Collaboration and Multitasking in Networks: Capacity versus Queue Control

Itai Gurvich, Jan A. Van Mieghem
Kellogg School of Management, i-gurvich@northwestern.edu, vanmieghem@northwestern.edu,

March 23, 2015

Motivated by the trend towards more collaboration in work flows, we study networks where some activities require the simultaneous processing by multiple types of multitasking human or indivisible resources. The capacity of such networks is generally smaller than the bottleneck capacity. In Gurvich and Van Mieghem (2015) we proved that both capacities are equal in networks with a nested collaboration architecture. This paper shows how this capacity is achieved through, and affected by, dynamic queue control.

Collaboration of indivisible resources presents challenges to four fundamental “congestion laws” in the control of queues. In well-operated traditional networks, (1) average queues remain finite if utilization $\rho < 100\%$; (2) the total workload scales like $1/(1 - \rho)$ as $\rho \to 100\%$; (3) this scaling rate can be maintained while prioritizing certain queues over others (i.e., keeping certain queues relatively small); and (4) $P = NP$: the restriction to Non-preemptive policies carries no significant efficiency loss relative to Preemptive policies.

We provide a conceptual framework and formalize coordination and switching idleness. We then devise decentralized rules that, in parallel collaborative networks with nested architectures, stabilize the network and achieve optimal scaling, addressing (1) and (2) above. In the case of non-preemption, these rules trade off switching and coordination idleness. We proceed to show that collaboration presents challenges to (3) and (4) above: prioritizing specific queues comes at a significant loss of capacity. This capacity loss differs between preemption and non-preemption showing that $P \neq NP$ in these networks, even in heavy-traffic. Our policies balance capacity and queue control while guaranteeing stability and optimal scaling.

Key words: collaboration, architectures, resource sharing, multitasking, priority, stability, control

1. Introduction and Summary of Results
Motivated by the prevalence of collaborative processing in services, we study how simultaneous collaboration and multitasking impact system capacity and responsiveness. Simultaneous collaboration means that some activities require the synchronous processing by multiple types of resources. Discharging a patient, for example, may require the presence of both a doctor and a nurse and chemical distilling requires two vessels. Multitasking means that a resource performs multiple activities. Multitasking is equivalent to resource sharing and means that multiple activities require the same resource. A doctor, for example, may be required for both patient diagnosis and for patient discharge. Simultaneous collaboration imposes constraints on the capacity of the process because multitasking resources have to be simultaneously at the right place. Human operators magnify the effect of these synchronization requirements because they are indivisible and cannot be “split.” An
Collaboration and Control in Networks

2

Article submitted to : manuscript no.

Figure 1 (LEFT) The Basic Collaboration (BC) network with a nested collaboration architecture. (RIGHT) Adding a third resource yields the BC+ network: the prototypical non-nested network.

emergency physician may split her time among multiple activities—spending $x\%$ of her time in one activity and the remaining $(100 - x)\%$ in another—and she may switch between activities frequently, yet she cannot process two activities at the same time (which may, in this example, require her physical presence in two distinct locations).

Collaboration of indivisible multitasking resources poses significant operational and interesting intellectual challenges. In general, the capacity of such networks is smaller than the bottleneck capacity. Consider, for example, the basic collaboration (BC) network on the left of Figure 1 with three activities ($a_1, a_2$ and $a_3$) and two multitasking resources (R1 and R2). Only resource $i \in \{1, 2\}$ is required for its individual activity $a_i$ while both resources are needed for the collaborative task $a_0$. Given mean arrival rate $\lambda_i$ and mean service time $m_i$ for activity $i$, the load on resource $i$ is

$$\rho_i = \lambda_0 m_0 + \lambda_i m_i.$$ 

Resource $j$ is a bottleneck if $j \in \arg\max\{\rho_1, \rho_2\}$ and the bottleneck load is

$$\rho^{BN} = \max\{\rho_1, \rho_2\}.$$ 

Processing networks can typically ‘handle any arrival rates’ (while keeping the system stable) as long as $\rho^{BN} < 1$, but that need not be true when indivisible multitasking resources collaborate. To see what can go wrong, consider the BC+ network in the right panel of Figure 1 where we added resource 3. With this resource we also added a collaboration dependency. The network must, at any given time, use one of only three feasible configurations vectors that specify which activities are “active.” As no two activities can be performed in parallel, only the unit vectors are feasible. For an arrival rate vector to be sustainable there must exist time allocations $\pi_1, \pi_2, \pi_3$ such that

$$\pi_0 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \pi_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \pi_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix}$$

for some $\pi \geq 0$ with $\sum_i \pi_i \leq 1$. The network utilization $\rho^{net}(\lambda) := \sum_i \pi_i$ can be interpreted as the average time it takes the network to process an input $\lambda$. This will in general exceed the time, $\rho^{BN}$,
it takes the bottlenecks, working in isolation, to process that input. If \( m_j \equiv 1 \) and \( 1/3 < \lambda_1 = \lambda_2 = \lambda_3 < 1/2 \), then \( \rho^{\text{BN}} < 1 \) yet \( \rho^{\text{net}}(\lambda) \geq 1 \) and the queues would explode.

Notice that if resources were divisible, there would be no capacity loss in the BC+ network: allocating a fraction of each resource to each of its activities would split the network into three independent queues. Each of these would be stable as long as \( \rho^{\text{BN}} < 1 \) and its expected stationary queue would scale like \( 1/(1 - \rho^{\text{BN}}) \) as \( \rho^{\text{BN}} \to 1 \).

The BC and BC+ networks fundamentally differ in their underlying collaboration architecture. A network’s architecture specifies how resources are assigned to activities. In Gurvich and Van Mieghem (2015) (further abbreviated to GVM) we developed necessary and sufficient conditions on this architecture that guarantee that the theoretical network capacity equals the bottleneck capacity. Networks with nested architectures do not introduce Unavoidable Bottleneck Idleness: \( \text{UBI}(\lambda) = \rho^{\text{net}}(\lambda) - \rho^{\text{BN}}(\lambda) = 0 \) for each \( \lambda \geq 0 \). In this paper, we specify how this theoretical network capacity is achieved through dynamic control and explore the interaction of throughput maximization, workload scaling, and queue control in these intriguing networks.

To conceptualize the challenges and tradeoffs we introduce two types of collaboration idleness that impact capacity more subtly than UBI. UBI is caused by the indivisibility of resources that prevent fractional capacity allocations. In nested networks, only integer allocations are optimal and hence indivisibility is costless (UBI = 0).

Dynamic control requires synchronized (joint) movements of resources between tasks. It is useful to think of the control rule setting an alarm that signals to resources when to switch. In the BC network, such alarm signals when resources should move from the individual tasks to the collaborative task, and back. The network incurs coordination idleness when a resource is idly waiting for the alarm to move to a collaborative task that has work—it is waiting for a coordinated move. In the BC network, resource 1 might not have work in activity \( a_1 \) but, depending on the control rule, might have to wait until the alarm sounds before moving to \( a_0 \) where there could be work. The network incurs switching idleness when a resource is idly waiting after the alarm has sound but before moving to the collaborative task. This happens only under non-preemptive controls when a resource has finished its service but is waiting for the remaining required resources to finish theirs. These two types of idleness are formalized in §5.

In contrast to UBI, coordination and switching idleness depend not only on the architecture but also on the control policy. Essentially, the idleness that a control imposes on the network is:

\[
\text{Collaboration idleness} = \text{Coordination idleness} + \text{Switching idleness}.
\]
Figure 2 The BC network of Fig. 1 can be relaxed by duplicating the arrivals to the collaborative task (LEFT) to yield the benchmark (Two-class M/M/1)² network (RIGHT).

It is instructive to introduce the benchmark system obtained from the BC network by duplicating the arrival stream to the collaborative task (left panel in Fig. 2). This breaks the collaboration constraint and allows the resources to work on the collaborative jobs separately. Effectively, each resource sees a two-class M/M/1 queue (bottom panel). The performance gap between the benchmark and the BC network represents the “cost” of collaboration.

The benchmark system exhibits four fundamental phenomena (“congestion laws”) in well-operated traditional queueing networks:

1. **Maximal throughput**: there exist a control policy that stabilizes the system for any arrival rate for which utilization $\rho < 1$.

