Packet scheduling with optional client privacy

Andrew Beams
abeams@cis.upenn.edu
University of Pennsylvania

Sampath Kannan
kannan@cis.upenn.edu
University of Pennsylvania

Sebastian Angel
sebastian.angel@cis.upenn.edu
University of Pennsylvania
Microsoft Research

ABSTRACT

Existing network switches implement scheduling disciplines such as FIFO or deficit round robin that provide good utilization or fairness across flows, but do so at the expense of leaking a variety of information via timing side channels. To address this privacy breach, we propose a new scheduling mechanism for switches called *indifferent-first scheduling* (IFS). A salient aspect of IFS is that it provides privacy (a notion of strong isolation) to clients that opt-in, while preserving the (good) performance and utilization of FIFO or round robin for clients that are satisfied with the status quo. Such a hybrid scheduling mechanism addresses the main drawback of prior proposals such as *time-division multiple access* (TDMA) that provide strong isolation at the cost of low utilization and increased packet latency for all clients. We identify limitations of modern programmable switches which inhibit an implementation of IFS without compromising its privacy guarantees, and show that a version of IFS with full security can be implemented at line rate in the recently proposed *push-in-first-out* (PIFO) queuing architecture.

ACM Reference Format:
Andrew Beams, Sampath Kannan, and Sebastian Angel. 2021. Packet scheduling with optional client privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3460120.3485371

1 INTRODUCTION

Networks, from roads to the Internet, are a scarce resource shared by all. Sharing is necessary, as the complexity and cost of having dedicated links or infrastructure between every pair of clients would be unimaginable. But as we have known for decades in a variety of contexts [9, 10, 16, 37, 51, 58, 60], sharing—and specifically the lack of strong isolation—is at odds with privacy. Today's networks rely on switches and routers that queue and schedule packets following policies such as first-in-first-out (FIFO) and priority queuing. These policies have many desirable properties ranging from fairness to minimizing average latency, but lack of interference between clients is not one of them. As a result, a client's traffic is influenced and shaped by others' traffic.

Why is this a problem? Consider situations in which there are multiple clients that share a network switch: multi-tenant data centers, corporate networks, universities, coffee shops, someone's home. In these settings, a client ("the victim") can be in a position where it sends or receives messages via a shared switch, while another client ("the attacker") is also using the same switch and observing how the victim's traffic affects its own traffic (if at all) by the way the switch schedules and queues packets. In other words, the attacker aims to exploit a *timing side channel* that leaks information about the victim's traffic. Prior works [29, 33, 35, 36] have shown that this flavor of side channel can reveal which Web sites a user is visiting or what words are being spoken over a VoIP application such as Skype or Zoom (even when the communication is encrypted [15, 27, 45, 57]). This leakage can also be used by data center tenants as a covert channel to bypass existing isolation and monitoring mechanisms [28].

Note that timing side channels in networks are far from new: the anonymity community has bravely fought them for decades in their quest to build onion routing and mix network systems [11, 14, 17–20, 22, 23, 25, 40–43, 56]. What is different in our setting is the threat model: our concern is not a malicious network provider or nation state actor that tries to deanonymize users. Instead, the focus is on what one user can learn about another when the network infrastructure is reliable and trustworthy. Not only is this a qualitatively different threat model, it is in many ways a more common one: a visitor at one's home could perform measurements to eavesdrop on a VoIP call taking place in a private room, or a tenant of a public data center might attempt to infer workload characteristics of a competitor with whom it shares a switch.

This paper's contribution. Our work has three goals. First, we wish to understand if timing side channels are exploitable today. Prior works [29, 33, 36] offer evidence of these attacks in simulations or Internet measurements on slow (297 kbps) DSL routers, but it is unclear whether those observations hold with fast gigabit switches. We replicate the results of Kadloor et al. [33] when the victim and the attacker share a traditional WiFi home router. However, conducting these attacks on a fast data center switch requires more effort on the part of the attacker. Nevertheless, we demonstrate the feasibility of leaking some information with fast switches.

Our second goal is to design a scheduler that provably guarantees privacy, which is a notion of strong isolation across clients. While there is already one scheduling discipline that provides this guarantee, namely *time division multiple access* (TDMA) and its randomized and weighted generalization [36] in which clients are allocated a window of time on which to send their packets, it has several drawbacks. Chief among them is that TDMA taxes *all* clients, in the sense that even clients who are indifferent about privacy must still pay the cost of using TDMA. Not only is this bad for privacy-indifferent clients, it also bad for the collective, as TDMA is not work conserving and wastes bandwidth when there are idle clients.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3485371

To address these drawbacks, we introduce a new hybrid scheduling discipline called *indifferent-first scheduling* (IFS). The key aspect of IFS is that clients who satisfied with the status quo and do not require privacy (e.g., tenants in a data center who are not running sensitive workloads) should continue to receive as good a service (or even better) than that provided by existing schedulers such as FIFO. On the other hand, clients who require privacy guarantees can opt into IFS's private mode and avoid leaking any information through the scheduler's decisions, at the cost of increased latency for their packets. Furthermore, IFS lets clients toggle between indifferent and private modes (e.g., a client may engage private mode when it starts a VoIP call). While transitions can be observed by an attacker and might leak the user's intent to be private, they do not leak the user's workload characteristics.

Our last goal is pragmatic. We ask to what extent we can implement privacy-preserving scheduling disciplines on programmable switches. We find that neither TDMA nor IFS are amenable to implementation in existing architectures, since, among other limitations, switches do not support pauses or random sampling. If we look at existing Intel Tofino switches, for example, the best we could manage is to provide privacy to client's *outgoing packets* (e.g., a client's request to an HTTP server leaks no information, but the corresponding response might). This is problematic since responses can leak just as much or even more information than requests. However, we show that a recently proposed queuing architecture for programmable switches called *push-in-first-out* (PIFO) [55] has all the building blocks that we need to build IFS and TDMA. We implement both of these schedulers on a PIFO simulator [4] and show that IFS achieves the best of both worlds: it provides better expected packet latency than FIFO or round robin to indifferent clients, and the same privacy guarantees and better latency than TDMA for private clients.

In summary, this work makes the following contributions:

- We replicate prior timing attacks on recent hardware and show that some leakage exists even on fast switches.
- We propose IFS, a new scheduling discipline that guarantees privacy to clients who want it without burdening those who do not, and which has many desirable properties.
- We show how to instantiate IFS in switches that support *push-in-first-out* (PIFO) [54, 55].
- We evaluate our implementation of IFS and find that its performance is better than existing schedulers for both indifferent and private clients, while simultaneously protecting private clients from timing side channels.

2 MOTIVATION AND RELATED WORK

This section discusses proposed attacks on schedulers and prior proposals to address the resulting privacy violations.

2.1 Timing attack on switches and schedulers

Our work is inspired by the observation of Kadloor et al. [33] that if a client is accessing content on the Internet while traversing a switch or router that uses a first-in-first-out queuing strategy, an adversary could issue a series of probes to this switch to determine when the victim client is sending packets (and their size). The high level idea is that the switch will enqueue the attacker's probes and will process them once it has spare cycles (presumably after it has processed any packets from the victim that arrived before the attacker's). The probes could be simple ICMP packets (though some switches treat ICMP traffic differently), but could also be TCP or UDP packets sent to a destination that the attacker controls and that ensures the attacker's traffic traverses the shared switch. Based on how long it takes for the attacker's probes to be processed, the attacker can infer the number of packets sent by the victim. This information can allow the attacker to learn which Web sites or services the victim is accessing, or even what phrases are spoken over VoIP calls, even if the traffic is encrypted [15, 27, 45, 57]. This attack can also be conducted within a data center thereby allowing a tenant to infer the workload characteristics of another tenant that uses the same network infrastructure. Figure 1 depicts this attack.

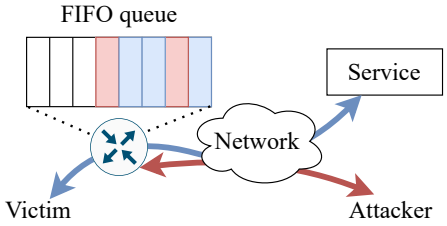


Figure 1: An attacker can learn whether the Victim is sending packets to some service (and potentially which service) by probing one of the switches used by the victim. Since the switch has limited resources it must queue the attacker's packets whenever there is contention. If the switch uses a FIFO queuing discipline and the attacker's packets arrive after the victim's, the attacker can observe changes in timing and infer that the victim is sending packets and the size of the burst of traffic. This attack was proposed by Kadloor et al. [33].

Two similar attacks include the work of Gong and Kiyavash [29] that shows that information leaks when users share a job event queue, and the work of Ghassami and Kiyavash [28] that shows how to create a covert channel between two otherwise isolated processes in a data center. In this latter work, even if the processes are given their own dedicated hardware, if the underlying physical network is shared, then one process can send a covert message to the other via the same strategy described above. The sender could encode their message by modifying the sizes or timing of seemingly innocuous Web traffic, after which the recipient could send probes to the shared switch to retrieve it. This might not trigger any red flags in a firewall or other monitoring system. In this way, a malicious actor could take advantage of this covert channel to leak potentially sensitive data.

The above attacks are actually not limited to FIFO: by using higher frequency probes, this same technique can be used on any work conserving scheduler [29, 35]. This motivates the need for scheduling mechanisms that protect against these types of side channels while still prioritizing other standard metrics such as fairness, low latency, etc.

In Section 7.1, we replicate the experiments performed by Kadloor et al. [33], first with a typical home router, and then with a state-of-the-art switch. Our findings are consistent with those of

prior work for the home router, but the high performance of a gigabit switch makes this attack more difficult to carry out. Nevertheless, we show that leakage is still present.

2.2 Existing proposals

In this section we describe prior work on building a scheduler that provides privacy across requests. Note that these proposals were not introduced in the context of packet switches, and are therefore hard to implement in our setting. They also force all clients to have privacy and pay for it even if some clients are indifferent and could do without it. Nevertheless, they illustrate the kinds of techniques that have been proposed to prevent the attacks mentioned in the prior section.

The work of Kadloor et al. [36] proposes two solutions. The solutions assume that clients send requests that, on a long enough timescale, follow a well-known distribution (e.g., Poisson distribution with a certain rate). This distribution is assumed to not be sensitive. What the attacker does not know, however, is the instantaneous number of requests being issued by a victim in some arbitrary time window. Such fine visibility into a victim's workload could leak what services a client is accessing, as we described previously.

