(t)$ at time t in the fluid model under algorithm $\mathcal{A}$ by defining a function ψ; $\dot{\bar{D}}(t)$ denotes the derivative of $\bar{D}(t)$ at t. For any square matrix X, we use X^{β} to denote a matrix of the same size with elements $X_{i,j}^{\beta}$, $X_{(\lambda)}$ denotes the matrix with elements $X_{i,j}/\lambda_{i,j}$, and $\|X\| = \sum_{i,j=1}^{N} X_{i,j}$. Function ψ represents the expected service rate of the virtual queues under $\mathcal{A}$, and can be recursively defined as follows.

Definition B.1. $\psi(X)$ is a matrix with the same dimension as the square matrix X with non-negative elements. If X is a 1×1 matrix, i.e., a real number, then $\psi(X) = 1$. For an $N \times N$ matrix X, $\psi(X)$ is an $N \times N$ matrix with elements

$$\psi_{i,j}(X) = \frac{X_{i,j}}{\|X\|} + \sum_{k \neq i, l \neq j} \frac{X_{k,l}}{\|X\|} \psi_{i,j}(X_{-(k,l)}),$$

where $X_{-(k,l)}$ is the $(N-1) \times (N-1)$ matrix obtained by deleting the kth row and lth column from X.

Next lemma follows directly from the definition of ψ.

Lemma B.1. $\psi(cX) = \psi(X)$ for any constant $c > 0$.

Note that by conditioning on the first timer to expire under $\mathcal{A}(W)$, it is straightforward to obtain $\mathbb{E}S_{i,j}(t) = \psi_{i,j}(W(t))$. From the construction of the fluid model, we have

$$\begin{aligned}\dot{\bar{D}}_{i,j}(t) &= \lim_{\delta \downarrow 0} \lim_{r \to \infty} \frac{1}{r\delta} \left(D_{i,j}(r(t+\delta) - D_{i,j}(rt)) \right) \\ &= \lim_{\delta \downarrow 0} \lim_{r \to \infty} \frac{1}{r\delta} \sum_{\tau = \lfloor rt \rfloor}^{\lfloor r(t+\delta) \rfloor} S_{i,j}(Q^{\beta}(\tau)).\end{aligned}$$

For each index pair (i,j), let

$$t_{i,j}^{*} = \frac{1}{r} \arg \max_{\tau = \lfloor rt \rfloor \cdots \lfloor r(t+\delta) \rfloor} \psi_{i,j}(Q^{\beta}(\tau))$$

be the time index that maximizes function $\psi_{i,j}$ in the specified time interval. We can upper bound $\dot{\bar{D}}_{i,j}$ as follows when $\bar{Q}_{i,j}(t) > 0$:

$$\begin{aligned}
\dot{\bar{D}}_{i,j}(t) &\leq \lim_{\delta \downarrow 0} \lim_{r \to \infty} \frac{1}{r\delta} \left(\lfloor r(t+\delta) \rfloor - \lfloor rt \rfloor \right) \psi_{i,j}(Q^{\beta}(rt^*_{i,j})) \\
&= \lim_{\delta \downarrow 0} \lim_{r \to \infty} \psi_{i,j}(r^{-1} Q^{\beta}(rt^*_{i,j}))) \\
&= \lim_{\delta \downarrow 0} \psi_{i,j}(\bar{Q}^{\beta}(t^*_{i,j})) \\
&= \psi_{i,j}(\bar{Q}^{\beta}(t)),
\end{aligned}$$

where the second equality is due to Lemma B.1 and the third equality follows from the continuity of ψ; it is clear from the definition of $t^*_{i,j}$ that $t \leq t^*_{i,j} \leq t + \delta$. Analogously, we can obtain a corresponding lower bound, which combined with the upper bound, yields

$$\dot{\bar{D}}_{i,j}(t) = \psi_{i,j}(\bar{Q}^{\beta}(t)), \text{ if } \bar{Q}_{i,j}(t) > 0. \tag{B.3}$$