2. **Optimal scaling**: the total workload scales like $1/(1 - \rho)$ as utilization $\rho \to 100\%$. Indeed, under any work conserving policy, the steady-state total number in an M/M/1 system, satisfies

$$E Q_+ (\infty) = \frac{\rho}{1 - \rho} = O \left( \frac{1}{1 - \rho} \right).$$

Characterizing the policies (if they exist) that guarantee this optimal scaling condition is a fundamental and active research question in queueing networks. We review the networks for which optimal scaling has been established in §2.

3. **Controllability**: the optimal scaling rate can be maintained while prioritizing certain queues over others (i.e., keeping certain queues relatively small). Indeed, whether one prioritizes queue 1 or 2 in a two-class M/M/1, the total workload follows the optimal scaling as long as the policy is work conserving. We call this *controllability*: the freedom to prioritize certain queues over others without significantly compromising the performance of the total queue count or workload.

4. **P = NP**: the restriction to non-preemptive policies carries no significant cost. Indeed, the impact of the service discipline (preemptive or non-preemptive) on queue lengths in a two-class M/M/1 system is small, as shown in Figure 3. In particular, it is negligible relative to $1/(1 - \rho)$.
Figure 3  The two-class $M/M/1$ queue illustrates the four “congestion laws.” (For $\lambda_1 m_1 = \lambda_2 m_2 = \rho/2$)

\[ \rho \to 1, \]  
the high priority queues remain finite under either discipline—and hence, when scaled by $(1 - \rho)$ both approach 0—while the low priority queues hold all the work and are $O(1/(1 - \rho))$. This is the reason that, in the multiclass $M/M/1$ queue with linear delay costs, the optimal preemptive and non-preemptive policies generate basically an identical result for large values of $\rho \to 1$. It is central to the dynamic control of queueing networks in heavy traffic that the restriction to non-preemptive policies on the objective function is negligible relative to the total delay cost.

The following summarizes our findings with regards to congestion laws in collaborative networks with multitasking indivisible resources:

1. **Maximal throughput**: We show that maximum pressure policies (introduced by [Tassiulas and Ephremides (1992); Dai and Lin (2005) and explained below) stabilize collaborative open queueing networks with indivisible resources for $\rho_{\text{BN}} < 1$ as long as the collaboration architecture is nested and preemption is allowed. For parallel networks with nested architecture, we present a natural nested priority rule and prove that it also maximizes throughput.

   If preemption is not allowed, neither maximum pressure nor priority policies maximize through-put. Frequent switching induces significant switching idleness that destabilizes the network for values of $\rho_{\text{BN}}$ strictly below 1. We prove that a natural decentralized polling policy where servers move only after they exhaust their own queue stabilizes the network for $\rho_{\text{BN}} < 1$ but introduces significant coordination idleness; see item 2 below. For parallel networks with nested architecture, we show that a natural nested priority with thresholds rule also maximizes throughput.

2. **Optimal scaling**: We show that in parallel collaborative networks maximum pressure guarantees the optimal scaling if the collaboration architecture is nested and preemption is allowed. If
preemption is not allowed care is needed. Decentralized polling stabilizes the system but introduces significant coordination idleness and leads to queues of order \((1 - \rho_{BN}^{3})^{-3/2}\). We prove that nested priorities with thresholds achieve the optimal scaling in parallel networks with nested architectures.

3 & 4. **Queue Controllability** and \(P = NP\): We show that both laws are violated for networks of collaborating, multitasking, indivisible resources: one cannot simply tag a subset of the queues as priority queues without losing significant throughput. Moreover, there is asymmetry: collaborative queues can be controlled (made small) with preemption but at most a subset (rather than all) individual-resource queues can be made small without losing capacity. With non-preemption, all queues must grow proportionally (rather than negligibly) relative to \(1/(1-\rho)\), so that the inequality \(P \neq NP\) persists in heavy-traffic. We prove that this tradeoff between capacity and controllability holds for any policy.

The tradeoffs of queue control and capacity in the BC network is captured by the \(2 \times 2\) matrix in Figure 4. We contrast priority to the collaborative queue (C priority) with priority to both individual queues (\(I^2\) priority) and also compare to the two-class \(M/M/1\) benchmark. While the plots use specific priority policies and parameters\(^1\) we prove that these are universal properties (i.e, that there is no control policy that can avoid these properties):

I. Preemptive priority to the collaborative task achieves maximal throughput and optimal scaling. This is the only quadrant in Figure 4 where collaboration comes at no cost. It also shows that the two-class \(M/M/1\) benchmark is tight: it is achievable.

II. Non-preemptive priority to the collaborative task leads to significant switching idleness and results in a capacity loss. In the simulation in Figure 4 throughput cannot exceed a bottleneck utilization of roughly 0.9. To generalized this observation we prove a tradeoff result: there exists no control policy that makes the collaborative queue short and keeps the total expected queue count finite as \(\rho_{BN}\) approaches 1.

III. Preemptive priority to both individual tasks leads to significant coordination idleness and results in capacity loss. We prove that the maximum throughput in this example is \(3 - \sqrt{5} = 0.764\).

IV. Non-preemptive priority to both individual tasks leads to significant capacity loss. While the system now incurs both coordination and switching idleness, it performs slightly better than the preemptive case because the switching frequency is reduced.

In summary: Except for one quadrant, collaboration introduces, either through excessive switching or coordination idleness, a fundamental trade-off between queue control and capacity and present a challenge to the congestion laws. The nested priority threshold policy that we introduce balances coordination and switching idleness traces this trade-off curve; see \(\S 6.2\).

\(^1\) The plots show the average queue lengths during a simulation of 10 million time units where \(m_i = 1/2\) and \(\lambda_i = \rho\).
2. Literature review

This is a continuation of our work on collaborative networks in Gervich and Van Mieghem (2015) (GVM) that studied the relationship between network and bottleneck capacity. With indivisible resources only integral allocations of servers to activities are feasible. We proved that in networks with nested collaboration architecture all optimal allocation are, in fact, integral so that indivisibility does not incur UBI. This integrality property together with graph-based property of nested architectures will be important building blocks for this paper. We refer the reader to GVM for a detailed review of collaboration and resource sharing in networks and will discuss below what specifically pertains to this paper.

We structure this literature review according to the four congestion laws described in the introduction and Table I serves as a roadmap.
Table 1  Summary of results in the literature and in this paper.

<table>
<thead>
<tr>
<th>divisible processors</th>
<th>Preemptive (Coordination Idleness)</th>
<th>indivisible processors</th>
<th>Non-Preemptive (Switching Idleness)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>Max Pressure</td>
<td>(1) Max Pressure (open nets)</td>
<td>(1) Polling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2) Nested priorities</td>
<td>(2) S-Nested priorities</td>
</tr>
<tr>
<td>Optimal scaling</td>
<td>Static splitting</td>
<td>Nested Priorities</td>
<td>(1) S-nested Priorities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(2) Polling; $(1 - \rho_{BN})^{-2}$</td>
</tr>
<tr>
<td>Controllability</td>
<td>$T^I$-priority = Throughput loss</td>
<td>(1) $T^I$-Priority = Throughput loss</td>
<td>(2) $T^I$-Priority = Throughput loss</td>
</tr>
<tr>
<td></td>
<td>(Workload dimension)</td>
<td>(2) $I^I$-Priority = Workload cost</td>
<td>(2) $I^I$- and $C^I$-priority = Throughput loss</td>
</tr>
</tbody>
</table>

Stability in networks is a central topic in queueing theory. We mostly relate here to the work on maximum pressure policies introduced by Tassiulas and Ephremides (1992) and their stability analysis by Dai and Lin (2005).

A vast literature studies performance analysis and optimization of resource sharing in networks, often inspired by communication networks that typically assume divisible resources; e.g., see Harrison et al. (2014) and the many references therein. To emphasize the connections, Figure 5 juxtaposes the different, but equivalent, pictorial network representations used in this paper versus typical papers on queuing networks and communication networks.

