Shrinking the Timescales at Which Congestion-control Operates

Vishnu Konda and **Jasleen Kaur**
Technical Report # TR08-011
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

TCP congestion-control is fairly ineffieicnt in achieving high throughput in high-speed and dynamic-bandwidth environments. The main culprit is the *slow bandwidth-search process* used by TCP, which may take up to several thousands of round-trip times (RTTs) in searching for and acquiring the end-to-end spare bandwidth. Even the recently-proposed "high-speed" transport protocols may take hundreds of RTTs for this.

In this paper, we design a new approach for congestion-control that allows TCP connections to boldly search for, and adapt to, the available bandwidth within a single RTT. Our approach relies on carefully orchestrated packet sending times and estimates the available bandwidth based on the delays experienced by these. We instantiate our new protocol, referred to as RAPID, using mechanisms that promote efficiency, queue-friendliness, and fairness. Our experimental evaluations on gigabit networks indicate that RAPID: (i) converges to an updated value of bandwidth within 1-4 RTTs; (ii) helps maintain fairly small queues; (iii) has negligible impact on regular TCP traffic; and (iv) exhibits excellent intra-protocol fairness among co-existing RAPID transfers. The rate-based design allows RAPID to be the first protocol that truly does not suffer from RTT-unfairness.

1 Introduction

"Congestion Control" can be easily listed among the top-10 networking problems of the past two decades. And indeed, why not? A congestion-control protocol has no simple task—it has to adaptively discover the end-to-end spare bandwidth available to a transfer in a quick and non-intrusive manner. Simultaneously achieving these properties turns out to be a significant challenge, especially for an end-to-end protocol that receives no explicit feedback from routers/switches. Indeed, the dominant end-to-end transport protocol, TCP NewReno [1], has been shown to be abysmally slow in discovering the spare bandwidth, especially in high-speed networks and on paths that experience dynamic bandwidth.

Several alternate protocols have been proposed to address this limitation. However, as discussed in Section 2, most of these protocols struggle to remain non-intrusive to other network traffic while achieving speed—consequently, these designs are still quite sluggish in probing for spare bandwidth. In particular, we show that even the so-called "high-speed" protocols may take hundreds-to-thousands of round-trip times (RTTs) to converge to a stable sending rate in gigabit networks.

In this paper, inspired by recent advances in the field of bandwidth estimation, we propose the idea that the sluggishness of transport protocols can be eliminated, without overloading the network, if we *limit the impact of probing for spare bandwidth*. In particular, we rely on carefully orchestrated packet sending times and use the relative delays experienced by the packets, to probe for an exponentially-wide range of rates within a single RTT—the impact of probing is limited by ensuring that the average sending rate is not high. We use this idea to design a novel approach, referred to as *RAPID Congestion Control (RAPID)*, that exhibits three significantly desirable characteristics. Most notably, it reduces the time it takes a transport protocol to acquire freshly-available bandwidth by more than an order of magnitude. Equally significantly, by relying on a delay-based congestion-detection strategy, the protocol ensures (i) that it is friendly to transfers that use regular loss-based TCP, and (ii) that packet queues at bottleneck links are small and transient. Finally, due to its speed, RAPID also exhibits excellent intra-protocol fairness properties at small-to-medium timescales, even in network environments with heterogeneous RTTs.

Table 1 summarizes several notations used throughout the paper—we use the terms "packets" and "segments" interchangeably. In the rest of this paper, we describe the sluggishness of existing protocols and present our key insight in Section 2. We describe the RAPID protocol mechanisms in Section 3 and present performance evaluation results in Section 4. We conclude in Section 5.

avail-bw, AB	available bandwidth
$ABest$	the AB estimate returned by the receiver
p-stream	multi-rate probe stream
N	the number of packets in a p-stream
P	packet size
r_{avg}	the average sending rate of a p-stream
r_i	the sending rate of the $(i+1)^{th}$ packet in a p-stream
m	the ratio of $\frac{r_{i+1}}{r_i}$ for all $i \in [1, N-1]$
τ	the duration over which an increase in AB is adopted