Equations (B.2) and (B.3) together form the fluid equations of the system operating under algorithm $\mathcal{A}(Q^{\beta})$. Any solution $(\bar{D}, \bar{Q})$ to the fluid equations is a fluid model solution under algorithm $\mathcal{A}(Q^{\beta})$. A switch operating under algorithm $\mathcal{A}(Q^{\beta})$ is rate stable if for every corresponding fluid model solution $(\bar{D}, \bar{Q})$ with $\bar{Q}(0) = 0$, we have $\bar{Q}(t) = 0$ for $t \geq 0$ (see Theorem 3 in [20]). Let $\langle \cdot, \cdot \rangle$ be the element-wise inner product operator of matrices. By Lemma 1 in [20], it is sufficient to show that

$$\frac{d}{dt} \langle \bar{Q}^{\beta}(t), \bar{Q}_{(\lambda)}(t) \rangle \leq 0 \tag{B.4}$$

for any $\bar{Q}(t) \neq 0$. The derivative in (B.4) satisfies

$$\begin{aligned}
\frac{d}{dt} \langle \bar{Q}^{\beta}(t), \bar{Q}_{(\lambda)}(t) \rangle &= (1+\beta) \langle \bar{Q}^{\beta}(t), \dot{\bar{Q}}_{(\lambda)}(t) \rangle \\
&= (1+\beta) \left(\|\bar{Q}^{\beta}(t)\| - \langle \bar{Q}^{\beta}(t), \dot{\bar{D}}_{(\lambda)}(t) \rangle \right),
\end{aligned} \tag{B.5}$$

where the second equality is due to (B.2). In the following discussion, index t may be dropped when there is no ambiguity.

(2 × 2 switch.) For any admissible 2×2 arrival matrix $[\lambda_{i,j}]$, there exists a matrix $[\lambda^*_{i,j}]$ such that $\lambda^*_{i,j} > \lambda_{i,j}$ and $\lambda^*_{1,1} + \lambda^*_{1,2} = \lambda^*_{2,1} + \lambda^*_{2,2} = 1$, since $[\lambda_{i,j}]$ satisfies (3.4). Definition B.1 and (B.3) yield

$$\dot{\bar{D}}_{i,j}(t) = \begin{cases} M_1 \|Q^\beta\|^{-1} 1_{\{Q_{i,i}>0\}}, & i = j, \\ M_2 \|Q^\beta\|^{-1} 1_{\{Q_{i,j}>0\}}, & i \neq j, \end{cases}$$

where $M_1 = \bar{Q}^\beta_{1,1} + \bar{Q}^\beta_{2,2}$ and $M_2 = \bar{Q}^\beta_{1,2} + \bar{Q}^\beta_{2,1}$, and, thus,

$$\begin{aligned} \langle \bar{Q}^\beta, \dot{\bar{D}}_{(\lambda)} \rangle &\geq \langle \bar{Q}^\beta, \dot{\bar{D}}_{(\lambda^*)} \rangle \\ &= \frac{M_1^2/\lambda^*_{1,1} + M_2^2/(1-\lambda^*_{1,1})}{\|Q^\beta\|}. \end{aligned}$$

The preceding inequality and (B.5) result in

$$\begin{aligned} \frac{d}{dt}\langle \bar{Q}^\beta(t), \bar{Q}_{(\lambda)}(t) \rangle &\leq \frac{1+\beta}{\|Q^\beta\|}\left(-\frac{1-\lambda^*_{1,1}}{\lambda^*_{1,1}} M_1^2 - \frac{\lambda^*_{1,1}}{1-\lambda^*_{1,1}} M_2^2 + 2M_1M_2 \right) \\ &= -\frac{1+\beta}{\|Q^\beta\|}\left(\sqrt{\frac{1-\lambda^*_{1,1}}{\lambda^*_{1,1}}} M_1 - \sqrt{\frac{\lambda^*_{1,1}}{1-\lambda^*_{1,1}}} M_2 \right)^2 \leq 0. \end{aligned}$$

Hence, the first statement of the theorem follows.