With divisible resources, the question of stability trivializes in the single-stage parallel networks like the BC network: a static allocation of fractional server capacity to each activity decouples the network into single activity-single server queues. Questions of optimal scaling are challenging even without resource sharing and fuel a growing body of research; see e.g., Gamarnik and Zeevi (2006); Budhirja and Lee (2009); Gurvich (2013). In the case of resource sharing of divisible resources Shah et al. (2014) only recently established that fair bandwidth-sharing policies achieve optimal scaling in parallel networks with general sharing or collaboration structures. In terms of control, Harrison et al. (2014) also shows that the equivalent Brownian workload formulation (Harrison and Van Mieghem (1997)) for the BC network is two-dimensional which suggests that simultaneous minimization of both individual queues may not be possible in a sequence of networks approaching heavy-traffic. Verloop and Núñez-Queija (2009) studies total queue length minimization in the BC network with preemption and offers in general a switching curve policy.

Relative to the impressive literature on stochastic networks, our paper studies the limitations imposed by collaborating, multitasking resources, focusing especially on their indivisibility and their service disciplines. Our first paper establishes that indivisibility becomes costless in nested networks, which we also consider in this paper. We also show that the maximum pressure policies, which typically assume divisible resources, can be applied in nested networks with indivisible resources: they stabilize them and provide optimal scaling assuming preemptive service. In our
arguments of stability and scaling bounds with indivisible resources we build on recent methods that are cited in the corresponding places in the proofs. Interestingly, the explicit study of collaboration also bring simplifications to some results in the stochastic networks literature. We provide two policies (one for preemption and a threshold variant for non-preemption) that also exhibit optimal scaling of $1/(1 - \rho^\text{RN})$ in parallel networks with nested architecture.

Importantly, indivisibility of collaborating multitasking resources brings about the need to classify the types of idleness that can be incurred due to synchronization conflicts and relate them to fundamental questions of control and capacity.

Finally, queues with collaboration turn out to be closely related to queues with switchover times and those with interruptions. Borst [1996] covers performance analysis for a variety of polling systems with switchover times; for optimization and control see, e.g., Reiman and Wein [1998]; Lan and Olsen Lennon [2006] and the references therein. While switchover times are typically exogenous parameters in the polling literature, they arise endogenously in collaborative networks. Some of our priority policies the work in the collaborative task represents an “interruption” for the individual queue; see the survey by Krishnamoorthy et al. [2014].

3. Network model, configuration vector and collaboration architecture

There is a set $\mathcal{K} = \{1, \ldots, K\}$ of resources and a set $\mathcal{J} = \{1, \ldots, J\}$ of activities. Each activity is associated with a single buffer: there are $J$ buffers. The average arrival rate into buffer $i$ is $\lambda_i > 0$. The average processing time of activity $i$ is $m_i > 0$. Unless stated otherwise, arrivals follow independent Poisson processes and service times are independent across customers and exponentially

![Diagram](image-url)
distributed. For most of the paper (Theorem 3 and Corollary 1 are the exception) we focus on parallel networks. These are networks in which customers arrive into a queue, are served, and then leave the system (rather than moving to another queue).

We adopt the following notation: All vectors are represented as column vectors. The vector of all ones is denoted by $e$, or by $e_d$ to emphasize the dimension $d$. Componentwise multiplication is denoted by $\ast$ so that $v \ast w$ represents the vector with $i$-th component $v_i w_i$. The component sum $\sum_k v_i$ is denoted by $v_+$. Following standard terminology we introduce the $K \times J$ resource-activity incidence matrix $A$ where $A_{ki} = 1$ if resource $k$ is required for activity $i$, and $A_{ki} = 0$ otherwise. The distinguishing feature of collaborative networks is that $A$ has at least one column (activity) with multiple 1’s (collaborative resources). The distinguishing feature of multitasking is that at least one row (resource) has multiple 1’s. For $i \in J$, we let $\mathcal{R}([i])$ be the set of resources required for activity $i$ (i.e, $k \in \mathcal{R}([i])$ if $A_{ki} = 1$). More generally, $\mathcal{R}(\mathcal{A})$ is the set of resources required for some activity in the set $\mathcal{A} \subseteq J$: $k \in \mathcal{R}(\mathcal{A})$ if $k \in \mathcal{R}([i])$ for some $i \in A$. To avoid trivialities we assume that each resource is required for at least one activity so that $\mathcal{R}(J) = K$. We let $\mathcal{S}(i, j)$ be the set of resources shared by activities $i$ and $j$: $\mathcal{S}(i, j) = \mathcal{R}([i]) \cap \mathcal{R}([j])$.

Remember the traditional notions of utilization and capacity: the average workload arriving into each queue per unit of time is $m \ast \lambda$, which we call the activity load vector $\rho^a$. Summing over the relevant activities yields the resource utilization vector $\rho$:

$$\rho^a(\lambda) = m \ast \lambda \text{ and } \rho(\lambda) = A\rho^a(\lambda).$$

The bottleneck resources are those with the highest utilization, denoted by $\rho^{BN}(\lambda) = \max_{k \in K} \rho_k(\lambda)$. It is useful to formulate this as a linear program, called the static planning problem (SPP):

$$\rho^{BN}(\lambda) = \min_{\rho \in \mathbb{R}_+} \{ \rho : A\rho^a(\lambda) \leq \rho e \}. \quad \text{(SPP)}$$

If $\rho^{BN}(\lambda) \leq 1$, then $\rho^{BN}$ is the fraction of time the bottlenecks are busy processing. The set of SPP-feasible activity load vectors $x = m \ast \lambda$ is the polyhedron

$$\mathcal{P} = \{ x \in \mathbb{R}_{+}^J : Ax \leq e \}.\quad \text{(2)}$$

Traditional networks can typically handle arrival rates $\lambda$ in the interior of $\mathcal{P}$ (i.e., with $\rho^{BN}(\lambda) < 1$): there is a control policy that makes the network stable with queues that remain finite in expectation. With collaboration and multitasking, this is not necessarily true because resources do not work in isolation.
Rather than merely requiring that each individual resource is not overloaded, we must require that the network as a whole is not overloaded while preventing any processing conflicts. The resource-activity matrix $A$ captures these processing conflicts—activities that cannot be performed simultaneously because they share resources. Based on these we can construct configuration vectors which specify which activities are being executed: A feasible configuration vector $v$ is a binary non-zero $J$-vector such that $v_i = v_j = 1$ if activities $i$ and $j$ do not share resources ($S(i, j) = \emptyset$) and can thus be performed simultaneously. The number of feasible configurations is at least as the number of activities $J$. Indeed, each activity can be performed in isolation which means that the unit vectors in $\mathbb{R}^J$ are natural feasible configuration vectors. In the absence of collaboration and multitasking, all activities can be performed simultaneously and all $2^J$ binary $J$-vectors are feasible configurations.

Notice that a binary vector $v$ is a feasible configuration if and only if $\sum_i A_{ki} v_i \leq 1$ for each resource $k$. In particular, the feasible configuration vectors are the integer vertices of the polyhedron $P$. Let $C$ be the matrix of these integer vertices; i.e., each column of the matrix $C$ is a feasible configuration vector. (Of course, $AC \leq e_K^J e'$. With some abuse of notation we write $a \in C$ when $a$ is a column of $C$.

To keep up with the average workload $m \ast \lambda$ arriving into each queue per unit of time, the corresponding average amount of time the network must be processing per unit of time—the network utilization—is given by the solution to the following linear program

$$\rho_{\text{net}}(\lambda) = \min_{\pi \in \mathbb{R}^J_+} e' \pi$$

$$\text{s.t. } C\pi = m \ast \lambda,$$

(SPPC)

where $\pi_c$ is interpreted as the long-run average allocation to configuration $c$. Denote by $\bar{\pi}$ the optimal time allocation in this Static Planning Program for Collaboration (SPPC). The network capacity is the set of throughput vectors $\lambda \geq 0$ for which $\rho_{\text{net}}(\lambda) = 1$, which implies that the network is fully utilized.