The first proposal, which the authors denote *accumulate-and-serve*, works by alternating between two phases. In the accumulation phase, the scheduler serves no requests, it merely enqueues them. After some time passes, the queued requests are serviced in FIFO order. As requests are being serviced, the scheduler begins to accumulate the next batch. The intuition behind this approach is that as the accumulation window is lengthened, the number of requests from each client approaches the number that would be expected based on their long-term rates from the well-known distribution, and hence the amount of useful information for the adversary decreases.

The second approach is a non-work-conserving variant of the classic *Time Division Multiple Access* (TDMA) protocol. In (non-work-conserving) TDMA, each client is assigned a specific time slot; if a client has a pending packet by the time the scheduler reaches that client's time slot, the client's packet is processed. Otherwise, the scheduler simply idles until the following time slot when it processes the packet (if any) of the next client. Since the delays of a client are independent of the traffic of other clients (as clients have their own statically allocated slots), an adversary's probes reveal no information about any other client in the system. Kadloor et al. [36] then generalize TDMA to include weights and randomize the allocation of slots (*proportional TDMA*).

An entirely different approach to address a related problem is given in Pacer [48], which prevents network side channels in a shared data center environment. One major distinction with Pacer is the location of the privacy mechanism, and its scope of coverage. Pacer's isolation mechanism is located at a server, and provides privacy to clients' responses from that specific server under two assumptions: (1) that the size and shape of a client's requests leak no information (necessary since Pacer does not touch clients' requests); and (2) that the server has a small set of responses which it pre-registers with Pacer (this requires modifying the server). These assumptions are reasonable in some settings. For example, within a data center where client virtual machines access a few file servers that host a specific set of files. However, our setting is more general: we support clients that make arbitrary requests to arbitrary services, without requiring any server modifications.

3 DEFINING PRIVACY

In this section we formalize what it means for a scheduling algorithm to provide privacy. At a high level, our definition of privacy captures a notion of strong isolation among all clients, in the sense that one client should not be able to affect the behavior of the packets of another client. Prior privacy definitions captured this with notions such as *mutual information* [33, 35], *correlation* [34], and *minimum mean squared error* [35, 36]. We instead give an *indistiguishability*-based definition in Section 3.2 that resembles more traditional cryptographic notions such as semantic security or pseudorandomness. We believe this definition is easier to understand. We begin with a concrete setting and threat model.

3.1 Setting and threat model

In order to give our formal definition, we abstract away the details of the switch and network topology, and treat each switch as a scheduler. Furthermore, we make the simplifying assumption that clients acquire a certain *rate* λ from the switch's operator (e.g., 10 Mbps), and that the switch is provisioned to support this rate. Clients' instantaneous number of packets, however, may be sampled from any distribution with expected value of λ (this allows clients to idle or be bursty). What does it mean for a client to acquire a rate of λ? In WAN settings, it means that clients purchase a dedicated rate from their ISP, similar to existing service tiers but with stronger SLOs. In a data center network, this means that the operator provisions the network in a way that ensures that each tenant (or their VMs) can achieve their purchased rate; there is already a vast literature on performance isolation and throughput guarantees [8, 12, 13, 44, 50] that considers this exact setup (but note that privacy is a stronger notion than the type of isolation studied in these works).

Setting and admission control. Clients can join and leave the system. When a client c_i joins, it requests a rate λ_{c_i} from the scheduler. The rate could be given in a standard metric such as bits per second, but for simplicity we assume that all λ_{c_i} have been normalized by the capacity of the scheduler, and are therefore a real number in $[0, 1]$. Each scheduler serves some set of *active* clients C, and this set changes over time as clients join and leave the system. The aggregate rate of active clients at the scheduler is $\Lambda = \sum_{c_i \in C} \lambda_{c_i}$. In order to reason analytically about the worst-case delay induced by our proposed scheduling mechanism (§4.4), our scheduler will maintain an *admission threshold* L such that $\Lambda \leq L \leq 1$. We will call a client $c_j \notin C$ *admissible* if $\lambda_{c_j} + \sum_{c_i \in C} \lambda_{c_i} \leq L$. A client can join the system only if it is admissible; otherwise, the client must wait until resources are freed up.

For our analysis we take time to be discrete, and assume that all requests issued to the scheduler are of uniform size, and that the scheduler can process one such request per time slot (we relax these assumptions later).

Threat model. We assume that the network infrastructure is itself honest. Indeed, our model differs from timing attacks on mix networks and anonymity systems in that we do not view the network or provider as the adversary, but rather an ally who is attempting to provide privacy to its users. The adversary in our setting controls any subset of the clients and wishes to learn about one or more victim clients' requests through a timing side channel attack (§2.1). We allow the adversary to adaptively issue requests from its compromised clients at any instantaneous rate it wishes, and to accurately measure the sending and receiving time of all of its packets. Our privacy guarantee ensures that the adversary learns nothing about the traffic of clients who are not compromised.

3.2 Indistinguishability of arrival sequences

For the setting we consider, there is already one definition of privacy in the literature [29] based on the mutual information between an adversary's observations and the times at which the victim transmits. This definition is cumbersome to work with because one must condition on what the adversary already knows (for example, the rate of each victim, the concrete scheduling policy, etc.) and then compute if there is non-zero mutual information. We propose an alternate definition below that is simpler and does not require making the adversary's prior knowledge explicit.

Let us focus on a particular client c (our analysis is symmetric for any client). At each time step t, there are n_t packets belonging to c that arrive at the queue. If the discretization is fine enough, we could imagine that n_t is 0 or 1, but more generally we assume that n_t is a non-negative integer. An *arrival sequence* S^T for the packets of client c is a sequence of integers $(n_0, n_1, \ldots, n_{T-1})$ that denotes the number of packets that arrive at the queue at all times less than T.

Definition 3.1 (Privacy). Let $\mathcal{A}$ be an adversary who controls all but one of the active clients using the shared scheduler. We will say that $\mathcal{A}$ violates the privacy of the remaining client c, if there exist any two arrival sequences S_1^T and S_2^T for c such that $\mathcal{A}$ can create arrival times for packets for the clients that it controls, observe their arrival and transmission times, and be able to correctly decide if c has arrival sequence S_1^T or S_2^T with probability higher than a random guess. Letting O denote the set of observable variables for $\mathcal{A}$ and letting $\mathcal{A}(O)$ denote $\mathcal{A}$'s guess of c's arrival sequence, we say that privacy is violated if:

$$\Pr[\mathcal{A}(O) = i \mid c \text{ has arrival sequence } S_i^T] > \frac{1}{2}$$

where $i \in \{1, 2\}$ and the probability is over the random coins of $\mathcal{A}$ and the scheduling mechanism. Note that this definition is given in terms of the conditional probability of guessing correctly, which avoids having to reason about the prior distribution of different arrival sequences. It is also a definition of statistical indistinguishability, as it is not based on any computational hardness assumptions.

Summary: Our definition basically states that a switch guarantees privacy if an adversary, by injecting any number of packets into the switch at the time slots of its choosing, cannot distinguish between two possible arrival sequences for a victim client's packets. This definition is very strong, essentially stating that the adversary gets no benefit from probing the switch, thereby eliminating all timing side channels.

3.3 Prior approaches guarantee privacy

Both TDMA and p-TDMA meet our definition of privacy (§2.2). To see why, observe that in these schemes the slots assigned to c are independent of c's arrival sequence—they are statically or randomly allocated. Moreover these slots are either used by c or "wasted" if c has nothing to transmit. To other clients these two scenarios are identical since that slot is never made available to them. Consequently, an adversary cannot distinguish between any two arrival sequences for c.

4 MAKING PRIVACY OPTIONAL

TDMA and p-TDMA provide the strong isolation that is needed to prevent the timing side channel discussed in the previous section. However, they force all clients—even those that do not care about privacy—to incur higher response times than they would under other disciplines. In the following subsections we introduce the idea of *indifferent clients*, which are clients that do not care about leaking some or all of their information (essentially the status quo). We then propose *indifferent-first scheduling* (IFS), a new scheduling discipline that provides the same guarantees as TDMA and p-TDMA for clients who desire privacy, without increasing the expected delay for indifferent clients.

4.1 Indifferent-first scheduling (IFS)

The high level idea of IFS is to process the packets of indifferent clients with a work-conserving scheduler, and to give priority to these packets over the packets of private clients.

In detail, let $C = P \cup I$ partition the clients into private and indifferent, respectively. Let $I_r \subseteq I$ be the subset of indifferent clients with at least one packet still queued during time slot r. For each time slot, IFS first determines if the slot will be given to a private or an indifferent client. If $I_r \neq \emptyset$ (i.e., there are packets from an indifferent client in the queue), the slot will be allotted to an indifferent client. To decide which client is serviced, IFS uses a work-conserving scheduling policy; in this work we focus on round robin and FIFO. For round robin, IFS picks a client from I_r to service at random using clients' purchased rates as weights; IFS then dequeues the first packet from this client. For FIFO, IFS processes packets from I_r in the order they entered the FIFO queue.

If there are no packets from indifferent clients in the queue, IFS then considers private clients. IFS picks a client randomly among P—which includes all private clients, even those with no packets enqueued—using their purchased rates as weights. If the chosen client has packets enqueued, the first packet of that client is dequeued and sent. Otherwise, IFS idles until the slot is finished, thereby wasting the switch's resources. This wastage is precisely the price that private clients must pay for privacy (it does not affect indifferent clients since they always "go first"). Figure 2 gives IFS's algorithm.

While conceptually simple, implementing IFS in a real switch is far from trivial since neither randomized round robin (weighted or otherwise) nor idling are supported by programmable switches. In Section 5 we propose approximations and adaptations of this design

```
function IFS(I, P, r)
  if I_r ≠ ∅ then
    c ← WCS(I_r)
  else
    c ← RandomSelect(P)
  SendPacket(c)
```

Figure 2: Pseudocode for indifferent-first scheduling. I is the set of indifferent clients, and P is the set of private clients. $I_r \subseteq I$ is the set of indifferent clients with packets in the queue as of time slot r. $RandomSelect(\cdot)$ chooses a client from the set randomly, weighted by clients' rates. WCS is any work-conserving scheduling discipline; we focus on FIFO and randomized Round Robin ($RandomSelect$). $SendPacket(\cdot)$ sends the next packet queued for client c, if any, or idles.

to conform to the reality of today's switches. Below we discuss the properties of IFS.

4.2 IFS guarantees privacy

IFS guarantees privacy (Definition 3.1) for private clients. The argument mirrors that of TDMA (§3.3): the slots allocated to a private client are independent of the arrival sequence of that client, and depend only on (1) the arrival sequences of the indifferent clients and (2) the internal randomness of the scheduler. Since indifferent clients are afforded no privacy, the ability of an adversary to observe the effect of these clients' packets on its own packets does not give the adversary any information about private clients.