(N × N switch.) Next, we consider a switch of arbitrary size under uniform traffic. The proof is by induction on the number of input/output ports N. In the case of $N = 1$, (B.4) holds trivially. Now, suppose that (B.4) holds for a switch with size $(N-1) \times (N-1)$; then (B.2), (B.4) and (B.5) imply

$$\langle \bar{Q}^\beta, \dot{\bar{D}}_{(\lambda)} \rangle = (N-1) \sum_{i,j=1}^{N-1} \bar{Q}^\beta_{i,j} \psi_{i,j}(\bar{Q}^\beta) \geq \|\bar{Q}^\beta\|, \tag{B.6}$$

where $\bar{Q}$ is a $(N-1) \times (N-1)$ matrix. For a switch of size $N \times N$, combining (B.2), Definition B.1 and $\lambda_{i,j} = 1/N$ yields

$$\begin{aligned}\langle \bar{Q}^\beta, \dot{\bar{D}}_{(\lambda)} \rangle &= N \sum_{i,j=1}^{N} \bar{Q}^\beta_{i,j} \left(\frac{\bar{Q}^\beta_{i,j}}{\|\bar{Q}^\beta\|} + \sum_{k\neq i, l\neq j} \frac{\bar{Q}^\beta_{k,l}}{\|\bar{Q}^\beta\|} \psi_{i,j}(\bar{Q}^\beta_{-(k,l)}) \right) \\ &= \frac{N}{\|\bar{Q}^\beta\|} \left(\|\bar{Q}^{2\beta}\| + \sum_{k,l=1}^{N} \bar{Q}^\beta_{k,l} \sum_{i\neq k, j\neq l} \bar{Q}^\beta_{i,j} \psi_{i,j}(\bar{Q}^\beta_{-(k,l)}) \right) \\ &\geq \frac{N}{\|\bar{Q}^\beta\|} \left(\|\bar{Q}^{2\beta}\| + \frac{1}{N-1} \sum_{k,l=1}^{N} \bar{Q}^\beta_{k,l} \|\bar{Q}^\beta_{-(k,l)}\| \right), \end{aligned} \tag{B.7}$$

where the inequality follows from applying the inductive hypothesis (B.6) to matrix $\bar{Q}^\beta_{-(k,l)}$. Next, let $q_{i,j} = \bar{Q}^\beta_{i,j} / \|\bar{Q}^\beta\|$ and rewrite the right-hand side of (B.7) as

$$\begin{aligned} &N\|\bar{Q}^\beta\| \sum_{k,l=1}^{N} q_{k,l} \left(q_{k,l} + \frac{1}{N-1} \sum_{i\neq k, j\neq l} q_{i,j} \right) \\ &\qquad \geq N\|\bar{Q}^\beta\| \inf_{\{q_{k,l}\}} \left\{ \sum_{k,l=1}^{N} q_{k,l} \left(q_{k,l} + \frac{1}{N-1} \sum_{i\neq k, j\neq l} q_{i,j} \right) \right\}, \end{aligned}$$

where the infimum is over all $q_{k,l} \in [0,1]$ such that $\sum_{k,l} q_{k,l} = 1$. The infimum is attained when $q_{k,l} = 1/N^2$ for all k, l, resulting in

$$\langle \bar{Q}^\beta, \dot{\bar{D}}_{(\lambda)} \rangle \geq \|\bar{Q}^\beta\|.$$

The preceding inequality, (B.5) and (B.6) imply the second statement of the theorem.

B.2 Proof of Proposition 3.1

Consider the fluid model described in Appendix B.1:

$$\begin{cases} \bar{Q}_{i,j}(t) = \bar{Q}_{i,j}(0) + \lambda_{i,j} t - \bar{D}_{i,j}(t), \\ \dot{\bar{D}}_{i,j}(t) = \psi_{i,j}(\bar{Q}^\beta(t)), \text{ if } \bar{Q}_{i,j}(t) > 0. \end{cases} \tag{B.8}$$

We show that $\bar{Q}_{i,j}(t) \to \infty$ as $t \to \infty$ for $j = i$ or $(i \bmod N) + 1$, under the diagonal traffic with $\alpha = 1/2$ and some $\rho \in (0, 1)$, and the following initial condition:

$$\bar{Q}_{i,j}(0) = \begin{cases} q, & j = i \text{ or } (i \bmod N) + 1, \\ 0, & \text{otherwise,} \end{cases}$$

where $q > 0$, i.e., $\bar{Q}_{i,j}(t) = 0$ for all $t \geq 0$ and i, j such that $\lambda_{i,j} = 0$. Given this initial condition, due to a symmetry in ψ and the arrival matrix, as well as Lemma B.1, (B.8) reduces to

$$\bar{Q}_{i,j}(t) = \begin{cases} q + \rho t/2 - \sigma t, & j = i \text{ or } (i \bmod N) + 1, \\ 0, & \text{otherwise.} \end{cases}$$

for all $t \geq 0$ when $\rho/2 > \sigma$ with σ being the service rate:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T} S_{i,j}(t) = \begin{cases} \sigma, & j = i \text{ or } (i \bmod N) + 1, \\ 0, & \text{otherwise.} \end{cases}$$