Notice that feasible activity load vectors $m \ast \lambda$ are a convex combination of the configuration vectors, which is a subset of $P$ (which also can have non-integer vertices). Indeed, $AC \pi^* \leq e'_K e' J \pi^* = \rho_{\text{net}} e_K$ so that $A m \ast \lambda \leq \rho_{\text{net}} e$ and $\rho_{\text{net}}$ is a feasible solution to (SPP). Hence, $\rho_{\text{net}} \geq \rho^\text{BN}$ meaning that the network must work longer than the bottleneck resources (which may not be able to work in parallel because of processing conflicts). In GVM, we called this gap:

Unavoidable Bottleneck Idleness $\text{UBI}(\lambda) = \rho_{\text{net}}(\lambda) - \rho^\text{BN}(\lambda) \geq 0$. (3)
An important question is when does collaboration reduce capacity and inflict positive UBI? The mathematical answer is simple: Theorem 4.2. in GVM states that $\rho_{\text{net}}(\lambda) = \rho_{\text{BN}}(\lambda)$ and $\text{UBI}(\lambda) = 0$ for any parameters $m, \lambda$, or any other probabilistic assumptions, if and only if the polyhedron $P$ is integral. Indeed, only then do the configuration vectors span the feasible region of the (SPP). Notice that the essential condition involves matrix $A$ and is independent of other parameters. To gain structural insight, we presented a graph representation of the matrix $A$ as a formal tool to characterize properties of what we call the collaboration architecture. The graph has a node for each activity and an edge between two nodes if their resource sets overlap. Our main result in GVM (Theorem 4.4) was that a “nested” collaboration architecture features no UBI. We refer the reader to GVM for the formal (general) definition of nestedness.

A useful implication, relevant for this paper, is that a nested architecture can be represented by a graph where nodes are arranged in levels $i = 1, \ldots, d$. Two nodes at the same level do not share resources and if one traverses a path $r = i_0, \ldots, i_\ell$ from level 1 to a node on the lowest level (with no level visited twice) then $S(i_m, i_\ell) \subseteq S(i_k, i_\ell)$ for all $m < k < \ell$. Figure 6 shows a simple example while GVM presents an algorithm for constructing such a leveled graph from the adjacency matrix $A$. Notice that in Figure 6 activities 3 and 4 do not share resources. Activities 1 and 3 share resource 1 and so does activity 2 that is on a path from a3 to a1. Similarly, a1 and a4 share resource 2 which is also used by a2 that is on a path from a4 to a1.

Hierarchical architectures are a special subset of nested architectures that we will exploit later.
in this paper. They have a leveled graph where each path from top to bottom has at least one resource shared by all activities on that path. Figure 6 and the BC network have such an hierarchy.

Parallel networks, as shown in Figure 7, allow us to isolate the effect of collaboration and multitasking yet are still surprisingly subtle and challenging to control. The only dependence between activities is through possibly overlapping resources and there are two extremes: If each activity $i$ uses a dedicated resource $i$, then we have $J$ independent single-class single-server systems. In contrast, if all activities share the same resources $R(1) = R(2) = \ldots = R(J)$ we recover one multiclass $M/M/s$ queuing system with $s = R(1)$ resources. In general, collaboration and multitasking fall in between these two extremes: some activities share some resources ($R(i)$ and $R(j)$ may overlap) and some resources process multiple activities.

The base network is the simplest parallel network with collaboration and multi-tasking: it has one collaborative activity, labeled activity 0, that requires all $K$ resources and the $K$ individual activities each requiring one distinct resource. Thus each resource $k$ is needed for collaborative task 0 and for its individual task $j$: $R(0) = K$ and $R(j) = j$ for $j \neq 0$. With only two resources ($K = 2$) we arrive at the BC network in Figure 1.

4. Dynamics of network workload $W^{\text{net}}$

A control rule specifies which configuration vector is active at each time $t$. Let $T(t)$ be the cumulative allocation (vector) process. Its component $T_c(t)$ is the cumulative amount of time a configuration vector $c$ is active during $[0,t]$:

$$T_c(t) = \int_0^t \mathbb{1}\{\text{configuration vector } c \text{ is active at time } \tau\} \, d\tau,$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function. Given that only one configuration can be active at any point in time, the total time the network is processing during $[0,t]$ is $\sum_c T_c(t) = e'T(t) \leq t$. The total time the network is idle (and no configuration is active) during $[0,t]$ is $I^{\text{net}}(t) = t - e'T(t)$.

To translate queue workload and activity load into network workload and network utilization we first abstract from detailed dynamics and consider a long-run average time allocation vector $\pi$.
and use the configuration matrix $C$. For a constant time-allocation vector $\pi$, the total amount of time activity $i$ is processed equals $\langle C\pi \rangle_i$. We define the network workload $W_{\text{net}}(Q)$ embodied in the queue length vector $Q$ as the minimal expected time needed to empty the network:

$$W_{\text{net}}(Q) = \min_{\pi \in \mathbb{R}_+^J} e'\pi$$

s.t. $C\pi \geq m \ast Q$.  

(4)

For the BC and BC+ networks of Figure 1 this yields the intuitive expressions:

$$W_{\text{BC}}(Q) = m_1Q_1 + \max\{m_2Q_2, m_3Q_3\} \quad \text{and} \quad W_{\text{BC+}}(Q) = m_1Q_1 + m_2Q_2 + m_3Q_3.$$  

(5)

It is instructive to consider the dual linear program to (4):

$$\max_{y \in \mathbb{R}_+^I} y'(m \ast Q)$$

s.t. $y'C \leq e'$.  

(6)

Letting $y^*(Q)$ denote the optimal solution, strong duality then gives us a simple deconstruction of network workload directly in terms of queue workload:

$$W_{\text{net}}(Q) = \sum_j y^*_j(Q)m_jQ_j.$$  

(7)

There are two extreme deconstructions depending on $A$ or $C$: When $A$ is the identity matrix (networks without collaboration or multitasking), then $W_{\text{net}}(Q) = \max_j(m_jQ_j)$ and $y^*$ is a piece-wise linear function of $Q$. When $C$ is the identity matrix (as in the BC+ networks), then $W_{\text{net}}(Q) = \sum_j(m_jQ_j)$ and $y^* = e$ is the constant vector of ones, indicating that resources perform as a single resource. In general, the dual variables of the instantaneous network workload depend on $Q$.

A long run average view simplifies these dual variables. Consider a network with initial queue vector $Q(0)$ and arrival process $A(t)$, which denotes the number of arrivals during $[0, t]$. Recall that $T_i(t)$ is the cumulative amount of time configuration vector $i$ is active during $[0, t]$ so that:

$$Q(t) = Q(0) + A(t) - S \circ CT(t),$$  

(8)

where $S \circ T(t)$ is the vector with components $S_j(T_j(t))$. Dividing both sides by $t$, letting $t \to \infty$ and recalling the strong law that $A_j(t)/t \to \lambda_j$ and $S_j(t)/t \to 1/m_j$, then in a stable network, we must have

$$C\pi = m \ast \lambda \quad \text{and} \quad \pi = \lim_{t \to \infty} \frac{T(t)}{t}.$$  

(9)
As shown earlier, the optimal time-allocation rates \( \bar{\pi} \) satisfy SPPC. If \( \bar{\pi}_k > 0 \), we say that configuration vector \( k \) is optimal; otherwise it is a suboptimal configuration. We label the configurations to allow the following block partitioning

\[
\bar{\pi} = [\pi^*, 0] \quad \text{and} \quad C = [C^*, C^0],
\]

where \( \pi^* > 0 \) and \( C^* \) contains all optimal configuration vectors.