Table 1: Notations Used in the Paper

Protocol	Search-step (per-RTT increase in probe-rate)	Feedback-metric	Experimentally-observed time for acquiring AB = 1 Gbps
NewReno	Additive Increase	Packet Loss	$\sim$ 7400 RTT
HighSpeed [2]	Multiplicative Increase	Packet Loss	$\sim$ 250 RTTs
CUBIC [3, 4]	Additive/Binary-search Increase	Packet Loss	$\sim$ 100 RTTs
FAST [5]	Additive Increase	Packet Delays (and Loss)	$\sim$ 50–100 RTTs
RAPID	Exponential Increase	Delays (and Loss)	$\sim$ 4 RTTs

Table 2: Time taken to acquire an AB of 1 Gbps by different protocols (see Section 4.1)

2 Problem Formulation

2.1 The Problem: Looong Feedback Loop

To understand the sluggishness issue, consider an end-to-end congestion-control protocol in steady-state—the protocol continuously operates a 2-step AB-search cycle: in the *search-step*, it successively probes for (by sending at) larger data sending rates. For each rate probed at, it examines performance feedback such as packet loss and high end-to-end delays that arrives after an RTT-worth of delay—this helps it estimate when the most recent sending rate was higher than the avail-bw.[1] When a higher rate is reached, the *reduction-step* of the protocol reduces the sending rate to a lower value and switches back to the search-step. The speed with which a single loop of this 2-step cycle is executed fundamentally determines how quickly a protocol can acquire and adapt to changes in the avail-bw.

Note that the search-step can not be executed at a timescale smaller than an RTT—performance feedback can arrive no earlier than this time. Unfortunately, most existing protocols execute this step at timescales much larger than this. This is because, primarily driven by the goal of not overloading the network, *all* previously-proposed protocols adopt two limiting design features:

1. *Only a single (larger) sending rate is probed for over a round-trip time.*

   The protocol probes for a candidate sending rate and then waits for performance feedback, which arrives after an RTT-worth of delay. A new and larger sending rate (probeRate) is then probed for only during the next RTT time interval. This is true for all previous protocols, including recent ones, such as HighSpeed TCP, FAST, Scalable, CUBIC, and PCP [2, 3, 4, 5, 6, 7, 8]. This legacy design decision is perhaps motivated by the fact that unless the single rate is deemed acceptable (not too high), other rates should not be probed for.

2. *The new rate probed for (probeRate) is not significantly larger than the previous sending rate (prevRate).*

   This feature is adopted primarily to prevent a single transfer from overloading the network, in case the previous rate was quite close to the avail-bw. Existing protocols differ in how the new probing rate relates to the previous rate—for most protocols, the ratio of probeRate/prevRate is only slightly larger than 1 in high-speed networks. For instance, these two quantities are additively related in NewReno and FAST, as in: probeRate = prevRate + α*MSS/RTT, where MSS is the maximum segment size allowed on the path, and α is set by default to 1 and 200 in the two protocols, respectively. Scalable, HighSpeed, and CUBIC rely on a multiplicative relation as in: probeRate = γ*prevRate; however, γ is again

[1] Different protocols differ in how the successive probing rates relate to the current rate, as well as in the performance measures they use as feedback. Table 2 summarizes these differences for some prominent protocols.