That is, for any $\rho > 2\sigma$, the algorithm is not rate stable.

In order to determine the service (departure) rate σ, we construct the interference graph of the switch under the diagonal traffic (see the top of Fig. B.1). Virtual queue (i, j) with positive arrival rate is represented as the node (i, j); the total number of nodes in the interference graph is $2N$. In this graph, two nodes are connected by an edge if they can not be scheduled concurrently. The graph under diagonal traffic is a $2N$-cycle, exhibiting the symmetry of the queues with positive arrival rates. Due to this symmetry, the service rate σ satisfies $\sigma = s_N/2N$, where s_N is the expected number of scheduled queues in a single time slot. Without loss of generality (due to symmetry), we assume that queue $(1, 1)$ is scheduled first, then it follows that queues $(1, 2)$ and $(N, 1)$ can not be scheduled in the same time slot. Thus, the remaining interference graph is linear with $2N - 3$ nodes, as shown in the bottom of Fig. B.1. We argue

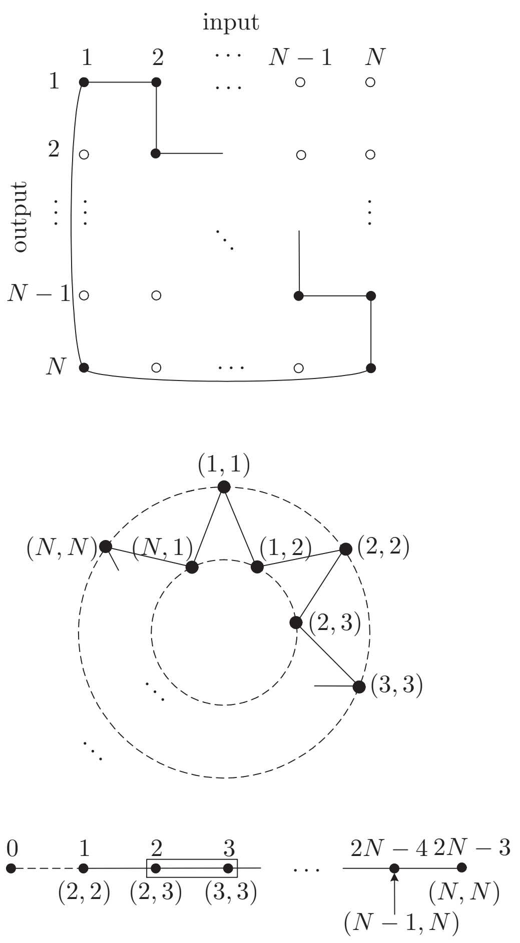


Figure B.1: The interference graph for an $N \times N$ switch under diagonal traffic is shown on the top, where nodes correspond to queues with positive occupancies and arrival rates; edges connect queues with a common input/output port. On the top, the interference graph is drawn according to the relative location of the nodes in the traffic matrix; in the middle the graph is drawn on two circles, each circle refers to a matching of size N. Conditioning on the event that queue $(1,1)$ is scheduled first, the remaining interference graph is linear as shown on the bottom of the figure. Here an additional fictitious node 0 is added to the left end of the line in order to draw an equivalency with a dimer packing problem considered in [65]. A dimer is placed on nodes 2 and 3, indicating that queue $(3,3)$ is scheduled. As a result, queues $(2,3)$ and $(3,4)$ can not be scheduled in the same time slot, since dimers are not allowed to overlap.