Let \( \bar{y} \) henceforth denote a static dual variable \( y^*(\lambda) \). By strong duality \( \sum_j \bar{y}_j m_j \lambda_j = \rho_{\text{net}} \) and complementary slackness further yields that \( \bar{y}' C_{-k} = 1 \) if the \( k \)th column is an optimal configuration, meaning \( \bar{\pi}_k > 0 \). In matrix notation, recalling (10), we have

\[
\bar{y}' C^* = e'.
\]

Let the 0 configuration mean that no resource is processing (the network idles). Then,

\[
\bar{y}' C \bar{\pi} = \bar{y}' C^* \pi^* = e' \pi^* = 1 - \bar{\pi}_0,
\]

where \( \bar{\pi}_0 \) is the optimal idleness rate (the fraction of time the zero configuration is used). In fact, the dual variables \( \bar{y} \) provide us with the simplification we were looking for: it can be shown that in heavy traffic as \( \rho_{\text{net}} \to 1 \), then \( y^*(\hat{Q}^\text{net}_\rho) \to \bar{y} \) using the typical scaling \( \hat{Q}^\text{net}_\rho(t) = (1 - \rho_{\text{net}}) Q^\text{net}_\rho((1 - \rho_{\text{net}})^{-2} t) \), provided that one uses a control that is “well behaved” in fluid scale. This means that

\[
W^\text{net}(\hat{Q}^\text{net}_\rho(t)) = \sum_j \bar{y}_j (\hat{Q}^\text{BN}_\rho(t)) m_j \hat{Q}^\text{BN}_j(t) \approx \sum_j \bar{y}_j m_j \hat{Q}^\text{BN}_j(t).
\]

Henceforth, the network workload process refers to

\[
W^\text{net}(t) = \sum_j \bar{y}_j m_j Q_j(t).
\]

Essentially, \( W^\text{net}(t) \) is the expected time the network needs to process the queue vector \( Q(t) \) while using the steady-state optimal time-allocations \( \bar{\pi} \). Combining (8) and (13) we have that

\[
W^\text{net}(t) = W^\text{net}(0) + \sum_j \bar{y}_j m_j A_j(t) - \sum_j \bar{y}_j m_j S_j((CT)_j(t))
\]

\[
= W^\text{net}(0) + \sum_j \bar{y}_j m_j \lambda_j t - \bar{y}' CT(t) + M(t),
\]

where \( M \) is the deviation from the fluid (first moment) processes:

\[
M(t) = \sum_j \bar{y}_j m_j \left( A_j(t) - \lambda_j t \right) + \sum_j \bar{y}_j m_j \left( \mu_j((CT)_j(t) - S_j((CT)_j(t) \right).
\]
Similar to [10], block partition the control vector as

\[ T(t) = [T^*(t); T^0(t)] \]

where \( T^* \) denotes the allocation to all optimal configurations and \( T^0 \) to the suboptimal configurations. Complementary slackness then yields

\[ \bar{y}'CT(t) = e'T^*(t) + \bar{y}'C^0T^0(t). \]

Recall that \( e'T(t) = t - I_{\text{net}}(t) \) so that

\[ \bar{y}'CT(t) = t - I_{\text{net}}(t) - (e' - \bar{y}'C^0)T^0(t). \]

Strong duality yields that \( \sum_j \bar{y}_j m_j \lambda_j = \rho_{\text{net}} \) so that we arrive at

\[ W_{\text{net}}(t) = W_{\text{net}}(0) - (1 - \rho_{\text{net}})t + (e' - \bar{y}'C^0)T^0(t) + M(t) + I_{\text{net}}(t). \]

The usage of suboptimal configurations increases the network workload. Conversely, for networks that have no suboptimal configurations (as in the BC+ network where \( C = C^* \) is the identity matrix) the “damaging term” \( (e' - \bar{y}'C^0)T^0(t) = 0 \) and equation (15) reduces to the single server equation. In general, equation (15), together with the fact that \( e' - \bar{y}'C^0 \geq 0 \), allows us to state the following result.

**Theorem 1** If \( \rho_{\text{net}} = \rho_{\text{BN}} + UBI > 1 \) then the network is transient under any policy, i.e.,

\[ \liminf_{t \to \infty} \frac{W_{\text{net}}(t)}{t} > 0. \]

5. Coordination and Switching Idleness

We have just established that “inefficiency” is captured by the suboptimal-configurations term \( (e' - \bar{y}'C^0)T^0(t) \). Suboptimal configurations may be used for several reasons: when resources have conflicting priorities (coordination idleness), when we cannot preempt service to switch to an optimal configuration (switching idleness) or when the only available work is in suboptimal configurations (availability idleness).

Switching idleness is incurred when activating a resource—putting an idle resource to work in an optimal configuration—does not necessitate the movement of any other resource. This happens when there is an available optimal configuration that is a superset of the currently used configuration. Consider, for example, the BC network (Fig. 1) when \( Q_0 = 0, Q_1 > 0, Q_2 > 0 \). Suppose that
we give non-preemptive priority to the collaborative task and that at time $t$ there is an arrival to an empty collaborative queue while both resources are working in their individual tasks (the configuration used at $t$ is $(0,1,1)$). If resource 1 completes service first, it must wait for resource 2 to complete its service before switching to the collaborative task, even though resource 1 still has work in queue 1: we use suboptimal configuration $(0,0,1)$ while the optimal configuration $(0,1,1)$ is still available. Hence, the term *switching idleness*.

In what follows, given $q \in \mathbb{R}^J_+$ and a configuration vector $b$, we write $q_b$ for the sub-vector that has $q_i$ for $i$ such that $b_i = 1$. Thus, $Q(t)$ is the vector of queues corresponding to activities that are active under configuration $b$.

**Definition 1 (Switching Idleness)** For suboptimal configuration $a_k = C_k^0$, let

$$I_k^s(t) = \int_0^t \mathbb{1}\{S^*(a,Q(t)) \neq \emptyset\}dT_k^0(t),$$

where $S^*(a,q) := \{b \in C^* : b > a, q_b > 0\}$. The network’s switching idleness is given by $I_k^s(t) = (e' - \bar{y}'C^0)I^s(t)$.

Coordination idleness is incurred when putting an idle resource to work necessitates moving other resources. When we use a suboptimal configuration $a$ while there exists an optimal configuration $b$ that would utilize more resources but would require moving a resource from an activity it is currently processing. Consider, for example, the BC network (Fig. 1) at a time when $Q_0 > 0, Q_1 = 0, Q_2 > 0$. If we prioritize the individual activities, then we would use suboptimal configuration $(0,0,1)$ where only resource 2 is processing while the optimal configuration $(1,0,0)$ would utilize both resources. Yet that configuration would require moving resource 2 from activity 2 that it is currently processing. In that case, resource 1 is incurring coordination idleness.

Using the exclusive-or operator $\otimes$, this means that $b \otimes a \neq 0$ and neither $b > a$ nor $a > b$. We can now define:

**Definition 2 (Coordination Idleness)** For suboptimal configuration $a_k = C_k^0$, let

$$I_k^c(t) = \int_0^t \mathbb{1}\{S^*(a_k,Q(t)) = \emptyset \text{ and } X^*(a_k,Q(t)) \neq \emptyset\}dT_k^c(s),$$

where

$$X^*(a_k,q) := \{b \in C^* : b \otimes a_k \neq 0, q_b > 0 \text{ and neither } b > a_k \text{ nor } a_k > b\}.$$  

The network’s coordination idleness is given by $I_k^c(t) = (e' - \bar{y}'C^0)I^c(t)$.

\footnote{For two binary vectors $a$ and $b$, $a \otimes b = a + b$ modulo 2.}
The indicator sets for coordination and switching idleness are disjoint so that $I^c_k(t) + I^s_k(t) \leq T^0_k(t)$. Availability idleness refers to the remaining time that the suboptimal configuration $k$ is used while not incurring coordination or switching idleness:

**Definition 3 (Availability Idleness)** For suboptimal configuration $a_k = C^0_k$ let

$$I^a_k(t) := T^0_k(t) - I^c_k(t) - I^s_k(t).$$

The network’s availability idleness is given by $I^a_k(t) = (e' - \bar{y}'C^0)I^a(t)$.

Availability idleness is the cumulative time that suboptimal configuration $a_k$ is used while there is no optimal configuration available that is not a subset of $a_k$. In the symmetric BC network, $I^a_k(t)$ increases when only one individual queue has work but the other two queues are empty. Such events are sufficiently rare that availability idleness does not amount to a capacity loss in the BC network or more generally:

**Theorem 2** Consider a parallel network with a nested collaboration architecture and a policy that, when the only configurations available are sub-optimal, prefers a configuration that utilizes a bottleneck resource. Then, given $\bar{y} \in \mathcal{Y}$, it holds that, almost surely,

$$\limsup_{t \to \infty} \frac{1}{t} I^a_k(t) \leq 1 - \rho^{BN}.$$

In contrast to availability idleness, coordination and switching idleness do lead to capacity losses as we will later show. Theorem 2 is easy to verify in the symmetric BC network: $\lambda_1 m_1 = \lambda_2 m_2$ and $(1, 1/2, 1/2)$ is an optimal $\bar{y}$. The time that the suboptimal configuration $(0, 1, 0)$ is used while no other optimal configuration ($(1, 0, 0)$ or $(0, 1, 1)$) is available is bounded from above by the amount of time that resource 2 has no work anywhere in the network. This is bounded by the time that the resource would have no work if it could work in isolation, which in turn is bounded by $1 - \rho^{BN}$.