4.3 IFS is incentive-compatible

The addition of privacy, unsurprisingly, increases the time it takes for a packet to be processed. One of our main motivations in designing IFS is to avoid sharing this burden with clients who are indifferent about privacy. IFS actually guarantees that if such clients declare themselves as indifferent, they will be better off (in terms of expected packet delay) than if they declare themselves as private.

We formalize this as follows. Let P and I be the sets of active private and indifferent clients, respectively. Let c be a client that is considering whether to declare itself as private or indifferent. IFS guarantees that for all P and I:

$$D_c(\lambda_c, P, I \cup \{c\}) \leq D_c(\lambda_c, P \cup \{c\}, I)$$

where D_c is the expected delay for c's packets given a rate λ_c, and a set of private and indifferent clients.

We give the proof of this claim in Appendix B. The intuition is that IFS can be viewed as a strict priority queue in which packets from indifferent clients have higher priority than those of private clients. Hence, being an indifferent client results in lower expected packet delay.

4.4 IFS is better for all clients

Since IFS is a scheduling algorithm that provides differentiated service to two types of clients (private and indifferent), it is natural to ask whether clients of either type would have preferred a scheduling algorithm that treats all clients the same as themselves (i.e., either all private if they are private, or all indifferent if they are indifferent). If the answer is no, then this can be seen as a type of *sharing incentive*, meaning that both private and indifferent clients are happy to share the infrastructure and be serviced by IFS. One way to do this is to show that the worst-case expected packet delay under IFS satisfies the following two properties: (1) if a client c is private, then c does worst when all other clients are private and are served by p-TDMA; and (2) if c is indifferent, then c does worst if all other clients are indifferent and are served by a round robin or FIFO scheduler.

The monotonicity definitions below imply these properties.

Definition 4.1 (Indifferent delay monotonicity). Let P and I be non-empty sets of private and indifferent clients, and let $p \in P$ be any private client. A scheduler is *indifferent delay monotonic* if for all indifferent clients $c \in I$ the expected delay for c's packets given rate λ_c is:

$$D_c(\lambda_c, P, I) \leq D_c(\lambda_c, P \setminus \{p\}, I \cup \{p\})$$

That is, changing a client from private to indifferent does not benefit any of the former indifferent clients.

Definition 4.2 (Private delay monotonicity). Let P and I be non-empty sets of private and indifferent clients and let $c \in I$ be any indifferent client. A scheduler is *private delay monotonic* if for all private clients $p \in P$, the expected delay for p's packets assuming p's rate is λ_p is given by:

$$D_p(\lambda_p, P, I) \leq D_p(\lambda_p, P \cup \{c\}, I \setminus \{c\})$$

That is, changing a client from indifferent to private does not benefit any of the existing private clients.

IFS's concrete guarantees. To show that IFS is indifferent delay monotonic (Definition 4.1), recall the following fact from Section 4.1: the packet delay of indifferent clients is only ever impacted by other indifferent clients because indifferent clients have a strict scheduling priority over private clients. As a result, more clients becoming indifferent necessarily hurts existing indifferent clients, as packets from new joiners can sometimes be scheduled first. We give a proof in Appendix C.

Proving that IFS is private delay monotonic (Definition 4.2) is challenging. The difficulty arises from two competing forces whose combined effects are hard to model: (1) the fact that indifferent clients are processed by a work-conserving scheduler and do not waste slots; and (2) the priority that indifferent clients have over private clients. In particular, since indifferent clients never waste slots, if an indifferent client has nothing to send, its slot will be given to another client (potentially a private one). In contrast, when an indifferent client becomes private it will never yield its slot, even when the client has nothing to send. Consequently, a client's transition from indifferent to private partially benefits existing private clients in the sense that there is one fewer client with higher priority, but it also partially harms them because this client will occasionally waste its slot without yielding it to others. Depending on the setting (make up of clients and weights), it is conceivable that one effect might be stronger than the other.

Nevertheless, we conjecture that private delay monotonicity holds for IFS. Appendix A shows empirical evidence in support of it, and Appendix D shows analytic results for several settings. A full proof that reasons about the interplay between the multiple schedulers in IFS remains an open question.

Monotonicity and worst case expected delay. If a scheduler satisfies both private and indifferent delay monotonicity, then the scheduling policy guarantees that there is an upper bound on the expected delay for all clients in all settings. In the context of IFS, this delay is precisely the expected delay of any client c of rate λ_c in a scheduler with an admission threshold of L—using p-TDMA if c is private and round robin or FIFO if c is indifferent. As a result, given the admission threshold supported by the scheduler, the client's rate λ, and whether the client is indifferent or private is enough to bound the worst case expected delay of that client. This holds regardless of any other clients who may enter or leave the system in the future. Appendix E discusses this in more detail.

4.5 Private client starvation

An issue with IFS, as presented, is starvation of private clients since they have lower priority than indifferent clients. As a result, IFS needs to enforce rate limits on clients to ensure that they do not send packets in excess of their allocated rates. This can be done through standard mechanisms such as the use of a token bucket (§6.2). An interesting question is whether private clients also need to be rate limited? After all, the point of IFS's design is that the traffic of a private client does not impact any other private or indifferent client.

We find that if IFS rate limits both private and indifferent clients, then privacy (Defintion 3.1), indifferent incentive (§4.3), and indifferent delay monotonicity (§4.4) continue to hold (the proofs are identical); private delay monotonicity holds if our conjecture holds. The drawback is that IFS would be giving a suboptimal service to private clients. In particular, whenever all indifferent clients idle (or have exhausted their tokens for a given window of time), IFS grants the slot to a private client (the "else" branch in Figure 2). If private clients are also rate limited, occasionally a private client will be chosen by IFS's *RandomSelect* but will have no tokens to send their packet; the scheduler will therefore be forced to idle, thereby wasting the slot and benefiting no one.[1]

One the other hand, if IFS rate limits the indifferent clients but not the private clients, then whenever a private client is chosen by IFS's *RandomSelect* they can send a real packet (if they have one), improving their service. The drawback is that indifferent incentive compatibility (§4.3) no longer holds in a handful of pathological cases. For example, if the switch only has a single client, this client would be rate limited if it were indifferent, but not if it were private; and since it is the only private client, it would receive 100% of the slots under *RandomSelect*. Hence, even if this client did not care about privacy, it would choose to be private to avoid the rate limit.

In the rest of this paper we choose to implement rate limits only for indifferent clients. We conclude that the possibility of improving service for private clients is worth the existence of a few pathological cases where indifferent clients might prefer to label themselves as private. Such mislabeling does not impact the packet delay of other indifferent clients (because they would have a higher priority) or the privacy of other private clients; hence, the arguments against this choice are mostly of theoretical rather than of practical value.

[1] $RandomSelect$ cannot be computed on just the subset of private clients with tokens, nor can the slot be given to another private client because it would leak information: an adversary can specify two arrival sequences that take into account rate limits and that violate Definition 3.1.

5 IFS ON PROGRAMMABLE SWITCHES

At a high-level, implementing IFS on a programmable switch requires four operations: (1) queuing packets with different priorities; (2) idling in response to some condition; (3) selecting randomly among packets; and (4) equalizing packet sizes. Of these features, existing switches provide only the first one. This section describes various ways that allow us to overcome some (though not all) of the missing features. For the remaining missing features, we leverage a switch architecture called PIFO [54, 55] that has attracted significant attention from the networking community, and for which there are preliminary implementation and approximation efforts [3, 5, 7, 59]. Figure 3 shows the high-level architecture of IFS; we discuss each component in the next sections.

5.1 Registration

A client decides whether its traffic should be private or not, and how much rate (upload and download combined) it requires. To do so, it sends a control plane registration packet to the switch containing this information. The switch determines whether it can support this additional bandwidth, and if so, it modifies its queue mapping and weights to take the new client into account. Later, the client may choose to modify its rate, change its type (indifferent or private), or leave the system by sending another control packet (de-registration can also be done automatically after a timeout). If IFS is deployed within a data center, registration packets can be sent by a controller that allocates network capacity to VMs or servers, as in Oktopus [12]. In the WAN context, these packets can be issued by the ISP when a new customer is enrolled. In a coffee shop setting, these packets can be sent by a server when the client authenticates through a WiFi captive portal.

5.2 Emulating switch idling

IFS and TDMA rely on the switch being able to idle for a time slot in the event that a private client's turn is next in the schedule dictated by *RandomSelect* and the client has no packets queued (§4.1). Since switches lack the ability to idle, an alternative is for the switch to inject a *dummy* packet into the head of the queue whenever a private client has nothing to send, and send the dummy instead. While promising, this approach is also not implementable since programmable switches cannot generate packets at line rate whenever a condition holds (e.g., lack of packets from a particular client). Instead, we settle for outsourcing the creation of the dummy packets to private clients, and requiring that they always have a "real" or a dummy packet queued up in order for them to receive privacy. This approach raises two challenges.

Challenge 1: Dummy preemption. How does a client know when to send dummy packets? The easiest option is to send them at frequent intervals, since it is important that either a real or a dummy packet be always available in the queue when IFS selects the client as part of *RandomSelect* (Figure 2). However, this means that if the client sends a real packet to the switch after having sent some dummies (now queued at the switch), the real packet will be processed after all of the dummies are, significantly increasing latency. IFS must therefore have a mechanism to preempt dummy packets.

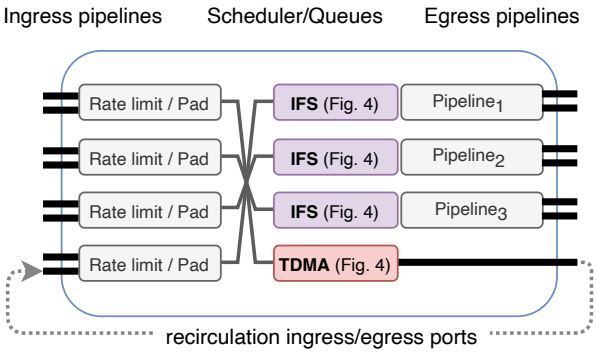


Figure 3: Switch architecture with IFS. Ingress pipelines are used for classifying, rate limiting, and padding packets. Each egress port is associated with an IFS queue (Figure 4), though we only show one IFS queue per pipe. A recirculation egress port is associated with a TDMA or p-TDMA queue and is used to feed back packets that require more padding (§6.3).