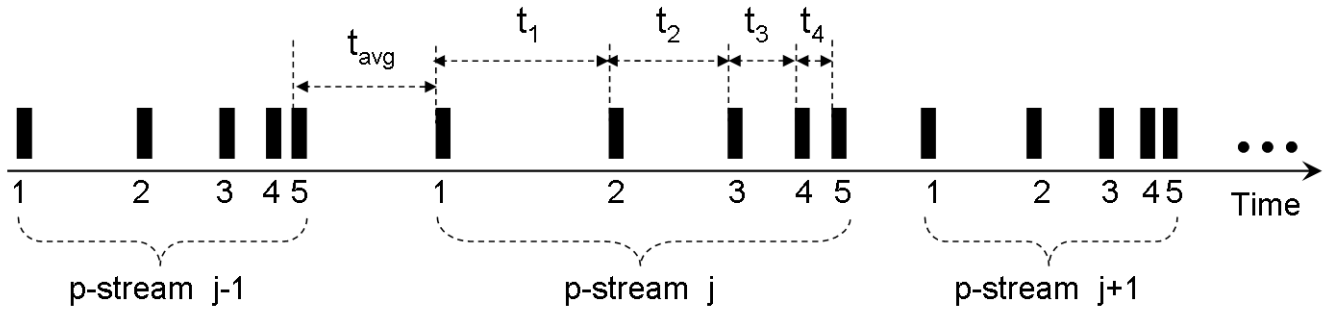


Figure 1: Illustration of a p-stream (here, $t_i = \frac{P}{r_i}$ and $t_{avg} = \frac{P}{r_{avg}}$)

restricted to small values.[2]

In addition to these two limitations, protocols that rely only on packet losses as an indicator of congestion suffer from yet another problem—these protocols may have to reach a sending rate much higher than the avail-bw (and oscillate several times around it) before stabilizing to it. This is because a loss-based protocol would need to fill up buffers in the bottleneck routers and suffer a loss before it can detect that it has acquired (and surpassed) the avail-bw.

As a result of these design limitations, most protocols—even the recent ones designed for high-speed networks [2, 7, 5, 8, 3, 4]—take a fairly long time for converging to the avail-bw. Table 2 lists, for different protocols, the experimentally-observed times taken by a single transfer for acquiring a spare capacity of 1 Gbps (say, after experiencing a packet loss).[3] We find that even the fastest of high-speed protocols can take hundreds of RTTs for converging to the avail-bw.

2.2 Key Insight: Limit Probing Volume

We believe that all of the above design limitations can be done away with, without overloading a network or inducing buffer overflow, if one *limits the volume (and impact) of probing for a larger sending rate*. In fact, we claim that a protocol can boldly probe for an exponentially-wide range of candidate sending rates within a single RTT, if it: (i) uses carefully-orchestrated inter-packet gaps, and (ii) relies on relative packet delays for estimating avail-bw. Any associated overloading impact can be avoided by using the following guidelines:

1. *Achieve a rapid AB search:* Probe for an *exponentially-wide* range of candidate sending rates within a single RTT. However, send extremely small probes (of one packet each) at each candidate rate in order to limit to very small timescales the overloading impact of the large rates.

2. *Avoid persistently overloading the network path:* Ensure that the *average* rate of packet transmission does not exceed the most-recently discovered estimate of avail-bw. This implies that some of the rates in the above-suggested exponential range will be smaller than this estimate, and some will be larger.

We use these ideas to design a new protocol, referred to as RAPID Congestion Control (RAPID), and show that it can adapt to fairly large changes in AB within 1-4 RTTs. Furthermore, RAPID can avoid persistently large router queues due to its primary focus on avoiding network overload. The benefits of the protocol are especially significant in dynamic bandwidth environments and high-speed networks.

It is important to note that several protocols such as Vegas, FAST, and PCP also rely on packet delays (instead of packet losses) for detecting congestion; however, *none* of these protocols exploit inter-packet gaps to probe for a wide range of rates within a single RTT.[4]

We next present the basic mechanisms used in RAPID.

[2]BIC [4] relies on a combination of additive-increase and a binary search based method after the AB is discovered for the first time—here, the ratio probeRate/prevRate depends on the past probing history.

[3]These experiments were run on the ns-2 simulator, and a packet size of 1040 B was used—see Section 4.1 for details.

[4]The rate-based PCP protocol [6] also adopts the idea of "limiting the probe volume". However, it probes for only a single larger rate per RTT and has an AB-search speed similar in magnitude to that of existing protocols. In all fairness, the protocol primarily focusses on minimizing response times in under-utilized networks—it has not even been evaluated for large transfers in high-speed networks.