Denoting $U^{UBI}(t) = (\rho^{net} - \rho^{BN})t$, we can capture all four types of collaboration idleness in (15):

$$W^{net}(t) = W^{net}(0) - (1 - \rho^{BN})t + M(t) + I^{UBI}(t) + I^c_k(t) + I^s_k(t) + I^a_k(t),$$

(18)

### 6. Maximal Throughput and Optimal Scaling

We now study the stability, maximal throughput and queue scaling of collaborative networks. The impact of simultaneous collaboration constraints depends on whether tasks can be preempted or not and we organize our results accordingly.
6.1. Preemptive Policies

Consider a fairly general family of preemptive policies specified as follows: At any time $t$ the policy chooses a feasible configuration $a$ that maximizes an instantaneous value function $V(a, Q(t))$ where $Q(t)$ is the queue length vector at time $t$. That is, we choose

$$a^*(t) \in \arg \max_{a \in C} V(a, Q(t)).$$

(19)

The dynamic rule (19) is combinatorial—it requires selecting a configuration out of the set of all feasible configuration vectors. In general, one cannot replace (19) with the continuous version

$$x^*(t) \in \arg \max_{x \in P} V(x, Q(t)),$$

(20)

using the polyhedron $P$ defined in (2), as the latter can choose fractional allocations $x^*(t)$ that are infeasible for indivisible resources. The following is a corollary of the analysis in GVM and proves that both formulations are equivalent for networks with nested collaboration architecture.

**Theorem 3** For a network with a nested collaboration architecture, each extreme point $x^*$ of $P$ is a feasible configuration vector, i.e., $x^* \in C$. In turn, if $V(x, y)$ is concave in $x$ for each $y$ then the maximum values of (19) and (20) are equal.

This has a direct implication to throughput maximization. Dai and Lin (2005) prove that resource-splitting, preemptive maximum pressure policies, introduced by Tassiulas and Ephremides (1992), maximize throughput in open networks (here we use the term to refer to networks where each activity has a single buffer and routing is probabilistic). Theorem 3 together with Dai and Lin (2005, Theorems 5 and 6) yields that a non-resource-splitting preemptive maximum pressure policy achieves the theoretical throughput and stabilizes the network for each $\rho_{BN} < 1$.

**Corollary 1 (Maximum pressure maximizes throughput)** Consider an open network with a nested collaboration architecture and indivisible resources. Then, for any $\rho_{BN} < 1$, the preemptive maximum pressure policy maximizes throughput, i.e, $Q(t)/t \to 0$ almost surely as $t \to \infty$.

In parallel networks the maximum pressure policy maximizes the instantaneous value function

$$V(x, Q(t)) = \sum_i \mu_i x_i Q_i(t).$$

(21)

If the collaboration architecture is nested, the optimal configuration vector $x^*$ is integral so that the maximum pressure policy chooses the configuration vector $c$ with highest $\sum_{i: c_i=1} \mu_i Q_i(t)$, which
generalizes a longest queue policy. Indeed, in the symmetric base network, the maximum pressure policy serves the collaborative queue 0 if $Q_0 > \sum_{i \neq 0} Q_i$, and the individual queues otherwise.

For the remainder of the paper, whenever we take $\rho^{\text{BN}} \to 1$ we assume that no task shrinks and disappears along the sequence of arrival rate vectors $\lambda^{\rho^{\text{BN}}}$; i.e., the sequence $\lambda^{\rho^{\text{BN}}} \to \bar{\lambda} > 0$.

**Theorem 4 (Maximum pressure in parallel networks)** In a parallel network with a nested collaboration architecture, the preemptive maximum pressure policy achieves the optimal scaling:

$$\limsup_{\rho^{\text{BN}} \to 1} (1 - \rho^{\text{BN}}) \mathbb{E}Q^{\rho^{\text{BN}}} (\infty) < \infty. \quad (22)$$

The maximum pressure policy requires frequent switching of resources. This, as we will later show, introduces significant switching idleness in a non-preemptive version of the policy. We propose, instead, an alternative family of preemptive policies that also achieve optimal scaling in nested networks and whose non-preemptive version will remain useful. We call this family *nested priority policies* because they are grounded in the collaboration architecture.

Recall that we can organize the nodes in levels for a network with a hierarchical collaboration architecture. We define nested priorities as follows: At any time $t$, resource $k$ is assigned, amongst its activities with positive queues, to the one at the highest level. In particular, a resource will work in level $l$ only if the queues of all its activities at higher levels $1, \ldots, l - 1$ are empty. In the base network, for example, the nested priority policy preemptively prioritizes the collaborative activity.

**Theorem 5 (Nested priorities)** Consider a parallel network with hierarchical collaboration architecture. A preemptive nested priorities policy stabilizes the network for any $\rho^{\text{BN}} < 1$, achieves optimal scaling, and keeps the asymptotic workload only at the activities at the lowest level $l$ of the collaboration graph:

$$\limsup_{\rho^{\text{BN}} \to 1} (1 - \rho^{\text{BN}}) \mathbb{E}Q^{\rho^{\text{BN}}}_l (\infty) < \infty \quad \text{and} \quad \limsup_{\rho^{\text{BN}} \to 1} (1 - \rho^{\text{BN}}) \mathbb{E}Q^{\rho^{\text{BN}}}_{i<l} (\infty) = 0.$$

### 6.2. Non-preemptive Policies

Preemptive and non-preemptive service perform fundamentally differently in networks where indivisible resources collaborate, even in heavy-traffic. Indeed, in the base network, a non-preemptive version of the maximum pressure policy looses its desirable performance. Figure \[\] demonstrates the capacity loss due to increasing switching idleness as throughput increases. In the symmetric
base network, the non-preemptive maximum pressure policy becomes unstable around $\rho \simeq 0.86$ because switching idleness then consumes the entire “idleness budget” $1 - \rho$. (Coordination and availability idleness pose no problem.) The policy thus no longer maximizes throughput, let alone achieve optimal scaling. (Notice that the M/M/1 queue is much smaller.) In fact, there exist no general results for non-preemptive maximum pressure policies in networks where resources process multiple activities.

To build some insight into the challenges of switching we focus first on the base network. We then proceed to present a non-preemptive extension of the nested priorities policy that, as we prove, maximizes throughput and achieves the optimal scaling. To minimize switching idleness it is natural to consider a polling policy. In the base network, a polling policy serves the collaborative queue until exhaustion (i.e., until $Q_0 = 0$), after which resources switch to their individual tasks and serve those until all individual queues are empty before moving back to the collaborative queue. In addition to minimizing switching idleness, polling is a decentralized policy that also allows a natural human discretion: people switch tasks only when their current task is exhausted. While polling also maximizes throughput, polling leads to extreme queue oscillations (Fig. 9) so that queues do not scale optimally (i.e., faster than $(1 - \rho_{BN})^{-1}$):

**Theorem 6 (Polling)** Non-preemptive polling stabilizes the base network for any $\rho_{BN} < 1$. If there are two or more asymptotic bottlenecks, i.e., $(1 - \rho_j^{BN}) = (1 - \rho_{BN}) + o(1 - \rho_{BN})$ for at least two resources, then the scaling rate is super linear in $(1 - \rho_{BN})^{-1}$:

$$\liminf_{\rho_{BN} \to 1} (1 - \rho_{BN})^{3/2} E Q_{+}(\infty) > 0.$$
The culprit behind the superlinear scaling is coordination idleness stemming from the requirement that all individual queues be empty before moving back to the collaborative queue (Fig. 10). The total time the individual queues are served during one cycle is the largest hitting time of zero among all individual queues (Fig. 9). There is no switching idleness here as moving to the collaborative task happens only when all individual queues are empty. Thus, while maximum pressure leads to instability due to frequent switching, polling (trying to avoid switching cost altogether) incurs excess coordination idleness that retains stability yet leads to significant queues.