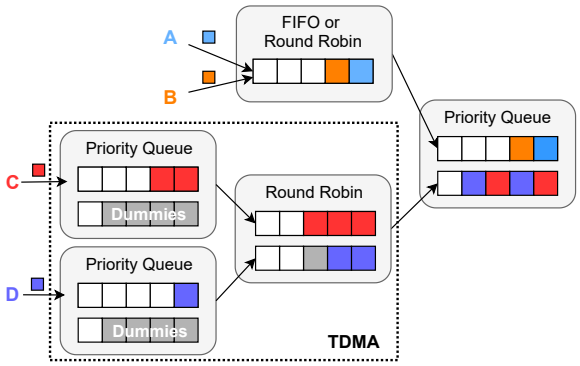


Figure 4: Concrete instantiation of IFS in PIFO [54, 55] for a setting with 2 indifferent clients (A and B) and 2 private clients (C and D). IFS relies on a hierarchy of queues with different scheduling disciplines. The dashed box implements (p-)TDMA, which IFS uses as a sub-component. Indifferent clients retain the status quo and can be serviced with FIFO or round robin.

Challenge 2: Response privacy. If clients only need to send outgoing messages, then sending dummies when they have nothing else to send would mimic the switch idling. However, this is not the case in practice, since clients also receive responses from the services with whom they interact. Worryingly, responses (e.g., the HTML and JavaScript payload in response to an HTTP GET request) can be just as revealing (and often more so) than requests: they tend to be larger and contain more diverse fingerprints. If not handled properly, responses to one private client can impact the responses for another client, creating yet again a timing side channel. This creates a challenge, as the client must somehow mask the absence of responses despite not knowing their size or arrival time a priori. Note that the server with whom the client communicates is completely oblivious to the client's desire for privacy or IFS's mechanisms, and will not send dummies.

Proposal: dummy pools and hierarchical queuing. Our idea to address the above two challenges is to have the switch maintain a pool of dummy packets for each client ready to use in the event that a private client's queue is chosen and has no "real" packets. To preempt dummies (Challenge 1), we implement dummy pools with a priority queue where the lower priority is assigned to dummy packets. This ensures that no dummy packet is ever sent before a queued real packet belonging to the same client. To deal with responses (Challenge 2), the switch filters packets based on source and destination and forwards them to the appropriate queue: all packets destined to a private client share that client's *incoming* queue and dummy pool, and all packets originating from a private client share one of the client's *outgoing* queues and dummy pools. We expand on this in Section 6.2.

Note that the introduction of dummy pools into IFS requires the switch to support a *layered* scheduling policy, as shown in Figure 4. The scheduler will first round robin or do FIFO among indifferent clients (A and B in the figure). If neither client has packets, the switch will round robin among private clients (C and D). For each private client, real packets have priority over dummy packets. Appendix F describes why this approach produces the same observable variables to a probing attacker as a switch capable of idling.

Unfortunately, existing switches lack support for layered policies: they can typically be configured to use a layer of deficit round robin followed by a priority queue, but this is not enough for IFS. We address these issues with PIFO (§6.2).

5.3 Approximate randomized round robin

So far, we have abstracted the *RandomSelect* mechanism of Figure 2 as a "randomized weighted round robin". Since no such scheme is supported by switches, we replace this mechanism with deterministic approximations of *weighted fair queuing* (WFQ) [24, 49]. These approximations work by computing an estimated start and finish time for packets when they arrive, and use these estimates to order packets. Section 6.4 discusses why approximating *RandomSelect* is safe in IFS, but for now we focus on two approximations that we consider useful in different cases.

Approximate Fair Queueing (AFQ) [53]. AFQ maintains a virtual start and finish time for each packet, and orders packets based on ascending virtual finish times. The approximation comes from various concessions, such as the use of count-min sketches due to lack of sufficient memory for per-flow state, dealing with a limited number of queues, and maintaining line rate. AFQ can be used as the round robin approximation for indifferent clients, but it is not appropriate for private clients because it combines different clients' packets in the same queue As a result, a private client's traffic could affect others.

Start Time Fair Queueing (STFQ). The second option is to use a weighted variant of STFQ [30], which schedules packets based on virtual start time. Unlike AFQ, STFQ does not combine the packets of different clients into the same queue, which provides the strong isolation required by private clients at the cost of more queues. And unlike deficit weighted round robin, which is readily available in existing programmable switches owing to its constant-time complexity, STFQ provides a better approximation of WFQ. Furthermore,

STFQ can be implemented at line rate in PIFO switches [54, 55], which we require to guarantee privacy for responses anyway (§5.2).

6 IFS ON PIFO SWITCHES

Push-In-First-Out (PIFO) [54, 55] is an abstraction that aims to support a variety of scheduling policies while still having a design that is implementable in hardware and that operates at line rate. The insight behind this abstraction is that in many policies the ordering of a packet in the queue depends only on some value that is calculated at its ingress. Therefore, once a group of packets are enqueued, their internal ordering does not change. Incoming packets are inserted into a sorted list and then dequeued uniformly from the head.

However, PIFO introduces an additional, perhaps more valuable functionality: the ability to compose multiple queues in a hierarchical fashion. Specifically, the outputs of lower queues can feed into upper ones, which gives designers a lot of flexibility. We exploit this flexibility to design a scheme that supports round robin (STFQ) between private clients while also ensuring that within each client queue, no dummy packet is processed before real packets in the queue. This is the precise hierarchy discussed in Figure 4. Below we discuss the details of implementing TDMA and IFS in PIFO.

6.1 Implementing TDMA

To implement TDMA and p-TDMA in PIFO, clients send dummies as described in Section 5.2. On the switch, we utilize a hierarchy of queues. At the base of this hierarchy, each client will insert its real and dummy packets into its own priority queue, with dummy packets having low priority. These priority queues then feed into a Start Time First Queue (STFQ), which can be implemented in PIFO as described by Sivaraman et al. [55]. The weights used in STFQ will be the rates purchased by each client. This scheme is the dashed box in Figure 4, and to our knowledge, represents the first implementation of a non-work-conserving scheduler (required to guarantee privacy) in a programmable switch—albeit leveraging PIFO and dummy packets.

6.2 Implementing IFS

To implement IFS, we need to introduce support for indifferent clients which adds three constraints: (1) ensure that every indifferent client has a higher priority than all private clients, (2) ensure that packets are enqueued in the appropriate queue, and (3) prevent starvation of private clients.

To address (1), we add a FIFO or round robin queue for indifferent clients. FIFO requires one physical queue for all indifferent clients, whereas for round robin the number of queues needed depends on whether we use AFQ or STFQ; AFQ's fairness guarantees are more approximate but it requires fewer queues. We then add a priority queue that takes as input packets from the above FIFO or round robin queue (high priority), and from the TDMA queue of private clients described in Section 6.1 (low priority). A schematic of this scheme is given in Figure 4.

To address (2), IFS treats packets based on their type:

- *Outgoing*: moving from a client to the upstream network.
- *Incoming*: moving from the upstream network to a client.
- *Internal*: moving from one client of the switch to another.

At ingress, a packet is categorized into one of these types, and then forwarded to the appropriate egress pipeline (based on IP or IP/-port matching). Each egress pipeline has an IFS queue, as depicted in Figure 3. Packets are then inserted into the queue (internal to IFS) associated with the sending or receiving client (depending on whether this is outgoing or incoming packet) as shown in Figure 4. Internal packets are associated with the queue of the client designated as private, if there is only one, or with the queue of the sender if both are private or indifferent. This is safe because privacy implies no leakage beyond what can be inferred in the absence of the switch. In other words, if an internal packet is sent from A to B and enqueued in B's queue, then B may learn something about A's traffic, but B would have learned this regardless.

Finally, to address (3), we implement rate limiting in the ingress pipelines with a token bucket. For each indifferent client, we use a stateful register that tracks the number of tokens and the last time the tokens were refilled, updating this register accordingly, and dropping packets in the absence of enough tokens. We do not rate limit private clients since, by construction, they cannot affect the performance of other private clients (§4.5).

6.3 Dealing with variable-size packets

Since different packet sizes take different amounts of time to be processed and be written on the wire, an adversary can infer packet sizes even with IFS. One solution is to limit egress ports to a rate of MTU/BW per packet, which is the moral equivalent of padding all packets to be MTU sized. For example, for a 1.5 KB MTU and 100 Gbps link, we would limit the port to 120 ns per packet. However, this approach also harms indifferent clients, which IFS aims to avoid. Our approach is to have private clients pad their requests to a uniform size, and implement logic in the switch to pad the responses from upstream services that are unaware of clients' privacy desires. In particular, we implement padding in the ingress pipeline by adding custom Ethernet headers to the *packet header vector* (PHV) until the frame reaches the MTU. This is safe because the switch pads only responses to private clients, who are the next hop and can ignore these headers. One issue is that the PHV has a limited size (vendor specific), so only a limited amount of padding can be added per ingress pipeline. To account for the worst case (padding a small frame to the MTU), the switch might need to *recirculate* small frames (i.e., send them back to an ingress port) a few times.

Typically, switches have recirculation ports that are backed by FIFO queues. In our case, however, this is problematic since FIFO does not guarantee privacy and an attacker can perform timing attacks by controlling an upstream service and issuing small responses that can be delayed by a concurrent victim's small responses. To address this we use a p-TDMA queue with the recirculation port. As with IFS, this p-TDMA queue requires each client to populate and maintain a dummy pool. However, unlike other ports, the client can just populate the dummy pool *once* with a few dummies, and these dummies are automatically recirculated and reused over and over (this is not possible in other ports because there the dummies leave the switch). Figure 3 depicts this process.

6.4 Analysis of IFS's properties

In Section 4 we identified four desirable properties for IFS: privacy, incentive compatibility, and two monotonicities. The use of STFQ to order private packets does not affect our privacy guarantee. Once a private client's actual packets are combined with that client's dummy packets, as long as later scheduling mechanisms do not differentiate between the two, privacy will be preserved. This was not possible without PIFO, as we would otherwise have no way to combine only a private client's packets together with its own dummies, and doing this off the switch would not provide response privacy.

Prioritizing all indifferent packets over private packets provides incentive compatibility. The use of STFQ might, however, affect our monotonicity properties (§4.4), as these are directly related to the "fairness" of the *RandomSelect* approximation. However, we do not observe violations to either of the monotonicity properties in our evaluation (Appendix A).

7 EVALUATION

This section aims to answer the following questions:

- Can the attacks of Section 2.1 be performed against standard home switches and high performance switches?
- Does IFS provide an effective defense?
- What is IFS's performance on private and indifferent clients compared to existing schedulers (FIFO / p-TDMA)?

In addition to the above questions, Appendix A gives empirical evidence in support of IFS's monotonicity properties.

