# 3 Scheduling

So far, we mostly looked at *static* routing problems such as BMFPs, i.e. all demands (or packets) are given from the beginning and no new demands (or packets) arrive during the routing. In the real world, however, packets are usually injected at the nodes in a continuous fashion. Models that take this into account are called *dynamic routing models*, as opposed to the *static routing models* we mostly considered before. These models can be usually characterized as either *stochastic* or *adversarial*. In stochastic models, the injection of packets is modelled with the help of stochastic processes, whereas in the adversarial model it is assumed that an adversary controls the injection of new packets [2]. In this section, we restrict ourselves to considering the following variant of the adversarial model, which is due to [1].

## 3.1 The adversarial injection model

In this model we have an adversary that is allowed to demand network bandwidth up to some specified limit. That is, it is allowed to choose any node at any time step to inject a new packet and it is allowed to select any path for the injected packet as long as the load of the edges does not exceed a certain limit. More formally, for any $w, \lambda > 0$, an adversary is called a $(w, \lambda)$-*bounded adversary* if for all edges $e$ and all time intervals $I$ of length $w$, it injects no more than $\lambda \cdot w$ packets during $I$ that contain edge $e$ in their routing paths. $\lambda$ is called the *injection rate* and $w$ is called the *burstiness* of the adversary. We demand that the adversary has to tell the system the paths it selects. Thus, we are left with solving a *dynamic scheduling problem*. An algorithm for this problem is called a *scheduling protocol*. A scheduling protocol is called *stable* for some $\lambda$ and some network $G$ if for any injection rate of at most $\lambda$ and any paths selected for the packets in $G$ the (expected) worst-case routing time of a packet (i.e. the time it spends in the system) does not grow unboundedly with time. Since an edge can transport at most one packet per step, $\lambda$ can be at most 1. A protocol that is stable for any network and any $\lambda < 1$ is called *universally stable*. We will show that there are both universally stable and non-universally stable protocols.

## 3.2 Queueing disciplines

Usually, every node has a packet queue for every outgoing edge. Hence, the simplest form of a scheduling protocol are protocols in which we just define a rule about how to rank packets in a queue so that we know which packet to send out next along an edge. Elementary queueing disciplines are:

- FIFO (first in first out): gives preference to the packet that was the first to arrive at the queue (among those that are still in the queue).
- NTO (nearest to origin): gives preference to the packet that has traveled the smallest amount of edges so far. Ties are broken arbitrarily.
- FTG (furthest to go): gives preference to the packet that has the largest number of edges to go. Ties are broken arbitrarily.
- NTG (nearest to go): gives preference to the packet that has the smallest number of edges to go. Ties are broken arbitrarily.

- SIS (shortest in system): gives preference to the youngest packet.
- LIS (longest in system): gives preference to the oldest packet.

Due to its simplicity, FIFO is the most widely used scheduling protocol in practice.

## 3.3 Universal stability of SIS

The SIS protocol always gives preference to the youngest packet. Ties may be broken arbitrarily. For this protocol we can prove the following amazing result.

**Theorem 3.1** *SIS is universally stable.*

**Proof.** Let $0 < \epsilon < 1$ be chosen such that $\lambda = 1 - \epsilon$. Suppose that there is a packet $P$ that requires more than $\sum_{i=1}^{d}(w+1)/\epsilon^i$ steps to traverse a path of length $d$. In this case there must be an $i \in \{1, \ldots, d\}$ for which $P$ was delayed for at least $(w+1)/\epsilon^i$ steps at the $i$th edge of its path. Consider the minimal $i$ for which this holds. Let $e$ be the corresponding edge on $P$'s path. Then the time difference $T$ between the injection of $P$ and the time at which $P$ had to wait for the $(w+1)/\epsilon^i$th time at $e$ is at most

$$\sum_{j=1}^{i}(w+1)/\epsilon^j = \frac{w+1}{\epsilon} \cdot \frac{(1/\epsilon)^i - 1}{(1/\epsilon) - 1} = (w+1) \cdot \frac{(1/\epsilon)^i - 1}{1-\epsilon} \; .$$