We seek an intermediate solution that trades off coordination and switching idleness. Polling variants like gated polling or follow-the-leader where the first resource to drain its individual queue moves to the collaborative queue and acts as the leader while other resources follow upon completion of their current service are potential candidates. Based on nested architectures, we next present another variant of polling that provides a mechanism to manage this tradeoff and achieve optimal scaling while providing more control on the size of the collaborative queue. A central result in §7, however, will be that there exists no policy that provides full control of all queues.

**S-Nested Priorities in the Base Network**

We first define the policy in the base network and later extend to general parallel networks. Consider the following non-preemptive version of the nested priorities policies in the base network with added threshold $S$: When resources are serving the individual queues, an alarm sounds when the
Figure 10  Coordination idleness under polling leads to extreme queue oscillations and suboptimal scaling in the base network.

collaborative queue hits level $S$. Resources move to the collaborative queue as soon as possible (those that are idle immediately, the others upon completion of their current service) and serve the collaborative queue to exhaustion before moving back to the individual queues.

This “$S$-nested priorities” policy retains the benefits of polling by avoiding excessive switching yet bounds the switching idleness per cycle by the “switching time” $T^s$, which is the time from the alarm sounding until all resources have moved to the collaborative task. With iid exponential service times, $T^s$ is the maximum of $J$ service times whose expectation is

$$m_1 \log(1 + J) < \mathbb{E}T^s = m_1 \sum_{j=1}^J \frac{1}{j} < m_1 (1 + \log J),$$

so that the expected switching idleness per cycle per resource $\mathbb{E}T^s - m_1$ is bounded by $m_1 \log J$.

Of course, $T^s$ depends on the policy: If we require a predetermined sequence in which resources stop and move to the collaborative queue (e.g., cheapest resources idles first), then $T^s$ is the sum of $J$ service times and $\mathbb{E}T^s \leq m_1 J$. For any sequencing rule, we can divide the switching idleness by the length of a cycle (see Appendix) to yield the average switching idleness rate for resource $j$:

$$\frac{\lambda_j m_j (\mathbb{E}T^s - m_j)}{\lambda_0 + m_j}.$$ 

A smaller threshold $S$ yields a smaller collaborative queue but requires more switching which reduces the maximal throughput. Accounting for the switching idleness yields an explicit stability condition and, moreover, choosing threshold $S$ smartly can achieve optimal scaling:
Theorem 7 The base network is stable under the S-nested priorities policy provided that
\[ \rho_j + \frac{\lambda_j m_j (\mathbb{E} T^s - m_j)}{S_0 + m_j} < 1 \] for all resources \( j = 1, \ldots, J \).
Moreover, there exists \( K \) such that with \( S = K (1 - \rho^{BN})^{-1} \) the policy maximizes throughput and achieves optimal scaling:
\[ \limsup_{\rho^{BN} \to 1} (1 - \rho^{BN}) \mathbb{E} Q_0^{BN} (\infty) < \infty \]

The theorem reflects the fact that, while \( S \) determines the average collaborative queue, the ratio \( \lambda_0/S \) determines the switching frequency. As \( \rho^{BN} \) increases we must reduce the switching frequency to preserve stability. The consequent increase of \( S \) trades off switching and coordination idleness as shown in Figure 11. For small values of \( S \), the servers switch too frequently and hence switching idleness dominates. For large values of \( S \) switching is less frequent but coordination idleness dominates due to the increase in instances where one individual queue is empty but the resources are not working on the collaborative task although it has a queue.

Given that the sample paths of the collaborative queue are roughly triangular, one expects the average collaborative queue length \( \mathbb{E} Q_0 \) to be roughly \( S/2 \). The following confirms this intuition.

Lemma 1 Consider the base network under the S-nested priority policy with sufficiently large \( S \) to guarantee stability. Then the expected collaborative queue \( Q_0 \) is bounded as follows:
\[ \mathbb{E} Q_0^{BN} (\infty) \leq \frac{S}{2} + \lambda_0 (1 + \frac{1}{2S}) \mathbb{E} T^s + \frac{\lambda_0^2}{2S} \mathbb{E} (T^s)^2. \]
If we set $S = K(1 - \rho_{BN})^{-1}$ the bound of the lemma\(^3\) becomes:

$$E Q_0^{BN}(\infty) \leq \frac{S}{2} + \lambda_0(1 + \frac{1}{2S})E T^s + \frac{\lambda_0^2}{2S}E(T^s)^2 \approx \frac{S}{2} + \lambda_0 E T^s.$$ 

### S-Nested Priorities in Parallel Networks

The BC network is the simplest instance of hierarchical collaboration architectures. The generalization to parallel networks with hierarchical collaboration architectures uses the ordering of nodes into a leveled graph (recall Figure 6). We write $i \preceq j$ if there is a path in the collaboration graph between $i$ and $j$ and $i$ is at a lower level than $j$. Then define the S-nested priorities policy as:

(i) Upon completing service in queue $i$, a resource moves to the highest-level activity $j$ for which $i \preceq j$ and $Q_j \geq S_j$, and serves that queue.

(ii) If no queue is found in (i), continue serving the current queue $i$ if it is non-empty.

(iii) If both (i) and (ii) fail, serve the highest-level activity $j \preceq i$ that has a nonempty queue.

(iv) If all (i)-(iii) fail, move to the highest-level activity $j \preceq i$ for which $i \preceq j$ and $Q_j > 0$; if no such $j$ exists, move to highest-level activity 1. (Of course, whether or not the resource can work in that queue depends on other resources as well.)

Notice that this policy prescribes actions to resources rather than to the network as a whole. The fact that it does not introduce conflicts stems from its grounding in the hierarchical collaboration architecture which guarantees that when a resource $k$ moves to an activity $j$ so does, upon service completion, any other resource $i$ required for that activity.

One may worry that too much switching is being introduced because some collaborative queues may be left before they are exhausted. Yet that worry is unsubstantiated because we can bound the switching idleness. Effectively, the highest level queues will be served to exhaustion before moving to lower level queues. That means that for thresholds that grow as $(1 - \rho_{BN})^{-1}$ it will take roughly the same amount of time $O((1 - \rho_{BN})^{-1})$ until switching. That guarantees that the switching idleness is at most $O(1 - \rho_{BN})$. To formalize this, define the cumulative workload of queues at levels below $j$ and the total workload as:

$$W_j(t) = \sum_{i \preceq j} m_i Q_i(t) \text{ and } W_+(t) = \sum_{i \in I} m_i Q_i(t).$$ \hspace{1cm} (25)

The following theorem nicely relates the nested architecture to the control prescriptions.

\(^3\) The appendix also presents bounds on the individual queues by relating them to single-server queues with vacations.
Theorem 8 Consider a parallel network with a hierarchical collaboration architecture. The $S$-nested policy stabilizes the network for all $S_j$ large enough. Moreover, there exist $K_j$ such that with $S_j = K_j(1 - \rho^{BN})^{-1}$ the policy maximizes throughput and achieves optimal scaling:

$$\limsup_{\rho^{BN} \to 1} (1 - \rho^{BN})E_{W^*_{\rho^{BN}}}(<\infty) < \infty,$$

and for each non-leaf node $j$,

$$\limsup_{\rho^{BN} \to 1} (1 - \rho^{BN})E_{W^*_{\rho^{BN}}}(<\infty) \leq \sum_{i \leq j} m_i K_i. \quad (27)$$

7. Controllability

Now we turn our attention from maximizing throughput to controlling queues in parallel networks. We shall show the building blocks behind the key insight that coordination or switching idleness typically implies that controllability comes at the expense of a capacity loss (recall Fig. 4 on p. 7).

7.1. I-Priority: Prioritizing the Individual Activities

In the following tradeoff results we focus on the BC network.