Experimental setup. We conduct our experiments using a Motorola Surfboard ("home switch"), an Edgecore Wedge 100BF with an Intel Tofino programmable chip ("DC switch"), and a PIFO simulator [4]. We use the home switch only for attacks since it is not programmable. For the DC switch, we implement IFS as described in Section 5, without dummy pools (since Tofino lacks the necessary layered scheduler support). We have six servers with 8-core Intel Xeon 4110 CPUs and 100 GB of RAM running Ubuntu 16.04; they connect to the switches with Intel X722 network cards. For the PIFO simulator, we implement all of IFS as described in Section 6.

7.1 Are timing side channels a real threat?

We begin with two simple hypotheses: (1) the attack described in Section 2.1 is possible on today's home and data center hardware, and (2) client-side traffic shaping (where clients send dummies without the use of IFS's other mechanisms) is not enough to provide privacy. To test these hypotheses, we perform the following experiments.

Home switch. We connect two machines to the home switch, with one acting as the victim and the other as the adversary. We configure the adversary to send repeated pings to the switch, with inter-ping delays of 10 ms, and capture all sent and received packets with Wireshark. Meanwhile, the victim runs a headless browser instrumented to visit one of the Alexa Top 50 sites [1] at random and then wait for a random amount of time (up to a minute) before executing again.

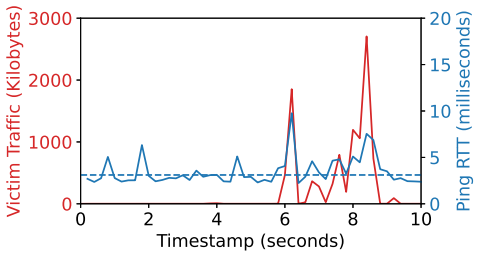


Figure 5: Attack on the Home switch. The victim's Web traffic (red line), has a clear effect on the RTT of the probes sent by the adversary every 10 ms (blue line). The dashed blue line shows the average probe delay over the length of the capture.

Setting	Pearson's r (p-value)	Spearman's ρ (p-value)
Home switch	0.829 (<.00001)	0.854 (<.00001)
DC switch	0.871 (<.00001)	0.636 (.00016)
FIFO simulation	0.519 (<.00001)	0.658 (<.00001)
IFS simulation	0.001 (.96446)	0.007 (.83134)

Figure 6: Correlation coefficients between adversarial ping RTT and victim traffic in a variety of settings. The corresponding two-sided p-values using the Student's t-distribution are given in parenthesis.

Figure 5 gives the result for a representative 10 second snapshot (the rest of our trace looks similar). Whenever the victim is actively accessing a Web page (red line spike), there is a noticeable impact on the adversary's aggregated ping delays (blue line). Furthermore, the additional packet delay is impacted by the amount of bytes fetched by the victim, giving the adversary the ability to infer volume and not just frequency. While measurements can be noisy, the duration and magnitude of packet delay stemming from real victim actions is distinguishable from noise: we compute statistical tests by averaging the RTTs within 100ms intervals and adding the victim's packet sizes within the same intervals. As shown in Figure 6, the Pearson and Spearman correlation coefficients between adversarial ping RTTs and victim traffic are over 0.8 (with both two-sided p-values less than .00001)—implying a strong linear dependence.

DC switch. We then ask whether a similarly simple attack works on our DC switch. This is not the case: the effect of fetching a Web page is simply too small to be noticeable. Consequently, we lower the bar on what constitutes a successful attack and ask instead whether an attacker who controls one or more machines can distinguish whether a victim is sending traffic or idling. This is admittedly less informative to the attacker, but even this binary information can be damaging [9]. To perform this experiment, we connect four servers to the programmable switch. We assign one of the servers to be the "recipient"; there is a 10 Gbps link between the switch and the recipient. One of the servers acts as the victim who communicates with the recipient, and the remaining two servers are controlled by the adversary. We then write a P4 data plane program to forward all packets to egress ports based on destination MAC address.

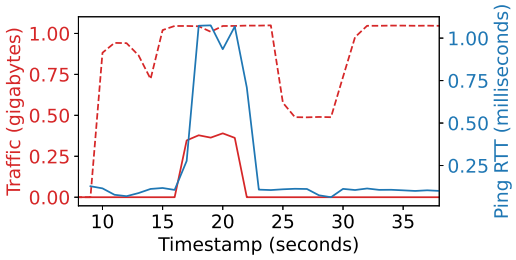


Figure 7: Attack on the DC switch. The adversary can observe the impact of the victim's traffic (red) on its probe's RTTs (blue). The dashed line shows the adversary's traffic needed to prime the attack.

The victim uses iPerf3 [2] to generate traffic, and alternates between sending 3 Gbps of UDP traffic to the recipient and idling. The adversary sends 8 Gbps of traffic from one of its servers to the recipient; this creates congestion at the egress port (toward the recipient) whenever the victim sends traffic. The adversary uses its other server to measure this congestion by pinging the recipient with an inter-ping delay of 10 ms. The results are shown in Figure 7. As with the home switch, we also compute the correlation coefficients between RTTs and victim traffic and give the results in Figure 6.

While this attack requires the victim to generate over 2 Gbps of traffic, an adversary can fine tune, à la binary search, its own contribution of traffic until it observes signs of congestion for victims with fewer traffic. It can also use this approach to get a rough estimate of a victim's volume.

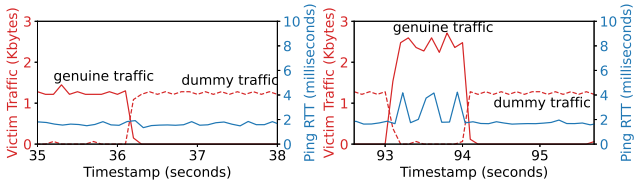
7.2 Client-side Traffic Shaping

Can a concerned victim prevent the above attack without the switch's help with the use of well-timed dummies? To answer this question we consider two cases: (1) victim alternates between sending unidirectional traffic for a few seconds (UDP traffic that does not trigger a response) and then idles; and (2) victim alternates between sending bidirectional traffic for a few seconds (UDP requests that trigger a response) and then idles. Throughout the experiment, the adversary probes the switch every 10 ms. Figure 8a shows the result for case (1) on the home switch. We see that masking idle time with dummy traffic (dashed red line) is indeed effective—observe the blue line, which captures the adversary's observations, remains unchanged as the client idles. This is expected, as dummy packets are indistinguishable from real packets.

Figure 8b shows the result for case (2). Unlike the prior case, the solid red line depicts not only requests but also the contribution of responses. We find that the adversary's observation (blue line) is significantly different when the victim is sending real traffic versus dummy traffic (which does not trigger a response). For the client to mask this discrepancy it would need to have a priori knowledge of the response distribution and timing and somehow fabricate dummy responses. Such a task is untenable in practice.

7.3 Does IFS hide private clients' actions?

As Section 7.2 shows, client-side traffic shaping (i.e., the addition of dummy traffic) on its own is not enough to provide privacy.



(a) Unidirectional traffic **(b) Bidirectional traffic**

Figure 8: With unidirectional traffic a client can get privacy by sending dummy packets in lieu of idling. When traffic is bidirectional, the additional response traffic impacts the adversary's pings (blue line), even with the added dummies. Here the amount of response traffic is the same size as the outgoing requests, but in general the user will have no way of forecasting the size or time of a response.

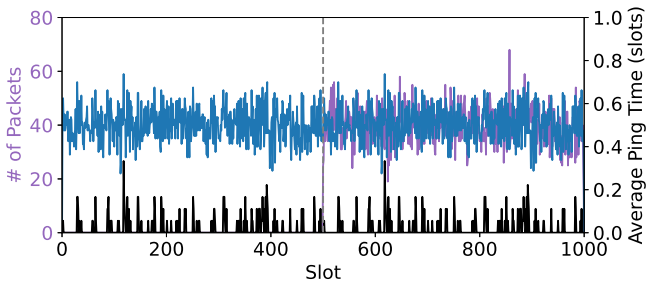
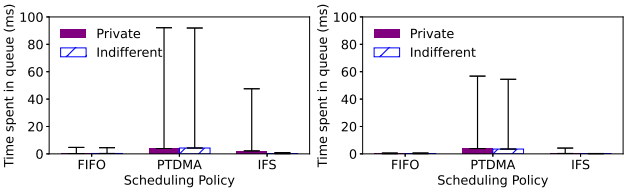


Figure 9: Attack under IFS in a PIFO simulator [4]. The black line shows the delay experienced by adversarial pings (right-side y-axis), whereas the violet and blue lines show the traffic patterns of private and indifferent clients respectively (left-side y-axis). We use identical indifferent traffic on both stages and activate private clients only in the second stage. The black line is identical in both stages indicating that private clients' actions do no affect the adversary's probes.

As a result, it becomes necessary to enlist the switch's help. To that end, we turn our attention to our implementation of IFS in a PIFO simulator [4]. Since indifferent clients can be serviced by any work-conserving scheduler, we use FIFO for them. We have private clients pre-load their dummy pools before our simulations start. In each experiment, the switch has a capacity of 100 packets per time slot and we consider a slot to consist of one phase of insertion and then a corresponding phase of dequeueing (the current PIFO simulator works at the granularity of abstract "packets").

We evaluate a setting with 5 clients: 2 are indifferent, 2 are private, and one is the attacker. We make the attacker a private client as well (if we make the attacker indifferent then its packets will have high priority and will not be affected by the packets of private clients). Clients send packets following a Poisson process with a rate of 20 packets per slot, whereas the attacker carries out the attack described in Section 2.1 by sending 18 ping packets every slot. We simulate two 500-slot stages. In the first stage, we activate only the indifferent clients (the non-adversary private clients remain idle). In the second stage we also activate the non-adversary private clients. To make the results more clear, we fix the random choices made by the indifferent clients to be the same as those of the first



(a) Bursty traffic distribution **(b) Steady traffic distribution**

Figure 10: Average queuing time across client types and schedulers. FIFO treats all clients as indifferent and is work-conserving, whereas p-TDMA treats all clients as private and wastes slots when clients idle. IFS (using FIFO for indifferent clients) achieves lower queuing time than FIFO for indifferent clients and than p-TDMA for private clients. Error bars show the 99^{th}-percentile waiting time.

stage so that the incoming indifferent traffic is identical in both stages.

Figure 9 gives the results. The x-axis gives the simulation's slots, the black line (and associated right-hand y-axis) shows the average waiting time of the adversary's probes, the blue lines give the total number of packets sent by indifferent clients, and the violet lines depict the number of packets sent by private client. As we expect, IFS ensures that the adversary's observations are identical in both stages despite private clients being idle in the first and active in the second. Indeed, the Pearson and Spearman correlation coefficients under IFS are close to 0, with p-values indicating that we cannot disregard the null hypothesis (there is no correlation). An (insecure) FIFO baseline running on the same simulator with the same packet distributions has much higher and statistically significant correlations. The results are in Figure 6.