3 RAPID Congestion Control

Unlike many congestion-control protocols, RAPID employs a multi-rate based transmission policy and relies on relative packet delays for estimating avail-bw. While the RAPID design is motivated by the primary goal of shrinking the timescales at which congestion-control operates, several equally-important goals are given due consideration in the design process [9]. Most significantly, a RAPID network strives to (i) maintain a *low buffer occupancy* at congested router links, (ii) achieve a *fair sending rate* when several RAPID transfers co-exist, and (iii) remain *friendly to regular low-speed TCP transfers*. Below, we describe the basic mechanisms used for achieving each of these.

3.1 Acquiring AB Within a Few RTTs

3.1.1 Rate-based packet transmission at the sender

When there is sufficient data to send, a RAPID sender continuously transmits data in logical groups of N packets each, referred to as a *multi-rate probe stream (p-stream)*.[5] The i-th packet in a p-stream is sent at a rate of r_{i-1}; this also implies that the sending times of packets i and $i-1$ differ by $\frac{P}{r_{i-1}}$, where P is the size of packet i (see Figure 1). The sender explicitly controls/manages the average sending rate of a p-stream, referred to as r_{avg}, which is given by:

$$r_{avg} = \frac{N-1}{\frac{1}{r_1} + \frac{1}{r_2} + \ldots + \frac{1}{r_{N-1}}} \tag{1}$$

Further, for all $i > 1$, $r_i > r_{i-1}$.[6]

3.1.2 AB-estimation analysis at the receiver

We observe the inter-packet gaps in a p-stream at the receiver, and use these for estimating avail-bw in the same manner as the PathChirp bandwidth estimation tool [10]. Recent evaluations have shown that PathChirp estimates avail-bw with good accuracy in multi-hop settings, while incurring the least overhead among existing tools [11].

Like PathChirp, when a RAPID receiver receives all packets of a p-stream, it computes the AB by looking for increasing trends in the intended inter-packet spacings. This analysis relies on the concept of *self-induced congestion*. Intuitively, if q_i is the queuing delay experienced at a bottleneck link by the i-th packet in a p-stream, then:

$$q_k = 0, \quad \text{if } r_k \leq AB \tag{2}$$

$$q_k > q_{k-1}, \quad \text{otherwise} \tag{3}$$

Thus, if i^* is the first packet in a p-stream such that $r_{i^*-1} \geq AB$, then each of the packets $[i^*, \ldots, N]$ will queue up behind its previous packet at the bottleneck link (since $r_i > r_{i-1}$, for all $i > 1$)—due to this "self-congestion", each of these packets will experience a larger one-way delay (and a larger increase in the pre-set inter-packet gap) than its predecessor. Thus, the smallest rate r_{i^*-1} at which the receiver observes an increasing trend in the inter-packet gaps can be used to compute an estimate of the current avail-bw as: $ABest = r_{i^*-1}$.[7] The actual analysis uses several heuristics to account for bursty cross-traffic—we refer the reader to [10] for details and the precise formulation.

The receiver encodes the value of $ABest$ obtained from the most recent p-stream in the acknowledgments sent to the sender.

3.1.3 Transmitting in a non-overloading, responsive manner

When the sender receives an $ABest$ value, it updates the r_{avg} of the next p-stream as: $r_{avg} = ABest$. Thus, the transfer acquires an *average* sending rate equal to the estimated avail-bw within an RTT. The sender then selects an appropriate set of rates, $r_1, \ldots, r_{N-1}$, for the next p-stream such that the average of these is equal to r_{avg}, as computed in Eqn (1).

The above mechanism helps simultaneously achieve two desirable properties. First, by setting r_{avg} equal to the estimated avail-bw, RAPID helps to ensure that the average load on the bottleneck link does not exceed its capacity—this is crucial for

[5] Unlike window-based protocols, the sender does not stall, waiting for acknowledgements. Also, if data for only $k < N$ packets is available, a p-stream of k packets is sent instead. Note that if $k \leq 2$, the sender sends these packets at a uniform rate of r_{avg}.

[6] The gap between the first packet of a p-stream and the last packet of the previous p-stream is set to r_{avg}. This can also be stated as: $r_0 = r_{avg}$. Typically, $r_0 > r_1$.