that determining the expected number of served queues on a linear interference graph is equivalent to computing the expected number of vacancies on a line packed with randomly placed dimers (non overlapped adjacent pairs of nodes). For convenience, in the following discussion, we relabel the nodes in the interference graph by natural numbers. With an additional fictitious node 0 on the left end of the line interference graph, placing a dimer at node pair $\{i-1, i\}$ is equivalent to an expiration of a timer corresponding to node i. In particular, when a dimer is placed at a node pair $\{i-1, i\}$, then a dimer can not be placed at node pairs $\{i-2, i-1\}$ and $\{i, i+1\}$ since dimers are not allowed to overlap. This is equivalent to preventing queues corresponding to nodes $i-1$ and $i+1$ from being scheduled when the queue corresponding to node i is already scheduled. Hence, if x_k is the expected number of vacancies on a line graph with k nodes (including the node 0), then the following relationship must hold:

$$(2N-3)+1 = 2(s_N - 1) + x_{(2N-3)+1}.$$

The value of x_k was obtained by Page [65]:

$$x_k = (k+2)\left(\sum_{r=0}^{k+1} \frac{(-2)^r}{r!} + \frac{1}{2}\frac{(-2)^{k+2}}{(k+2)!}\right),$$

and, therefore, we have

$$\begin{aligned} s_N &= N - \frac{1}{2}x_{2N-2} \\ &= N\left(1 - \left[\sum_{r=0}^{2N-1} \frac{(-2)^r}{r!} + \frac{1}{2}\frac{(-2)^{2N}}{(2N)!}\right]\right). \end{aligned}$$

Recall that the critical relative load ρ^* can be at most s_N/N. For example, the preceding expression yields $s_{32} \approx 27.6693\ldots$, indicating that $\rho^* < 0.8647\ldots$ for a 32×32 switch (see the top of Fig. 3.6). In addition, it also follows that $\rho^* < 1 - e^{-2}$ in the limit as $N \to \infty$.

B.3 A procedure for estimating ρ^* for $\alpha \in (0,1)$

In order to upper bound the maximum relative load under diagonal traffic, we consider the fluid model analyzed in Appendix B.2. In particular, we focus on the fluid model (B.8) with the following initial condition:

$$\bar{Q}_{i,j}(0) = \begin{cases} \kappa q, & j = i, \\ (1-\kappa)q, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise}, \end{cases} \tag{B.9}$$

where $\kappa \in [0,1]$ is such that

$$\frac{\psi_{i,i}(\bar{Q}^\beta(0))}{\psi_{i,(i \bmod N)+1}(\bar{Q}^\beta(0))} = \frac{\alpha}{1-\alpha}. \tag{B.10}$$

Note that such κ exists for any $\alpha \in [0,1]$ due to continuity and symmetry of ψ. Due to symmetry of the arrival matrix, under (B.9) and (B.10), (B.8) renders for all $t \geq 0$, when $\rho > \sigma$,

$$\bar{Q}_{i,j}(t) = \begin{cases} \kappa q + \alpha(\rho - \sigma)t, & j = i, \\ (1-\kappa)q + (1-\alpha)(\rho-\sigma)t, & j = (i \bmod N) + 1, \\ 0, & \text{otherwise}, \end{cases}$$

where σ is the total service rate of queues (i,i) and $(i,(i \mod N)+1)$. In this case, the algorithm is not rate stable.

The value of κ that yields (B.10) can be evaluated numerically since ψ is defined recursively in Definition B.1. Once this value is obtained, one can obtain $\sigma = \psi_{1,1}(\bar{Q}(0)) + \psi_{1,2}(\bar{Q}(0))$ and an upper bound $\rho^* < \sigma$ follows. For example, when $N = 32$ and $\alpha = 2/3$, we have $\kappa = 0.6295\ldots$, $\psi_{1,1}(\bar{Q}(0)) = 0.5829\ldots$, $\psi_{1,2}(\bar{Q}(0)) = 0.2915\ldots$ and $\sigma = 0.8744\ldots$.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Ajmone Marsan, A. Bianco, E. Leonardi, and L. Milia. RPA: A flexible scheduling algorithm for input buffered switches. *IEEE Trans. Commun.*, 47(12):1921–1933, 1999.

[2] L. Adleman. Molecular computation of solutions to combinatorial problem. *Science*, 266:1021–1024, 1994.