Since $\lambda = 1 - \epsilon$, at most

$$(1-\epsilon)T + w = (w+1) \cdot \Big((1/\epsilon)^i - 1\Big) + w < (w+1)/\epsilon^i$$

packets can be injected during $T$ time steps that intend to cross $e$. Only these packets can be preferred against $P$ at $e$. However, since our assumption requires $P$ to be delayed by at least $(w+1)/\epsilon^i$ packets at $e$, we arrive at a contradiction Thus, the delay of a packet following a path of length $d$ can be at most $\sum_{i=1}^{d}(w+1)/\epsilon^i$, and therefore SIS must be stable. □

## 3.4 Universal stability of LIS

In contrast to SIS, the LIS protocol always gives preference to the oldest packet. Also LIS is stable.

**Theorem 3.2** *LIS is universally stable.*

**Proof.** Let $\lambda = 1 - \epsilon$ and let $D$ be the maximal length of a path that can be chosen by the adversary. Suppose that there is a packet that requires more than

$$\sum_{j=0}^{d-1}(w+1)/\epsilon^{D-j}$$

steps to traverse a path of length $d$. Let $P$ be the first packet with this property and let its $i$th edge be the first edge at which this property is fulfilled. Let $e$ be the $i$th edge on $P$'s path, and let $t$ be chosen such that at the beginning of time step $t$ $P$ has an age of $\sum_{j=0}^{i-2}(w+1)/\epsilon^{D-j}$ steps. Then $P$ was delayed

at $e$ for at least $(w+1)/\epsilon^{D-(i-1)}$ steps following $t$. Since, according to our assumptions, the maximal age of any packet before time step $t+(w+1)/\epsilon^{D-(i-1)}$ was at most $\sum_{j=0}^{D-1}(w+1)/\epsilon^{D-j}$, the difference $T$ between this age and the age of $P$ at time step $t$ is at most

$$\begin{aligned}\sum_{j=0}^{D-1}(w+1)/\epsilon^{D-j} - \sum_{j=0}^{i-2}(w+1)/\epsilon^{D-j} &= \sum_{j=i-1}^{D-1}(w+1)/\epsilon^{D-j} = \sum_{j=1}^{D-i+1}(w+1)/\epsilon^{j} \\ &= \frac{w+1}{\epsilon}\cdot\frac{(1/\epsilon)^{D-i+1}-1}{(1/\epsilon)-1} \\ &= (w+1)\cdot\frac{(1/\epsilon)^{D-i+1}-1}{1-\epsilon}\,.\end{aligned}$$

Since $\lambda = 1-\epsilon$, at most

$$(1-\epsilon)T + w = (w+1)\left((1/\epsilon)^{D-i+1}-1\right) + w < (w+1)/\epsilon^{D-i+1}$$

packets can be injected from time step $t - \sum_{j=0}^{D-1}(w+1)/\epsilon^{D-j}$ to the injection time of $P$ that intend to cross $e$. Since these are the only packets that can delay $P$ at $e$, and our assumptions require $P$ to be delayed by at least $(w+1)/\epsilon^{D-i+1}$ packets, we arrive at a contradiction. Hence, LIS must be stable. □

## 3.5 Instability of FIFO

The FIFO protocol always gives preference to the packet that was the first to arrive at a node. The next result demonstrates that FIFO is not universally stable.

**Theorem 3.3** *For $\lambda \geq 0.85$ there is a network and an adversary that causes FIFO to be unstable.*

**Proof.** The adversary uses the network shown in Figure 1. It is not difficult to check that this network can be embedded in a grid and a 3-dimensional hypercube. Thus, it is possible for our counterexample to happen in important standard networks.

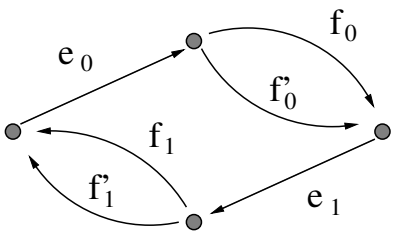


Figure 1: The counterexample for FIFO.