[7] If no increasing trend is detected in a p-stream, r_{N-1} is taken as the AB estimate. Also, if the increasing trends starts at the first or second packet ($i^* \leq 2$), $\frac{r_1}{2}$ is returned as the AB estimate.

# of RTTs	r_{avg}	Rates that can be probed for
RTT 0	x_0	$0.45x_0 - 3.22x_0$
RTT 1	$3.22x_0$	$1.5x_0 - 10.7x_0$
RTT 2	$10.7x_0$	$4.7x_0 - 33.4x_0$
RTT 3	$33.4x_0$	$15x_0 - 108x_0$
RTT 4	$108x_0$	$48x_0 - 346x_0$
RTT 5	$346x_0$	$156x_0 - 1115x_0$

Table 3: AB-acquiring speed by a RAPID sender

maintaining small and transient queues at the bottleneck links. Second, by selecting a set of rates which includes values larger (r_{N-1}) as well as smaller (r_1) than r_{avg}, a p-stream is able to simultaneously probe for both increase and decrease in the current end-to-end avail-bw—this greatly helps the RAPID sender in quickly detecting and adapting to changes in AB.

3.1.4 Setting $[r_1, \ldots, r_{N-1}]$ (speeding up the search process)

Each p-stream probes for the range of sending rates given by: $[r_1, \ldots, r_{N-1}]$. Note that for a given r_{avg} and N, there are infinite choices for this set of rates, such that they satisfy Eqn (1). However, the larger is the ratio $\frac{r_{N-1}}{r_1}$, the faster would be the AB-search process—this is because a single p-stream now probes for a wider range of rates.

So for instance, while these rates could be additively-related, a faster search will be obtained by using a multiplicative-relation as in:

$$r_i = m^{i-1} * r_1, \qquad 1 < i < N \tag{4}$$

RAPID adopts the above relation. Given r_{avg}, Eqns (1) and (4) can be used to compute r_1 as:

$$r_1 = \frac{m^{N-1} - 1}{(N-1)(m-1)m^{N-2}}\, r_{avg} \tag{5}$$

$r_2, \ldots, r_{N-1}$ can then be computed from Eqns (4) and (5).

Selecting m and N For a given N, two conflicting considerations guide the choice of m. A larger value of m would also result in a larger ratio of $\frac{r_{N-1}}{r_1}$, and would improve the speed as well as adaptivity of the AB-search process. A smaller m, on the other hand, would result in a smaller ratio of $\frac{r_{i+1}}{r_i}$—this would result in a finer rate granularity with which the avail-bw is probed. A coarse-granularity estimate of avail-bw would prevent a collection of RAPID senders from efficiently utilizing the bottleneck link.

The selection of N is also faced by two opposing considerations. For a given m, a larger value of N would improve the AB-search range ($\frac{r_{N-1}}{r_1}$). However, a larger p-stream would also be more intrusive to cross-traffic (more packets would be sent at a rate larger than r_{avg}) at bottleneck links.

RAPID adopts the default values of $N = 30$ and $m = 1.07$ (7% granularity in rates probed for). These values have been selected after controlled experimentation under diverse topology and traffic settings–Details of this sensitivity analysis can be found in Appendix B.

The above choices of m and N yield: $r_1 \approx 0.45 * r_{avg}$ and $r_{N-1} \approx 3.22 * r_{avg}$. This enables a RAPID sender to probe for freshly-available spare bandwidth spanning several orders of magnitude within a few RTTs (see Table 3).

3.1.5 Achieving a Quick-yet-Slow Start

RAPID faces a similar dilemma as all congestion-control protocols—how to obtain the initial $ABest$ (or the initial r_{avg}) for a new transfer? The main concern here is that the initial choice of r_{avg} may be too high for a given network path. We address this issue by being only as aggressive as the TCP Slow-Start mechanism (which is also adopted by most other protocols). Specifically, in slow-start, a RAPID sender sends only as many packets in an RTT as would a TCP transfer—fortunately, the ability of a p-stream to probe for multiple rates within an RTT makes the RAPID slow-start terminate much earlier than other protocols.