7.4 How does IFS impact clients?

At the outset, our philosophy was that IFS should preserve the performance of the status quo for indifferent clients, while providing privacy for those who want it at a cost comparable to prior approaches. We evaluate this goal with 3 metrics: latency, throughput, and network overhead for private clients.

Latency. In order to measure the latency impact of IFS on clients we consider two workloads: a bursty workload (e.g., Web browsing) and a steady workload.

Bursty workload: we record 150 5-minute packet traces following a similar idea to Section 7.1: we visit an Alexa top-50, wait a random amount of time, and then visit another site. We implement FIFO, p-TDMA, and IFS on the PIFO simulator configured with an slot interval of 280 μs and 300 Mbps bandwidth (similar to our home switch). We feed the 150 traces (representing 90 private and 60 indifferent clients) into the PIFO simulator, and measure the mean and 99^{th} percentile time that packets of each client type spend in the queue. Figure 10a gives the results.

Since FIFO has no notion of client type and is work conserving, all clients' packets achieve reasonably low latency (although without privacy). On the other hand, p-TDMA treats all clients as private and uses dummy packets that waste a slot whenever a client idles. As a result, clients' packet latency is considerably higher than in FIFO, which highlights the cost of privacy. Finally, IFS is type-aware and processes clients' packets accordingly. Indifferent clients, owing to their high priority, achieve lower latency than they do under FIFO due to the sharing incentive implied by indifferent delay monotonicity (§4.4). And while private clients do worse than indifferent clients due to their low priority, they are still better off than under p-TDMA owing to the sharing incentive implied by IFS's conjectured private delay monotonicity (§4.4). Specifically, since indifferent clients yield their slot whenever they idle, these spare slots are given to private clients, thereby lowering their expected packet delay when compared to p-TDMA.

Steady workload: we generate the same number of packets as the bursty workload, but following a Poisson distribution with exponential packet size. The results are shown in Figure 10b and are similar to those of the bursty experiment—the main difference is that all clients enjoy lower latency in this scenario, which is expected from an arrival pattern with lower variance [39].

Overall, the above experiments demonstrate that IFS can indeed benefit all types of clients: private clients get privacy, albeit at a cost, whereas indifferent clients satisfied with the status quo can do even better than they do today!

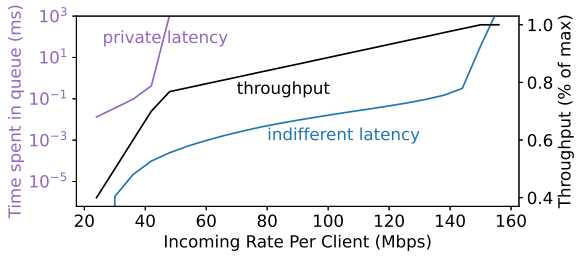
Network overhead. We also measure the additional bandwidth required by the private client for both padding and dummy packets for one of the 5-minute traces. The original workload contained 19.83 MB of data; padding increases the network communication to 38.37 MB. Furthermore, submitting the dummies required by IFS introduces an additional 51.08 MB of network communication.

Throughput. To measure how IFS responds to increasing load, we carry out several experiments that tease out the effect of clients' sending rates on private and indifferent client latency and throughput, and the effect of rate limits. We use the same settings and traces for the PIFO simulator as the previous experiment, which are similar to our home switch. A key distinction is that we map the 150 traces to 5 larger clients (rather than dealing with 150 clients) to increase the range of sending rates for each client.

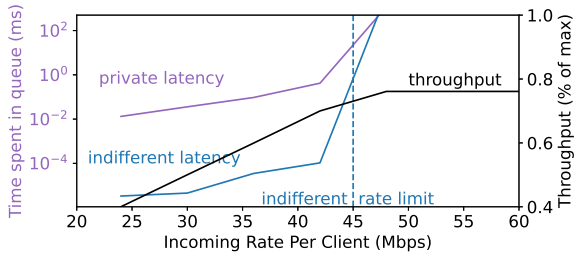
The experiments are as follows. The first experiment sets the sending rate of all clients to 24 Mbps, which corresponds to a total rate of 120Mbps, or 40% of the simulator's capacity. We disable all rate-limiting and run the simulator for 5 minutes (≈1M slots), measuring the average latency of private and indifferent clients, and the total throughput. We then increment each client's rate by 6 Mbps, repeating the entire process until the throughput has flattened out. This experiment gives us an idea of how different clients are impacted in the absence of rate limits.

The second experiment is the same as the first, but rate-limits indifferent clients to 45 Mbps. The third experiment is the same as the second but only increments the rates of indifferent clients during each trial—private clients' rates are fixed at 42 Mbps (we chose this value as it was the inflection point at which the latency spiked for private clients in the second experiment). This last experiment helps us tease out if private clients are at all negatively impacted by rate-limited indifferent clients increasing the load in the system (thereby violating IFS's sharing incentive properties).

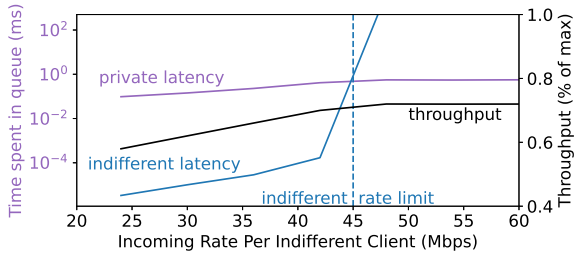
Figure 11 gives the results. Figure 11a shows that as clients' load increase, private clients begin to starve early on and experience a latency spike at around 42 Mbps due to indifferent clients going



(a) No rate limiting



(b) Rate-limited indifferent clients



(c) Rate-limited indifferent clients with fixed private client rate

Figure 11: Effect of clients' rates on average private latency (purple), indifferent latency (blue), and throughput (black). When applicable, we show the rate limit imposed on indifferent clients (blue dashed).

first. Indifferent clients, on the other hand, only suffer later when the switch approaches its capacity. This confirms our intuition that without rate-limiting, indifferent clients are free to consume all of the available bandwidth and completely starve private clients.

Figure 11b shows that placing a rate limit on indifferent client is indeed sufficient to prevent the degradation of private clients' latency while still keeping cumulative throughput high. Note here throughput flattens at around 80% of the switch's capacity, in contrast to the first experiment. This is not because of dummy packets—the private queues are congested and so never need to process a dummy—but rather due to private clients padding their packets which reduces the system's goodput.

Lastly, Figure 11c demonstrates that private clients', whose rate is fixed, continue to experience the same service even as indifferent clients increase their load. Likewise, observe that the line for indifferent clients is nearly identical to that of the second experiment despite the fact that private clients start off sending more traffic than indifferent ones. This is expected since in IFS, indifferent clients are not affected by private ones. Note that the cumulative throughput here is lower because we are not increasing the load from private clients, and indifferent clients are rate limited, so no client is using the switch's spare capacity.

8 DISCUSSION

IFS is a hybrid scheduling discipline that provides privacy to clients who want it, without burdening clients who are indifferent. While we are able to build IFS at line rate on switches that support PIFO, and our evaluation confirms that indifferent clients are as well off or even better under IFS than they are today, the requirements for private clients are high.

Main limitations. Private clients need to maintain dummy pools for all egress ports (Figure 3), which is costly. A compromise is for them to have dummy pools only for the egress ports they use, at the risk of potentially leaking the destination of their traffic (although redundant data center topologies like FatTrees [6], VL2 [31], and F10 [46] might sufficiently obscure the destination). This limitation could be addressed with switch extensions. For example, support for idling or the generation of packets would free clients from having or stocking dummy pools. Alternatively, if queues could be associated with multiple ports, some of those ports could be used to recirculate dummies as we do for padding (§6.3).

IFS also requires the switch to have access to a number of queues that is linear in the number of clients. Even in a setting where we treat clients as hosts rather than network flows, this is challenging to satisfy. One potential idea is to leverage TEA [38], which allows switches to use external memory to store additional state for lookup tables. With TEA (or some other similar architecture), IFS could store buffers off-switch, thereby virtualizing the necessarily large number of queues.

Potential optimization. In our threat model (§3.1), the attacker targets a particular switch. As a result, it might be hard for the attacker to "chase" packets deep into the network, as doing so would require probing all potential switches on all possible paths without prior knowledge of which service the victim is even accessing. Furthermore, switches deeper in the network aggregate traffic from many clients, masking the contribution of any one client. Consequently, clients might be able to let dummy packets have small TTLs so that they are discarded early on in the network to reduce overhead for other switches. We leave finding the optimal TTL as a function of network topology, packet aggregation rate, and attacker capabilities as an interesting open question.

Algorithmic extensions. In modeling clients' actions and desires, which were critical to our incentive compatibility argument of Section 4.3, we made a few assumptions. First, we assumed that private client's desire for privacy was absolute: in other words, private clients accept any additional delay instead of risking a privacy violation. Second, we assumed that privacy is a binary notion: either one enjoys privacy or not. Third, we focused only on packet delay but not on throughput.

Future work could relax these assumptions. For example, we could allow for "partially private" clients, which are clients willing to leak a controlled amount of information. The challenge here is characterizing the leakage of information that results from an attackers' observations and devising a mechanism that can enforce

a bound on such leakage. A starting point is to draw inspiration from differential privacy [26].

Since IFS's treatment of indifferent clients is work conserving, it gifts any slack to private clients. As a result, private clients are actually allocated a bandwidth share higher than the rate they purchased whenever the switch is underutilized (due to indifferent clients idling). This might lead an indifferent client who greatly values bandwidth and who is willing to tolerate a higher latency to label itself as private to enjoy more bandwidth at a lower financial cost. We could incorporate this additional variable into clients' objective functions, and design a variant of IFS that incentivizes truthfulness in the presence of these other objectives.

Code. Our code is available at https://github.com/eniac/IFS.

Acknowledgments

We thank the SIGCOMM and CCS reviewers for their feedback, which significantly improved the content and presentation of our work. We also thank Vincent Liu for invaluable discussions. This work was funded in part by NSF grants CCF-1733794, CNS-2045861, CNS-2107147, CNS-2124184; by DARPA contract HR0011-17-C0047; and by a gift from JP Morgan Chase & Co. Any views or opinions expressed herein are solely those of the authors listed.