We divide the injection strategy of the adversary into several phases. Our induction hypothesis will be that at the beginning of phase $j$ there are at least $s+j$ packets in the buffer of $e_i$ that intend to cross the edges $e_i$ and $f_i$, where $i = j \bmod 2$ and $s$ is a sufficiently large constant.

In phase 1 we have to inject packets in such a way that at the end of this phase there are $m = s+2$ packets in the buffer of $e_0$ that have a remaining path of $(e_0 f_0)$. This can be achieved by attaching a set of $m$ lines of nodes to the starting node of $e_0$, where line $i$ has a length of $i/\lambda$. An injection rate of

$\lambda$ allows $m$ packets to be injected at the endpoints of the lines, one per endpoint, so that all $m$ packets reach $e_0$ at the same time. This ensures the induction hypothesis for phase 2.

We consider now an arbitrary even phase $j$. The odd phases work in the same way. We will show that if at the beginning of phase $j$ the buffer of $e_0$ contains a set $M_0$ of $m = s + j$ packets with remaining path $(e_0 f_0)$, then at the beginning of phase $j + 1$ there will be at least $m + 1$ packets in the buffer of $e_1$ that have a remaining path of $(e_1 f_1)$.

In the following we describe the injection strategy of the adversary for phase $j$. To simplify the proof, we will avoid dealing with floors and ceilings. If $m$ is sufficiently large, then their effects can be neglected. Phase $j$ consists of 3 stages.

Stage 1 consists of $m$ steps. During this stage we inject a set $M_1$ of $\lambda m$ packets with path $(e_0 f_0' e_1 f_1)$. These packets are blocked by the packets in $M_0$. Furthermore, we inject $\lambda m$ packets with path $(f_0)$. These packets ensure that at the end of stage 1 at least $\lambda m$ packets with remaining path $(f_0)$ are still in the buffer of $f_0$.

Stage 2 consists of $\lambda m$ steps. During this stage we inject a set $M_2$ of $\lambda^2 m$ packets with path $(f_0 e_1 f_1)$. These are blocked by the packets that are already in the buffer of $f_0$. Furthermore, we inject $\lambda^2 m$ packets with path $(f_0')$. These mix with the packets in $M_1$. Every $1/\lambda$ steps we have in addition to $1/\lambda$ packets from $M_1$ that reach the buffer of $f_0'$ one packet with path $(f_0')$. This has the effect that every $1 + 1/\lambda$ steps $1/\lambda$ packets from $M_1$ pass $f_0'$, which reduces the number of packets in the buffer of $f_0'$ after stage 2 to

$$\lambda m - \frac{\lambda m}{1 + 1/\lambda} \cdot \frac{1}{\lambda} = \lambda m - \frac{\lambda m}{\lambda + 1} = \frac{\lambda^2 m}{\lambda + 1} .$$

Stage 3 consists of $\lambda^2 m$ steps. During this stage, the packets in $M_1$ and $M_2$ move forward and mix in the buffer of $e_1$. Furthermore, $\lambda^3 m$ packets are injected with path $(e_1 f_1)$. Since $\lambda^2 m$ packets traverse $e_1$, the number of packets in the buffer of $e_1$ with remaining path $(e_1 f_1)$ after stage 3 is equal to $\lambda^3 m + \lambda^2 m/(\lambda + 1)$.

This ends phase $j$. Since for $\lambda \geq 0.85$ we have that $\lambda^3 + \lambda^2/(\lambda + 1) > 1$, we arrive at the induction hypothesis for phase $j + 1$. □

Although FIFO cannot handle well adversarial traffic, it was shown by Bramson that it is actually universally stable for stochastic traffic [3]. Since FIFO queues are so much easier to build than other queues and Internet traffic is of somewhat stochastic nature, this explains why there has been no need so far to choose queueing disciplines other than FIFO in Internet routers.