In the slow-start phase, a RAPID sender sends only a *single* p-stream over an RTT. Further, we initialize $N = 2$, and double the value of N over successive RTTs, up to a maximum value of 16.[8] For constructing the p-streams, we use a multiplicative

[8]Note that this process is no more aggressive than the slow-start adopted by most protocols, which multiplicatively increases the number of packets sent over an RTT in exactly the same manner (and start with an initial value of 1 or 2 segments). Also note that the slow-start threshold is usually set to a much higher value than 16 segments.

# of RTTs	N	Range of rates probed for
RTT 1	2	0 - 100 Kbps
RTT 2	4	100 - 800 Kbps
RTT 3	8	800 Kbps - 102 Mbps
RTT 4	16	102 Mbps - 3342 Gbps

Table 4: Speed of RAPID Slow-Start

factor of $m = 2$ throughout slow-start—thus, the granularity with which a RAPID sender probes for avail-bw during slow-start is coarse, but is the same as all existing protocols (that double their window size every RTT) [1, 12]. Unlike most protocols, however, the AB-search speed is quite high—since it relies on an AB-estimation analysis, a RAPID transfer in slow-start discovers avail-bw much earlier than any other protocol.

For a new RAPID transfer, we initialize r_{avg} to a small value (100Kbps)—this implies that, in the first RTT, the transfer sends $N = 2$ packets at a rate of $r_1 = 100Kbps$. If the receiver returns $ABest < r_{N-1}$, we exit slow-start; if not (which implies, $ABest = r_{N-1}$), we double the value of N, set $r_1 = ABest$, and send the next p-stream.[9] This process is repeated till the receiver returns an $ABest$ value less than r_{N-1}. Following this, we switch to the steady-state RAPID mode, in which $m = 1.07$ and $N = 30$.

Table 4 illustrates the number of RTTs that a single RAPID transfer would take to probe for different amounts of avail-bw in slow-start—an RAPID sender can probe for more than 1 Tbps in just 4 RTTs, while being no more aggressive than TCP slow-start! In Section 4, we show that existing protocols take much longer.

3.2 Achieving Fairness Among Co-existing RAPID Transfers

The RAPID design should justifiably raise fairness concerns when multiple RAPID transfers share a bottleneck link: *how do the p-streams of different transfers interact—do the transfers obtain a fair share of the avail-bw?* In this section, we address two sources of unfairness that have been identified in the literature [13, 14].

3.2.1 RTT-unfairness

Most window-based congestion control protocols have been shown, both experimentally and analytically, to suffer from *RTT-unfairness*—transfers with a long RTT get a lower throughput than short-RTT transfers [13, 14, 5]. This happens because window-based protocols update their sending rates *once per RTT*—in heterogeneous RTT environments, this RTT-dependence results in differences in the rate updating frequency as well as rate increments, and results in a bias against long RTT transfers [14].

Fortunately, the rate-based design of RAPID is not influenced by the value of RTT—a RAPID sender continuously send p-streams, for both long and short RTT transfers. The rate-updating frequency (once per p-stream) as well as rate increment (determined by $ABest$) are independent of the RTT—consequently, and by design, *RAPID truly does not suffer from RTT-unfairness.* Our experiments in Section 4.2 confirm this.

3.2.2 Bias due to rate-proportional feedback frequency

As described so far, however, the RAPID design is likely to be unfair for a different reason. The rate at which a RAPID sender receives $ABest$ values (once per p-stream), is directly proportional to the current value of r_{avg} and is given by: $\frac{r_{avg}}{N*P}$, where P is the packet size. Consequently, a RAPID transfer that has achieved a larger sending rate will receive more frequent notifications of any spare capacity that becomes available, than a co-existing transfer with a lower r_{avg}—thus, the former is likely to attain an even higher sending rate than the latter. In fact, such differences in the rate at which feedback arrives have been shown to result in unfair throughput allocations even in window-based protocols [14].