[3] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, and Ph. Whiting. Scheduling in a queuing system with asynchronously varying service rates. *Probab. Engin. Inform. Sci.*, 18:191–217, 2004.

[4] B. Ata and W. Lin. Heavy traffic analysis of maximum pressure policies for stochastic processing networks with multiple bottlenecks. *Queueing Syst. Theory Appl.*, 59(3-4):191-235, 2008.

[5] A. Akella, A .Seshan R. Karp, S. Shenker, and C. Papadimitriou. Selfish behavior and stability of the internet:: a game-theoretic analysis of TCP. *SIGCOMM Comput. Commun. Rev.*, 32(4):117-130, 2002.

[6] G. Barrenechea, B. Beferull-Lozano, and M. Vetterli, "Lattice sensor networks: Capacity limits, optimal routing and robustness to failures," in *Proc. IPSN*, Berkeley, CA, April 2004.

[7] G. Barrenetxea, B. Beferull-Lozano, and M. Vetterli, "Efficient routing with small buffers in dense networks," in *Proc. IPSN*, Los Angeles, CA, April 2005.

[8] A. Buchsbaum, G. Fowler, and R. Giancarlo, "Improving table compression with combinatorial optimization," in *Proc. ACM-SIAM SODA*, San Francisco, CA, January 2002.

[9] T. Bonald, M. Feuillet and A. Proutiere. Is the "Law of the Jungle" Sustainable for the Internet? In *Proc. IEEE Infocom*, Rio de Janeiro, Brazil, April 2009.

[10] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Prentice-Hall, 1992.

[11] T. Bonald and L. Massoulié Impact of fairness on Internet performance. In *Proc. ACM Sigm.*, Santa Clara, California, June 2000.

[12] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia, "Routing with guaranteed delivery in ad hoc wireless networks," *Wireless Networks*, vol. 7, pp. 609–616, 2001.

[13] M. Bayati, B. Prabhakar, D. Shah, and M. Sharma. Iterative scheduling algorithms. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.

[14] P. Chaporkar, K. Kar, and S. Sarkar. Throughput guarantees through maximal scheduling in wireless networks. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2005.

[15] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[16] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* John Wiley & Sons, Inc., 1991.

[17] E.A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

[18] J. Dai and W. Lin. Asymptotic optimality of maximum pressure policies in stochastic processing networks. *Ann. Appl. Probab.*, 18:2239–2299, 2008.

[19] H. Duan, J. Lockwood, and S.M. Kang. Matrix unit cell scheduler (MUCS) for input-buffered ATM switches. *IEEE Commun. Letters*, 2(1):20–23, 1998.

[20] J. Dai and B. Prabhakar. The throughput of data switches with and without speedup. In *Proc. IEEE Infocom*, Tel Aviv, Israel, March 2000.

[21] S. Deb, D. Shah, and S. Shakkottai. Fast matching algorithms for repetetive optimization: An application to switch scheduling. In *Proc. CISS*, Princeton, NJ, March 2006.

[22] N.G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Trans. Networking*, 9(3):280–292, 2001.

[23] A. Dimakis and J Walrand. Sufficient conditions for stability of longest queue first scheduling: Second order properties using using fluid limits. *Adv. Appl. Probab.*, 38(2):505–521, 2006.

[24] A. El Gamal, J. Mammen, B. Prabhakar, and D. Shah, "Optimal throughput-delay scaling in wireless networks - Part I: The fluid model," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2568–2592, 2006.

[25] ——, "Optimal throughput-delay scaling in wireless networks - Part II: Constant-size packets," *IEEE Trans. Inform. Theory*, vol. 52, no. 11, pp. 5111–5116, 2006.

[26] A. Eryilmaz, A. Ozdaglar, and E. Modiano. Polynomial complexity algorithms for full utilization of multi-hop wireless networks. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.

[27] M. Franceschetti, O. Dousse, D. Tse, and P. Thiran, "On the throughput capacity of random wireless networks," *IEEE Trans. Inform. Theory*, vol. 52, no. 6, pp. 2756–2761, 2006.

[28] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the internet. *IEEE/ACM Trans. Netw.*, 7:458V472, 1999.

[29] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE J. Select. Areas Commun.*, 21(4):546–559, 2003.