Lemma 2 (P-Priority to both individual tasks) With preemptive priority to all individual tasks, the base network is stable (positive recurrent) if and only if

$$\frac{\lambda_0}{\mu_0} < \prod_{i=1}^{2} \left(1 - \frac{\lambda_i}{\mu_i}\right).$$

For the numerical example shown in Fig. 4 for the symmetric BC network with $m_i = 0.5$, the maximal throughput $\lambda_i = \bar{\lambda}$ under preemptive priority solves $\bar{\lambda}/2 = (1 - \bar{\lambda}/2)^2$ with solution $\bar{\lambda} = 3 - \sqrt{5} \simeq 0.764$. This maximal $\rho^{BN} = (m_0 + m_1)\bar{\lambda} = 0.764$ represents a capacity loss of 23.8%.

Non-preemptive priority to the collaborative task regains some capacity but not all:

Lemma 3 (NP-Priority to both individual tasks) With non-preemptive priority to both individual tasks, the BC network is unstable if

$$\frac{\lambda_0}{\mu_0} > \left(1 - \frac{\lambda_1}{\mu_1}\right) \left(1 - \frac{\lambda_2}{\mu_2}\right) \left[\frac{1 + m_0(\lambda_1 + \lambda_2)}{1 + m_0(\lambda_1 + \lambda_2) \left(1 - \frac{\lambda_1}{\mu_1}\right) \left(1 - \frac{\lambda_2}{\mu_2}\right)}\right].$$
Figure 12  Coordination idleness under preemptive priority to both individual tasks leads to a capacity loss.

Notice that the right hand side of the sufficient condition is strictly less than 1 meaning that we cannot take $\rho^{\text{BN}}$ close to 1 without making the system explode. Indeed, since $(1 - \frac{\lambda_1}{\mu_1})(1 - \frac{\lambda_2}{\mu_2}) < 1$, we have (with strict inequality) that

$$\left[\frac{1 + m_0(\lambda_1 + \lambda_2)}{1 + m_0(\lambda_1 + \lambda_2)} \frac{(1 - \frac{\lambda_1}{\mu_1})(1 - \frac{\lambda_2}{\mu_2})}{1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}{1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}}^{-1}}\right] < \left(1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}{1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}{1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}{1 - \frac{\lambda_1}{\mu_1} \frac{1 - \frac{\lambda_2}{\mu_2}}}^{-1}}\right)^{-1}.$$

These findings, and the simulation results in Fig. 12, are explained by coordination idleness: nonpreemptive priority completes each collaborative service and thus has smaller coordination idleness than preemptive priority. Notice that neither policy idles a non-empty individual queue and thus none incurs switching idleness. In addition, for $\rho^{\text{BN}}$ sufficiently close to 1, the collaborative queue, when not being served, grows very fast so that the availability idleness $I_k^a(t)/t \to 0$ for either suboptimal configuration $k$. Therefore, the entire capacity loss is due to coordination idleness: the moments in time where only one resource works in an individual task although the collaborative queue has work.

The essential insight is that making all individual queues small entails a loss of throughput. Alternatively, to maximize throughput one must give up on minimizing all individual queues. This stark trade-off holds for any stabilizing policy:

**Theorem 9 (Priority to individual tasks trade-off)** Consider the BC network with both

---

4 One can derive a necessary, but complex, condition for stability using hitting times of a two-MM1-queue system.
resources being bottlenecks (i.e., $1 - \rho_i^{BN} = 1 - \rho^{BN} + o(1 - \rho^{BN})$). Then, any stabilizing sequence of policies, Preemptive or Non-Preemptive, has at least one large individual queue $i$ such that

$$\lim\inf_{\rho^{BN} \to 1} (1 - \rho^{BN}) E Q_i^{\rho^{BN}} (\infty) > 0. \quad (28)$$

The next theorem shows that there is a difference between preemptive and non-preemptive.

**Theorem 10 (I-Priority: Preemptive vs. Non-Preemptive)** Consider the BC network with both resources being bottlenecks. Then, any sequence of non-preemptive stabilizing policies has (28) for both individual queues. There exists, however, a sequence of preemptive stabilizing policies such that, for only one queue $j \in \{1, 2\}$

$$\lim\sup_{\rho^{BN} \to 1} (1 - \rho^{BN}) E Q_j^{\rho^{BN}} (\infty) = 0,$$

and such that for any $\epsilon > 0$

$$\lim\sup_{\rho^{BN} \to 1} (1 - \rho^{BN})^{1+\epsilon} E Q_j^{\rho^{BN}} (\infty) = 0.$$ 

This last result shows that coordination idleness prevents the preemptive prioritization of both individual queues without loosing capacity but allows for the preemptive prioritization of one individual queue while retaining full capacity. In contrast, additional switching idleness prevents the non-preemptive prioritization of even one individual queue.

### 7.2. C-Priority: Prioritizing the Collaborative Activity

Theorem 5 proved that preemptive nested priorities to collaborative tasks “empties” the collaborative queues and guarantees stability and optimal scaling. Indeed, these policies incur no switching or coordination idleness because they operate as a collection of single server queues.

Non-preemptive policies that prioritize the collaborative activity are a special case of our $S$-nested priorities policies with $S = 1$. Theorem 7 gives a sufficient stability condition that shows that a small fixed size of the threshold $S$ leads to a capacity loss. The culprit is the excessive switching idleness incurred by minimizing the collaborative queue. (Note that this policy does not incur any coordination idleness.)

The essential insight is that making the collaborative queues small under non-preemption entails a loss of throughput. Alternatively, to maximize throughput one must give up on minimizing the collaborative queue. This stark trade-off holds for any stabilizing non-preemptive policy:
Theorem 11 (C-NP Priority)  Consider the BC network with both resources being bottlenecks (i.e., $1 - \rho_{i}^{BN} = 1 - \rho^{BN} + o(1 - \rho^{BN})$). Then, any sequence of stabilizing non-preemptive policies has a non-negligible collaborative queue:

$$\lim \inf_{\rho^{BN} \to 1} (1 - \rho^{BN})E_{0}^{\rho^{BN}}(\infty) > 0.$$ 

8. Concluding Remarks

Networks with simultaneous collaboration by multiple types of multitasking human or indivisible resources present challenges to capacity management and queue control. We introduced and formally defined two types of idleness—coordination and switching—to explain and quantify these challenges. When prioritizing certain queues over others, different policies may introduce different amounts of coordination and switching idleness; see Table 2. Coordination and switching idleness can each destabilize the network: coordination destabilizes under $I^{2}$-P and switching under C-NP. Both types of idleness can also have more subtle impact: e.g., the coordination idleness under polling is not large enough to destabilize the network but does lead to exceedingly large queues.

Good non-preemptive policies must trade-off coordination and switching idleness. Limiting switching idleness means, however, giving up on queue control. This is the stark tradeoff between capacity and queue control. The decentralized S-policy we presented achieves stability and optimal scaling by balancing coordination and switching idleness. It captures explicitly the tradeoff shown in Fig. 13: increasing the threshold decreases the capacity loss but increases the average collaborative queue.

Nested architectures are central to our results and are relevant to network design. Nested architectures facilitate the decentralized coordination of resources to keep collaborative queues under control (through thresholds) without significantly compromising the total network performance.
The optimal control of networks with collaboration and multitasking indivisible resources remains an interesting open challenge. One would want to characterize the structure of optimal policies that minimize specific delay costs. Our results contribute to this future pursuit by highlighting the new limitations that these intriguing networks add relative to traditional queueing networks. Of importance for dynamic control in heavy-traffic are the facts that:

1. Preemption and non-preemption are fundamentally different. Certain prioritizations are impossible under either. But what is achievable under preemptive policies may not be achievable, not even asymptotically, under non-preemptive policies (recall Theorem 10).

2. Non-negligible thresholds are necessary to limit the switching idleness. This is similar to the work on queues with exogenous switchover times \cite{ReimanWein1998}. Switchover times here are, however, endogenous which makes control more difficult.

3. Nested architectures simplify resource coordination and give simple answers when the costs are aligned with the hierarchy defined by the levels in the collaboration tree. For example, the preemptive nested priorities policy may minimize linear holding costs if activities at higher levels in the tree incur higher cost.
References