## 3.6 Routing in leveled networks

The stability results for SIS and LIS above demonstrate that the worst-case delay of a packet is finite, but it may be very large. In this section we demonstrate that in certain situations much better delay bounds can be shown.

Consider using the LIS rule in leveled networks. We assume that every packet has a rank that is determined by its birth date plus some small offset $< 1$ that gives the packets a strict ordering (so that we never run into a tie). When using the adversary in a leveled network, we only allow the adversary to inject paths that go from lower to higher levels, i.e. for every edge $e$ on such a path, $e$ must go from some node in level $k$ to some node in level $k + 1$.

**Theorem 3.4** *Let $L$ denote the depth of the leveled network. For any $(w, \lambda)$-bounded adversary with $\lambda \leq 1$, LIS is stable, and every packet reaches its destination in at most $(1 + \lambda w)L$ time steps.*

**Proof.** Suppose on the contrary that there is a packet $p_0$ that needs more than $(1 + \lambda w)L$ time steps to reach its destination. We will use a delay sequence argument to show that this is not possible. First, we follow $p_0$ backwards in time from the point where it reached its destination until it was delayed by some packet $p_1$. We then follow $p_1$ backwards in time until is was delayed by some packet $p_2$, and so on, until we get to a packet $p_s$ that had no prior delays. The path $q$ recorded in this process is called a *delay path* and must have a length of at most $L$, because we only allow the adversary to inject paths with edges from level $k$ to $k + 1$ for some $k$. Since LIS is used, we also know that for every $i \in \{1, \ldots, s\}$, $p_i$ must be at least as old as $p_{i-1}$. Hence, the time spanned by the delay sequence must be more than

$$(\mathrm{birth}(p_0) - \mathrm{birth}(p_s)) + (1 + \lambda w)L \ ,$$

and the total amount of delay events must therefore be more than

$$(\mathrm{birth}(p_0) - \mathrm{birth}(p_s)) + \lambda w L \ .$$

Furthermore, for every edge $e$ along the delay path $q$ we can associate an interval $I_e$ containing the birth dates of all packets recorded in the delay sequence at $e$. Since the birth dates of the packets are strictly decreasing, the $I_e$'s can be chosen so that

- $\sum_{e \in q} |I_e| \leq \mathrm{birth}(p_0) - \mathrm{birth}(p_s)$, where
- for every edge $e$ at which no delay was recorded, $I_e = [r, r]$ where $r$ is the rank of the packet that passed through $e$, and
- for any two edges $e$ and $e'$ on the delay path $q$, $I_e \cap I'_e = \emptyset$.

On the other hand, the number of packets that can be injected during $I_e$ with a path through $e$ is at most $\lambda(|I_e| + w)$. Hence, the number of packets that can be injected during the time span of the delay sequence, $\bigcup_e I_e$, is at most

$$\sum_{e \in q} \lambda(|I_e| + w) \leq \sum_{e \in q} |I_e| + \lambda w L \leq (\mathrm{birth}(p_0) - \mathrm{birth}(p_s)) + \lambda w L \ ,$$

which contradicts the fact that there have to be more than $(\mathrm{birth}(p_0) - \mathrm{birth}(p_s)) + \lambda w L$ delays along the delay path. □

Though LIS works fine if there is always sufficient space for packet buffering, it can run into problems if this is not the case. Here, the *enforced longest in system* (or ELIS) protocol can be used.

### Enforced longest in system

ELIS is a variant of LIS that was developed for leveled graphs to enforce that packets arrive at their destinations in a strictly ordered way based on their age. This can be important, for example, for consistent data updates.