REFERENCES

[1] Alexa - top sites. https://www.alexa.com/topsites.
[2] iperf3. https://iperf.fr, 2015.
[3] pifo-hardware. https://github.com/programmable-scheduling/pifo-hardware, 2015.
[4] C++ reference implementation of a pipeline of Push-In First-Out queues. https://github.com/programmable-scheduling/pifo-machine, 2016.
[5] Sp-pifo: Approximating push-in first-out behaviors using strict-priority queues. https://github.com/nsg-ethz/sp-pifo, 2020.
[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM Conference*, 2008.
[7] A. G. Alcoz, A. Dietmüller, and L. Vanbever. SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Feb. 2020.
[8] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska. End-to-end performance isolation through virtual datacenters. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2014.
[9] S. Angel, S. Kannan, and Z. Ratliff. Private resource allocators and their applications. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2020.
[10] S. Angel, D. Lazar, and I. Tzialla. What's a little leakage between friends? In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES)*, Oct. 2018.
[11] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Nov. 2016.
[12] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM Conference*, 2011.
[13] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea. Chatty tenants and the cloud network sharing problem. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
[14] O. Berthold and H. Langos. Dummy traffic against long term intersection attacks. In *Proceedings of the Workshop on Privacy Enhancing Technologies (PET)*, Mar. 2002.
[15] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted HTTP streams. In *Proceedings of the Workshop on Privacy Enhancing Technologies (PET)*, 2005.
[16] A. Cabrera Aldaya, B. B. Brumley, S. ul Hassan, C. Pereida García, and N. Tuveri. Port contention for fun and profit. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2019.
[17] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), Feb. 1981.
[18] D. L. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1), 1988.
[19] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed onion routing at the network layer. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Oct. 2015.
[20] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso. TARANET: Traffic-analysis resistant anonymity at the network layer. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, Apr. 2018.
[21] R. Cooper. *Introduction to Queueing Theory*. North Holland, 1981.
[22] H. Corrigan-Gibbs and B. Ford. Dissent: Accountable anonymous group messaging. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, Oct. 2010.
[23] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2003.
[24] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of the ACM SIGCOMM Conference*, 1989.
[25] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the USENIX Security Symposium*, Aug. 2004.
[26] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Theory of Cryptography Conference (TCC)*, Mar. 2006.
[27] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-Boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2012.
[28] A. Ghassami and N. Kiyavash. A covert queueing channel in FCFS schedulers. *IEEE Transactions on Information Forensics and Security*, 13(6), 2018.
[29] X. Gong and N. Kiyavash. Quantifying the information leakage in timing side channels in deterministic work-conserving schedulers. *IEEE/ACM Transactions on Networking (TON)*, 24(3), 2016.
[30] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Transactions on Networking*, 5(5), Oct. 1997.
[31] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM Conference*, 2009.
[32] C. R. Heathcote. Preemptive priority queueing. *Biometrika*, 48(1/2):57–63, 1961.
[33] S. Kadloor, X. Gong, N. Kiyavash, T. Tezcan, and N. Borisov. Low-cost side channel remote traffic analysis attack in packet networks. In *Proceedings of the IEEE International Conference on Communications (ICC)*, Aug. 2010.
[34] S. Kadloor, X. Gong, N. Kiyavash, and P. Venkitasubramaniam. Designing router scheduling policies: A privacy perspective. *IEEE Transactions on Signal Processing*, 60(4):2001–2012, 2012.
[35] S. Kadloor and N. Kiyavash. Delay-privacy tradeoff in the design of scheduling policies. *IEEE Transactions on Information Theory*, 61(5), 2015.
[36] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam. Mitigating timing side channel in shared schedulers. *IEEE/ACM Transactions on Networking (TON)*, 24(3), 2016.
[37] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. In *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Sept. 1998.
[38] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan. TEA: Enabling state-intensive network functions on programmable switches. In *Proceedings of the ACM SIGCOMM Conference*, 2020.
[39] J. F. C. Kingman. On queues in heavy traffic. *Journal of the Royal Statistical Society. Series B (Methodological)*, 24(2):383–392, 1962.
[40] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Horizontally scaling strong anonymity. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2017.
[41] A. Kwon, D. Lazar, S. Devadas, and B. Ford. Riffle: An efficient communication system with strong anonymity. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, July 2016.
[42] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A scalable, traffic analysis resistant anonymity network for VoIP systems. In *Proceedings of the ACM SIGCOMM Conference*, Aug. 2015.
[43] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. In *Proceedings of the ACM SIGCOMM Conference*, Aug. 2013.
[44] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma. Application-driven bandwidth guarantees in datacenters. In *Proceedings of the ACM SIGCOMM Conference*, 2014.
[45] M. Liberatore and B. N. Levine. Inferring the source of encrypted HTTP connections. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.
[46] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A fault-tolerant engineered network. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
[47] M. B. Mamoun, J.-M. Fourneau, and N. Pekergin. Analyzing weighted round robin policies with a stochastic comparison approach. *Computers & Operations*

Research, 35(8), 2008.
[48] A. Mehta, M. Alzayat, R. de Viti, B. B. Brandenburg, P. Druschel, and D. Garg. Pacer: Network side-channel mitigation in the cloud, 2020.
[49] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3), June 1993.
[50] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos. Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing. In *Proceedings of the ACM SIGCOMM Conference*, 2013.
[51] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud:exploring information leakage inthird-party compute clouds. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2009.
[52] D. Shah and J. Shin. Randomized scheduling algorithm for queueing networks. *Annals of Applied Probability*, 22(1), Feb. 2012.
[53] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.
[54] A. Sivaraman, S. Subramanian, A. Agrawal, S. Chole, S.-T. Chuang, T. Edsall, M. Alizadeh, S. Katti, N. McKeown, and H. Balakrishnan. Towards programmable packet scheduling. In *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2015.
[55] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *Proceedings of the ACM SIGCOMM Conference*, 2016.
[56] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2012.
[57] C. V. Wright, L. K. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Uncovering spoken phrases in encrypted voice over IP conversations. *ACM Transactions on Information and System Security*, 13(4), 2010.
[58] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of the USENIX Security Symposium*, 2012.
[59] Z. Yu, C. Hu, J. Wu, X. Sun, V. Braverman, M. Chowdhury, Z. Liu, and X. Jin. Programmable packet scheduling with a single queue. In *Proceedings of the ACM SIGCOMM Conference*, 2021.
[60] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2011.

A DELAY AND CLIENT COMPOSITION

IFS is a hybrid scheduling algorithm that treats two types of clients: private and indifferent. Here, we study the effect that varying the number of clients of a particular type has on the average delay experienced by clients of each type. Our experiment is done on the PIFO simulator [4] and consists of four clients, each with a Poisson arrival rate with mean 0.1 packets/slot. For each trial, we measure the average waiting time experienced by each client, and then compute the average of these values for the private and indifferent clients respectively. We then vary the number of clients that are private and indifferent. We start with all four clients being private and then progressively convert one client each trial to become indifferent until all clients are indifferent. Like our other IFS simulations, we ensure that each private client has a sufficient amount of dummy packets already present in the system

Figure 12 gives the results. We make three observations:

- The average delay experienced by private clients is strictly higher than that of indifferent clients in all configurations.
- The delay of indifferent clients is strictly increasing.
- The delay of private clients is strictly decreasing.

The first observation showcases IFS's incentive compatibility (§4.3), whereas the latter two demonstrate IFS's indifferent and private delay monotonicity (§4.4). We have also experimented with other rates and number of clients, and the trends are similar.

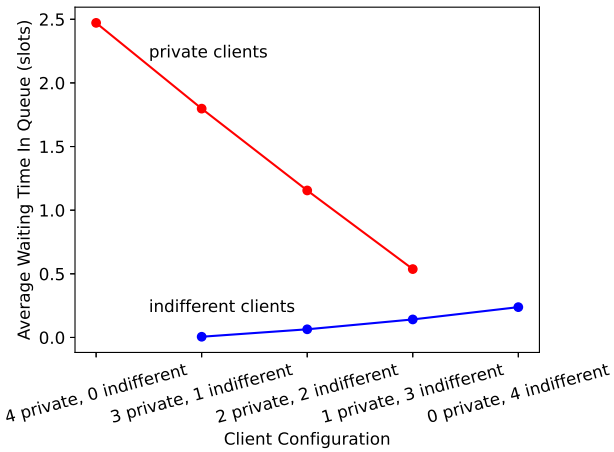


Figure 12: For each configuration described by the x-axis, the red point represents the average waiting time of a private client in that configuration and the blue point represents the same for an indifferent client. We can see that the red line is strictly higher, which incentivizes indifferent clients.

B INDIFFERENCE INCENTIVE

Recall from Section 4.3 the definition of indifference incentive. That is, an indifferent client is incentivized to tell the truth because doing so is its dominant strategy.

We formalize this as follows. Let P and I be the sets of active private and indifferent clients, respectively. Let c be a client that is considering whether to declare itself as private or indifferent. IFS guarantees that for all P and I:

$$D_c(\lambda_c, P, I \cup \{c\}) \leq D_c(\lambda_c, P \cup \{c\}, I)$$

where D_c is the expected delay for c's packets given a rate λ_c, and a set of private and indifferent clients. Note that here we assume that either no rate limiting is done, or all clients (private and indifferent) are rate limited. Otherwise, this definition does not always hold as we discuss in Section 4.5.

Proof. Let a scheduler's client allocation sequence $S = c_a, c_b \ldots$ be defined such that the ith element in S, $S_i = c_a$ if and only if the scheduler allocates the ith slot to client c_a, i.e. the scheduler processes a packet from client c_a in its ith time slot. Note that if there is at least one private client registered to the scheduler, each slot will be allocated to one client; In the case of all indifferent clients, we represent unallocated slots by letting that element of the sequence be $\emptyset$.

We start with the following simple lemma:

Lemma B.1. *If $S_i = c_a$ and $c_a \in P$, then at time slot i no indifferent client can have any packets waiting at the scheduler.*

Proof. If there were any indifferent packets queued, IFS would have processed them before allocating a slot to a private client due to its strict priority for indifferent clients. □

We can also partition a sequence into subsequences in which the scheduler allocated slots to indifferent clients and those in which

the scheduler allocated slots to private clients, and if we require these subsequences to be as long as possible, this partitioning is unique.

Now we will examine two client allocation sequences S and S'. The sequence S occurs in a scheduler with private clients P and indifferent clients I such that there is an indifferent client $c \in I$. S' describes the allocation that occurs in a nearly identical scheduler with private clients $P' = P \cup \{c\}$ and indifferent clients $I' = P \setminus \{c\}$.

Now we will examine a packet p sent by c. Suppose that this packet was processed at time slot i, so that $S_i = c$. S_i falls into an indifferent partition, and we claim that for any random values chosen by the scheduler, p would have been processed in this partition. p could not have been processed in a previous partition, because the immediate preceding partition is private and this contradicts Lemma B.1. Similarly, p could not have been processed in a later partition, because the immediate following partition is also private. We also know that p must have arrived during this same partition.