To ensure that co-existing RAPID transfers achieve a fair-share of the avail-bw, we explicitly *equalize the rate at which RAPID senders converge to the* $ABest$ values they receive. For this, we select a parameter τ that represents the common (large) time interval over which any RAPID sender, independent of its sending rate, would converge to an increase in avail-bw. Specifically, when a sender receives an $ABest$ that is larger than its current r_{avg}, it computes a new value as:

$$r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg}) \tag{6}$$

[9]Note that in slow-start, RAPID sets r_1 (and not r_{avg}) equal to $ABest$.

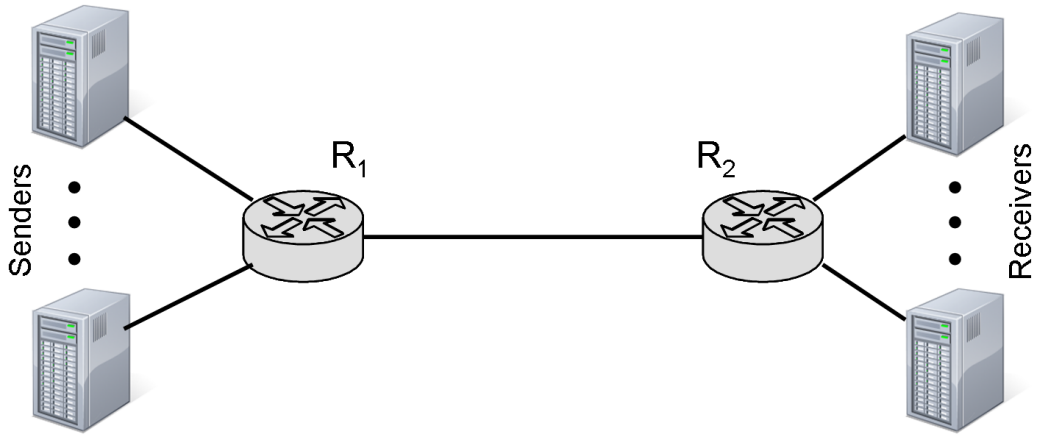


Figure 2: Experimental Topology

where l is the duration of the most-recent p-stream, which is given by: $l = \frac{N*P}{r_{avg}}$. The above filter is used to repeatedly update r_{avg} for all subsequent p-streams, till r_{avg} converges to $ABest$. The effect of this filtering mechanism is that it would take a RAPID sender roughly τ time units (or $\frac{\tau}{l}$ p-streams) for converging to an updated value of $ABest$—for a transfer with a small value of r_{avg}, $\frac{l}{\tau}$ will be close to 1 and the sender will immediately adopt the $ABest$ as its new r_{avg}. Thus, even though the *feedback frequency* depends on the sending rate of a RAPID transfer, the *rate-increment frequency* does not.[10]. Our experiments in Section 4.2 show that this mechanism helps achieve excellent fairness among RAPID transfers. τ is set by default to 200 ms.

3.3 Remaining TCP-Friendly

RAPID, like FAST, is quite non-intrusive to regular low-speed TCP NewReno transfers. The prime reason for this is that these protocols rely on increased packet delays for detecting network congestion, whereas TCP reduces its sending rate only on witnessing packet losses. When a router carrying both TCP and RAPID transfers gets congested, the RAPID transfers would respond to the congestion (and reduce their sending rates) much earlier than the TCP transfers would. This would ensure that the performance of the low-speed TCP transfers is not significantly impacted due to the presence of high-speed RAPID transfers.

The downside is that in the presence of long-lived TCP transfers, RAPID transfers would obtain lower throughput than the former. However, this problem plagues any network that simultaneously runs fundamentally different congestion-control protocols [5]—a simple solution is to provision routers with separate queues for traffic from different protocols.

We next experimentally evaluate how well the above mechanisms achieve the stated goals for RAPID.

4 Experimental Evaluation

There are at least three types of concerns that the RAPID design is likely to raise in the minds of a reader: *does a short p-stream really help RAPID in accurately estimating avail-bw, especially in high-speed networks? RAPID seems aggressive in the rates it probes for—is it really friendly to router queues and competing low-speed TCP traffic? And, how do the AB-estimation processes of co-existing RAPID transfers interact—don't they interfere with each other (and do they get a fair share of the avail-bw)?* In this section, we address these concerns by experimentally evaluating RAPID. We have also experimentally studied: (i) sensitivity of RAPID to paramter settings, (ii) fairness among multiple high-speed protocols, and (iii) RAPID performance in multi-hop settings the details of which are given in appendix.