We assume that every node has a queue of size $q$ for every *incoming* edge. The packets are assigned ranks in order to decide which packet is preferred in case of contention. For each packet $p$, let $\mathrm{birth}(p)$

denote the time step at which $p$ was injected. The rank of $p$ is set to $\mathrm{birth}(p)$ plus some small value $x$ from the interval $[0, \kappa)$, for some $\kappa < 1$, where $x$ is chosen such that each packet has its own, unique rank (e.g., by using the identification number of the process that injected the packet). Packets with smaller ranks, i.e., older packets, are always preferred against packets with higher rank, i.e., younger packets. Special ghost packets help the algorithm to maintain the following invariant:

*A packet is routed along an edge only after all the other packets with lower ranks that must pass through the edge have done so.*

Suppose that we have a network with $L+1$ levels, numbered from 0 to $L$. For every packet injected into the system, the adversary has to provide a path that starts at some source node in level $i$, ends at some destination in level $j > i$, and that only uses edges that go from some level $k$ to some level $k+1$. In order to give time for initializing the network, we assume that packet injections on level $k$ do not start before time step $k$.

The following algorithm is executed for each outgoing link $e$ of a node $v$ on level $k$ in each time step $t \geq k$ (recall that each edge buffer can hold up to $q$ packets):

- Let $r$ denote the minimum rank of a packet that is stored in one of $v$'s buffers and that aims to pass edge $e$. If there is no such packet then $r = \infty$.
- Let $g$ denote the minimum over all ranks of packets or ghost packets that arrived at $v$ at the beginning of step $t$. If there is no such packet (as $v$ is a node without incoming edges, e.g., on level 0) then $g$ is set to $t + \kappa$.
- if $r < g$ then
  if the buffer of $e$ contains less than $q$ packets at the beginning of step $t$ then
  forward the (unique) packet with rank $r$ along $e$
  else send a ghost packet with rank $r$ along $e$
  else send a ghost packet with rank $g$ along $e$.

Ghost packets are discarded as soon as they are delayed in a step. Thus, they never block the buffer for following packets. The role of the ghost packets is to enforce the invariant given above. They make sure that edges in larger levels are always kept informed about which is the largest rank of a packet at a smaller level that may still want to traverse that edge.

**Theorem 3.5 ([4])** *Let $L$ denote the depth of the leveled network and $q$ be the size of the edge buffers. For any $(w, \lambda)$-bounded adversary with $w \leq (q-2)/(2\lambda)$ and $0 \leq \lambda \leq 1$, ELIS is stable, and every packet reaches its destination in at most $(1 + \lambda \cdot w)L$ time steps.*

## 3.7 Routing in arbitrary networks

Suppose that we want to route packets efficiently in an arbitrary network. If we make sure that all paths used by the packets are shortest possible (or short-cut free) paths (as this is the case for $x-y$-routing in the mesh), we can use the *growing rank protocol* or GRP to achieve a low delay. The GRP is basically a combination of LIS and NTO. The initial rank of a packet is equal to its injection time. Every time a packet crosses an edge, its rank is increased by some parameter $\sigma$. For any edge queue in the system,

GRP gives preference to the packet with smallest rank. Ties are broken based on the source ID of the packet (or other unique information).

Using a stochastic injection model ($\lambda$ represents the average number of packets injected at a step that traverse an edge, and each injection process at a node injects packets independently from other processes and other steps), the following result can be shown for the GRP.

**Theorem 3.6 ([4])** *Suppose all routing paths are shortest paths. Then GRP is stable for any injection rate $\lambda$ up to some $1 - \epsilon$ with $\epsilon = \Theta((\log \sigma)/\sqrt{\sigma})$. Furthermore, the routing time for any packet $p$ that has to travel along a routing path of length $d$ is $O(\sigma \cdot d)$, expected, and $O(\sigma \cdot (d + \log N))$, with high probability.*

# References


[1] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. of the 37th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 380–389, 1996.

[2] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queueing theory. In *Proc. of the 28th ACM Symp. on Theory of Computing (STOC)*, pages 376–385, 1996.

[3] M. Bramson. Convergence to equilibria for fluid models of fifo queueing networks. *Queueing Systems*, 22:5–45, 1996.

[4] C. Scheideler and B. Vöcking. From static to dynamic routing: Efficient transformations of store-and-forward protocols. In *Proc. of the 31st ACM Symp. on Theory of Computing (STOC)*, pages 215–224, 1999.