We will now examine how the partitions of S differ from those of S'. There are three cases: (1) an indifferent partition in S consisted entirely of packets produced by c, in which case when c becomes private this partition becomes private and merges with its two surrounding partitions, and (2) an indifferent partition in S contains no packets from c, in which case this partition does not change in S', and (3) an indifferent partition in S contains packets from c as well as those from clients that are not c, in which case the partition splits in S', with the slots allocated to c merging with the following, private partition. Finally, we note that in case (1), the set of packets from c will not be assigned slots earlier than they were in S (individual packets may fare better, but only by switching slots with other packets from c) and as well for case (3). Further, if there is at least one other client in the scheduler, either indifferent or private, then c will perform worse on expectation. □

C INDIFFERENT DELAY MONOTONIC

First, we state the following. Given four clients, p, q, r, and s:

$$D_p(\lambda_p, \{r, s\}, \{p, q\}) \leq D_p(\lambda_p, \{s\}, \{p, q, r\})$$

In fact, this statement holds not just for the expected delay, but also for any possible arrival patterns that are compatible with the clients' announced rates.

This is trivially true when the indifferent clients use FIFO. When we use randomized round robin, we can define our random selection over indifferent clients in the following way: each time slot, we randomly choose a client from the set $\{p, q, r\}$, based on each client's purchased rate. If the first client selected is indifferent and has a packet queued, that client's packet will be processed this slot. Otherwise, remove this client from the set, and if r is private and still in the set, remove r also. Now continue to pick clients from the set until the selected client has a packet queued—if the set is empty before this happens, idle this slot. This will yield our desired selection probabilities, but will allow us to use the same random selections regardless of the privacy of client r, allowing us to compare the above situations. It is clear that no packet of p will benefit from the indifference of r.

Note that both q and s can be either zero-rate clients (identical to an empty set of clients) or the client obtained by combining multiple clients' packet sequences (identical to a set of multiple clients). Therefore, the above property covers every possible case claimed by indifferent delay monotonicity.

D PRIVATE DELAY MONOTONIC

Recall from Definition 4.2 in Section 4.4 that private delay monotonicity means that as more clients become private, none of the existing private processes is better off. As a result, private clients do not care about the composition of the system (in terms of private or indifferent clients). We conjecture that IFS is private delay monotonic.

We do not have a formal proof for this conjecture, but we have empirical evidence that supports this (Appendix A). We also prove it to be true in a special case. We leave it for future work to formally prove or disprove this conjecture. It requires modeling complex interactions between multiple scheduling algorithms, which is outside the scope of this work.

Evidence in favor. We can show the validity of our conjecture in the specific case where there are exactly two clients of Poisson rates present, with some minor stipulations. Notice that in this situation, our policy reduces to a simple priority queue when one client chooses to be private and the other indifferent, and p-TDMA when both are private. We have provided the delay for the latter case in Appendix E. For the former case, we reference the work of Heathcote [32] to obtain the waiting time of a second class client in a preemptive priority queue of M/D/1 form with a fixed service distribution of 1. Letting p denote our private client and q our indifferent client, with λ_p, λ_q their respective rates and $\lambda = \lambda_p + \lambda_q$, the expected waiting time of client p is:

$$\frac{1 - \frac{\lambda}{2}}{(1 - \lambda_q)(1 - \lambda)}$$

We can compare this waiting time with that obtained via our equation for p-TDMA. Our conjecture holds when the waiting time of p is lower when q is indifferent. If we assume that $\lambda_p = \lambda_q$, then we find that this waiting time is strictly lower than that of pTDMA on the interval $(0, 1)$, so our conjecture holds throughout.

We will also explore two additional cases in this two-client situation: We will let $\lambda_p = \frac{1}{b} \cdot \lambda$ or $\lambda_p = \frac{b-1}{b} \cdot \lambda$ for some positive integer b. For the first case, we can calculate the expected delay directly from our p-TDMA equation as

$$W_{ptdma}(\lambda, b) = 1 + \frac{1}{2(1 - \lambda)} + \frac{b - 1}{1 - \lambda}$$

and from the priority queue delay above as

$$W_{prio}(\lambda, b) = \frac{1 - \frac{\lambda}{2}}{(1 - \frac{b-1}{b}\lambda)(1 - \lambda)}$$

Letting $f(\lambda, b) = W_{prio}(\lambda, b) - W_{pdtma}(\lambda, b)$, we can compute the partial derivative of f with respect to b an find that this is negative when $0 < \lambda < 1$. Because $f(\lambda, 1)$ is negative for all λ, $f(\lambda, b)$ is negative for the range of values we care about.

In the second case, we can compute the p-TDMA delay when $\lambda_p = \frac{b-1}{b}$ by using the fact that, letting this delay be $\hat{W}_{ptmda}(\lambda, b)$,

$$\frac{1}{b}W_{ptdma}(\lambda, b) + \frac{b-1}{b}\hat{W}_{ptdma}(\lambda, b) = W_{ptdma}(\lambda, 2)$$

Using the same method as above, we find that $g(\lambda, b) = W_{prio}(\lambda, \frac{b}{b-1}) - \hat{W}_{ptdma}(\lambda, b)$ and the partial derivative of g with respect to b is negative when $0 < \lambda < 1$ and $b > 1$. This, coupled with the fact that the limit of $g(\lambda, b)$ approaches $-.5$ as b goes to infinity, regardless of λ, means that $g(\lambda, b)$ is also negative for the range in which we are concerned.

While admittedly limited, these two cases, and our inability to identify counterexamples analytically or empirically, give us hope that our conjecture (or some slight weakening that perhaps takes into account the scheduler's admission threshold L) might be true for all settings.

E WORST-CASE EXPECTED DELAYS

To calculate the expected delay for a private client c, we can use the following fact, proven by Kadloor et al. [36]: for a proportional TDMA scheduler with n clients with combined rate λ, the average delay experienced for any packet is:

$$1 + \frac{1}{2(1-\lambda)} + \frac{n-1}{1-\lambda}$$

We can find the delay for a client with rational rate that consumes a/b of the total rate by first calculating the delay of a client with rate $1/b$, using the fact that in p-TDMA regardless of the number of clients, a client with rate λ in a scheduler with combined rate Λ will always have the same delay.

To calculate the expected delay for an indifferent client, we first note that all of the other guarantees of IFS are independent of the specific flavor of round robin we use, although in Section 4.1 we have abstracted it away as a random selection. The work of Shah and Shin [52] provides an analysis of the expected delay of random scheduling, which is a similar concept to ours but in a different context, and the work of Mamoun, Fourneau, and Pekergin [47] provides an analysis of the delay of (deterministic) weighted round robin. If instead we use FIFO on the indifferent clients, the expected delay is $1 + \frac{1}{2(1-\lambda)}$ [21].

F EQUIVALENCE OF DUMMY POOLS

We can show equivalency between a model in which the switch idles for a time slot where a private client has nothing to send ("abstract model"), and a model in which the switch leverages a dummy packet that has been previously enqueued ("actual model").

We make the simplifying assumption that there is no latency between the client and the switch. In practice, this just means that the switch has all of the dummies already present, which can be ensured by having clients preload their dummy pools and maintain them populated. First we will fix all of the inputs (clients and corresponding rates, packets departure times and corresponding arrival times, and internal randomness) for both models.

We will start with the case where all clients are private. Let $s = 0, 1, 2, \ldots$ be the sequence of slots over the lifetime of the scheduler. We can partition s into subsequences over the set of clients, i.e. s_p is the subsequence of s assigned to client p. Because indifferent clients do not generate dummy packets, the arrival times for the packets of each indifferent client will be identical in both models. One consequence of IFS is that private clients cannot affect the schedule of any indifferent client, so for any indifferent client p_i, the subsequence s_{p_i} will also be identical in both models. We know that any slot not allocated to an indifferent client will be allocated to a private client, and the selection of which private client depends only on the reserved rates of the private clients and a sequence of random selections, both of which we have ensured to be equal in both models. Therefore, for any client, indifferent or private, p, the subsequence s_p is identical in the two models.

Now we will introduce a function $next_p(i) = \min\{x | x > i, x \in s_p\}$. In other words, $next_p(i)$ returns the next slot allocated to client p after slot i. We will also let $arr(r_i)$ and $dep(r_i)$ express the arrival and departure slots associated with packet i, respectively. We assume that all packets arrive at the end of a slot, after the packet associated with that slot has left the queue.

Lemma F.1. *For two packets of a common client p, r_i and its preceding packet r_{i-1},*

$$dep(r_i) = next_p(max(arr(r_i), dep(r_{i-1})))$$

Proof. Assume towards contradiction that a packet r_i actually departs at a slot $dep(r_i)$ which is different from the one our lemma predicts, $dep'(r_i)$.

In both models, for any packet r_i, $dep(r_i) > arr(r_i)$. When r_i arrives at the queue, at slot $arr(r_i)$, there are two possibilities for r_{i-1}. Either it had left the queue on this turn or a previous one ($dep(r_{i-1}) \leq arr(r_i)$), or it is still present in the queue. In the latter case, we know it will be processed before r_i, so either way, $dep(r_i) > dep(r_{i-1})$. We also know that $dep(r_i) \in s_p$. Combining these three statements, we have that $dep(r_i) > dep'(r_i)$.

Now consider what kind of packet is handled at slot $dep'(r_i)$. By definition, $dep'(r_i) \in s_p$, so it must be a packet associated with client p. Further, $dep(r_i) > dep'(r_i) > dep(r_{i-1})$, so because it occurs between the departures of two consecutive packets from client p, it cannot be an actual packet, so it must be a dummy. But at this slot the subqueue associated with client p contains r_i, as $dep(r_i) > dep'(r_i) > arr(r_i)$. In the case of the abstract model, a dummy packet for process p will not be created with an actual packet for client p in the subqueue; for the real model, even if there is a dummy packet in the subqueue it will not be handled before the higher priority actual packet (r_i). Therefore, no such packet other than r_i can be assigned to slot $dep'(r_i)$, which is a contradiction.

We can argue similarly that the first packet r_0 of each client p will be allocated to slot $next_p(arr(r_0))$. At this point we have shown that in both models, each of client p's packets $r_0, r_1, r_2, \ldots$ are assigned the same departure slots $d_p = dep(r_0), dep(r_1), dep(r_2), \ldots$. Further, if we take any slot in s_p that is not present in d_p, we can show that at this slot there are no actual packets associated with p in the queue. Therefore, each such slot is idled (in the abstract model) or assigned to a dummy packet of p (in the real model). □