[30] P. Giaccone, D. Shah, and B. Prabhakar. An implementable parallel scheduler for input-queued switches. *IEEE Micro*, January-February 2002.

[31] H.N. Gablow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853, 1991.

[32] Matthias Grossglauser and David N. C. Tse, "Mobility increases the capacity of ad hoc wireless networks," *IEEE/ACM Trans. Netw.*, vol. 10, no. 4, pp. 477–486, 2002.

[33] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. Inform. Theory*, vol. 46, no. 2, pp. 388–404, 2000.

[34] J. Herdtner and E. Chong, "Throughput-storage tradeoff in ad hoc networks," in *Proc. IEEE Infocom*, Miami, FL, March 2005.

[35] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[36] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, New York, 1995.

[37] P. Jelenković, P. Momčilović, and M. Squillante, "Scalability of wireless networks," *IEEE/ACM Trans. Netw.*, vol. 15, no. 2, pp. 295–308, 2007.

[38] A. Jovičić, P. Viswanath, and S. Kulkarni, "Upper bounds to transport capacity of wireless networks," *IEEE Trans. Inform. Theory*, vol. 50, no. 11, pp. 2555–2565, 2004.

[39] L. Jiang and J. Walrand. A distributed CSMA algorithm for throughput and utility maximization in wireless networks. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2008.

[40] I. Keslassy, S.-T. Chuang, and N. McKeown. A load-balanced switch with an arbitrary number of lines. In *Proc. IEEE Infocom*, Hong Kong, March 2004.

[41] I. Keslassy, C.-S. Chang, N. McKeown, and D.-S. Lee. Optimal load-balancing. In *Proc. IEEE Infocom*, Miami, FL, March 2005.

[42] F. Kelly, *Reversibility and Stochastic Networks.* New York: Wiley, 1979.

[43] B. Karp and H. Kung, "Greedy perimeter stateless routing," in *Proc. ACM MobiCom*, Boston, MA, August 2000.

[44] I. Keslassy and N. McKeown. Analysis of scheduling algorithms that provide 100% throughput in input-queued switches. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, October 2001.

[45] E. Kushilevitz and N. Nisan, *Communication Complexity.* Cambridge University Press, 1997.

[46] P. R. Kumar and L.-L. Xie, "A network information theory for wireless communications: Scaling laws and optimal operation," *IEEE Trans. Inform. Theory*, vol. 50, no. 5, pp. 748–767, 2004.

[47] I. Keslassy, R. Zhang-Shen, and N. McKeown. Maximum size matching is unstable for any packet switch. *IEEE Commun. Letters*, 7(10):496–498, 2003.

[48] F. T. Leighton, *Introduction to parallel algorithms and architectures: Arrays, trees, and hypercubes.* San Mateo, CA: Morgan Kaufmann Publishers, 1992.

[49] O. Leveque and E. Telatar, "Information theoretic upper bounds on the capacity of ad hoc networks," *IEEE Trans. Inform. Theory*, vol. 51, no. 3, pp. 858–865, 2005.

[50] J. Li, J. Jannotti, D. D. Couto, D. Karger, and R. Morris, "A scalable location service for geographic ad hoc routing," in *Proc. ACM MobiCom*, Boston, MA, August 2000.

[51] S. Low. A duality model of TCP and queue management algorithms. *IEEE/ACM Trans. Networking*, 11(4):525–536, 2003.

[52] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Networking*, 7(2):188–201, 1999.

[53] M.G. Ajmone Marsan, P. Giaccone, E. Leonardi, and F. Neri. On the stability of local scheduling policies in networks of packet switches with input queues. *IEEE J. Select. Areas Commun.*, 21(4):642–655, 2003.

[54] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Trans. Comm.*, 47(8):1260–1267, 1999.

[55] P. Momčilović. A distributed switch scheduling algorithm. (In *Proc. of Performance*, Cologne, Germany, October 2007.) *Perform. Eval.*, 64(9-12):1053–1061, 2007.

[56] P. Mani and D. Petr. Clique number vs. chromatic number in wireless interference graphs: Simulation results. *IEEE Commun. Letters*, 11(7):592–594, 2007.

[57] J. Roberts and L. Massoulie. Bandwidth Sharing and Admission Control for Elastic Traffic. *Telecommunication Systems*, 15:185–201, 1998.

[58] J. Roberts and L. Massoulie. Bandwidth sharing: objectives and algorithms *IEEE/ACM Trans. Networking*, 10(3):320–328, 2002.

[59] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proc. ACM Sigmetrics*, Saint Malo, France, June 2006.

[60] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, Cambridge, UK, 2000.

[61] P. Ng, D. Edwards and S. Liew. Colouring link-directional interference graphs in wireless ad hoc networks. In *Proc. IEEE Globecom*, Washington, DC, November 2007.

[62] G. Narlikar and F. Zane, "Performance modeling of fast IP lookups," in *Proc. ACM Sigmetrics*, Cambridge, MA, June 2001.

[63] A. Okabe, B. Boots, and K. Sugihara, *Spatial Tessellations Concepts and Applications of Voronoi Diagrams.* New York: Wiley, 1992.

[64] A. Orlitsky and J. Roche, "Coding for computation," in *Proc. IEEE FOCS*, Milwaukee, WI, October 1995.

[65] E. Page. The distribution of vacancies on a line. *J. Royal Stat. Soc. B*, 21(2):364–374, 1959.

[66] G. Paun. *Membrane computing.* Springer, Berlin, 2002.

[67] S. Fuller, T. Li, J. Yu and K. Varadhan. Classless inter-domain routing(CIDR): an address assignment and aggregation strategy. RFC 1519, September 1993.

[68] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proc. ACM MobiCom*, San Diego, CA, September 2003.

[69] B. Raghavan and A. Snoeren. Decongestion control. In *Proc. of the Fifth Workshop on Hot Topics in Networks (HotNets-V)*, Irvine, California, November 2006.

[70] D. Shah. Maximal matching scheduling is good enough. In *Proc. IEEE Globecom*, San Francisco, CA, December 2003.

[71] D. Shah. *Randomization and heavy traffic: New approaches for switch algorithms.* PhD thesis, Stanford University, 2004.

[72] S. Sarkar and K. Kar. Achieving 2/3 throughput approximations with sequential maximal scheduling under primary interference constraints. In *Proc. of Allerton Conf. on Comm., Control, and Comput.*, Monticello, IL, September 2006.

[73] L. Schrage and L. Miller. The M/G/1 queue with the shortest remaining processing time first discipline. *Operations Research*, 14(4):670–684, 1965.

[74] G. Sharma, N. Shroff, and R. Mazumdar. Joint jongestion control and distributed scheduling for throughput guarantees in wireless networks. In *Proc. IEEE Infocom*, Anchorage, AK, May 2007.

[75] A. Stolyar. MaxWeight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic. *Ann. Appl. Probab.*, 14(1):1–53, 2004.

[76] D. Shah and D. Wischik. Optimal scheuling algorithms for input-queued switches. In *Proc. IEEE Infocom*, Barcelona, Spain, 2006.

[77] L. Tassiulas. Adaptive back-pressure congestion control based on local information. *IEEE Trans. Automat. Control*, 40(2):236–250, 1995.

[78] L. Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proc. IEEE Infocom*, San Francisco, CA, April 1998.

[79] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling for maximum throughput in multihop radio networks. *IEEE Trans. Automat. Control*, 37(12):1936–1949, 1992.

[80] M. Tariq, A. Dhamdhere, C. Dovrolis and M. Ammar. Poisson versus periodic path probing (or, does PASTA matter?). In *Proc. ACM SIGCOMM IMC,*, 11–124, Berkeley, CA, October 2005.

[81] G. De Veciana, T. Lee and T. Konstantopoulos Stability and performance analysis of networks supporting elastic services. *IEEE/ACM Trans. Networking*, 9(1):2–14, 2001.

[82] K. Winick. Personal communication.

[83] Y. Yi and S. Shakkottai. Hop-by-hop congestion contorl over a wireless multi-hop network. *IEEE/ACM Trans. Networking*, 15(1):133–144, 2007.

[84] H. Zhang, D. Towsley and W. Gong. TCP Connection Game: A Study on the Selfish Behavior of TCP Users In *Proc. IEEE ICNP*, Boston, MA, November 2005.