We use the ns-2 simulator [15] for our evaluations. We have implemented RAPID in ns-2.33 and have re-used the NewReno code base for dealing with loss detection and recovery. We also use publicly-available ns-2 implementations of three other protocols for comparison, namely HighSpeed TCP, CUBIC, and FAST TCP [16]—the first two are loss-based protocols with fairly different window-growth functions; FAST is a high-speed version of the delay-based Vegas protocol.[11]

[10]For stability reasons, we do not use the above filter when the avail-bw decreases—details are provided in Appendix B

[11]We rely on the default parameter settings configuration for all protocols, other than for FAST. FAST exhibited severe oscillations for the experiments of Section 4.1 when run with default ns-2 parameters—for the evaluations presented here, we have instead relied on parameters used in sample test scripts supplied along with the implementation [16] (experiments with other parameter settings are included in appendix B.

For several of our simulations, we rely on a simple dumbbell topology in which multiple sources are aggregated at a single bottleneck link (R_1–R_2 in Fig 2)—this bottleneck link is the only link shared by co-existing transfers.[12]. All links other than the bottleneck link have a transmission capacity of 10 Gbps—the bottleneck link capacity is set by default to 1 Gbps, but is reduced for some experiments. All links are provisioned with a delay-bandwidth product (DBP) worth of buffers, where the delay is the average end-to-end propagation delay for transfers (specified for each experiment), and the bandwidth is that of the bottleneck link (also specified). We set the maximum size of each network-layer packet to 1040 B.

The main performance statistics we are interested in are throughput obtained by transfers instantiated between the sender and receiver nodes as well as the queue build-up on the bottleneck link. We sample each of these statistics periodically at regular intervals of 50 ms each. In what follows, we summarize our experiments and observations.

4.1 Speed of Acquiring Spare Bandwidth

There are at least two types of scenarios in a high-speed network where the ability of a congestion-control protocol to acquire spare bandwidth quickly is crucial. The first is during slow-start, when a transfer begins without any advance knowledge of the avail-bw, and the second is when the avail-bw suddenly changes by a large amount (perhaps due to the arrival of additional traffic). In our first set of experiments, we compare the performance of RAPID with that of other protocols by simulating examples of both of the above scenarios. For all of the experiments in this section, we simulate a 1Gbps bottleneck link (R_1–R_2 in Fig 2) and set the end-to-end propagation delay to 100ms.

4.1.1 Slow-start in High-speed Networks

In the first set of experiments, we simulate a single transfer from a sender node to a receiver node (see Fig 2). Each experiment uses a different underlying congestion-control protocol. Fig 3(a) plots as a function of time, the throughput obtained by the transfer with different protocols. Fig 3(b) plots the queue buildup observed over time at the bottleneck router. We find that:

- HighSpeed TCP takes the longest time (250 RTTs) for acquiring a sending rate of 1 Gbps. And once it acquires that sending rate, being a loss-based protocol, it starts filling up the bottleneck queues.[13]

  CUBIC is faster, initially acquiring 1 Gbps after merely 10 RTTs—but it also fills up the bottleneck buffers as quickly, later leading to packet losses and a drop in throughput. It eventually stabilizes to 1 Gbps after 110 RTTs. Being a loss-based protocol, it also maintains nearly full buffers at the bottleneck link.

- FAST TCP is less aggressive in filling up router queues and is able to acquire avail-bw faster—within 50 RTTs. However, even a single transfer can regularly fill up thousands of packet buffers in the bottleneck router.

- RAPID is significantly faster than all other protocols, acquiring the 1 Gbps bandwidth in just 4 RTTs—this is exactly as was predicted by the design in Section 3.1.5. Furthermore, the single RAPID transfer induces very low queuing (less than 20 packets) on the bottleneck router.

4.1.2 High-speed Networks with Dynamic AB

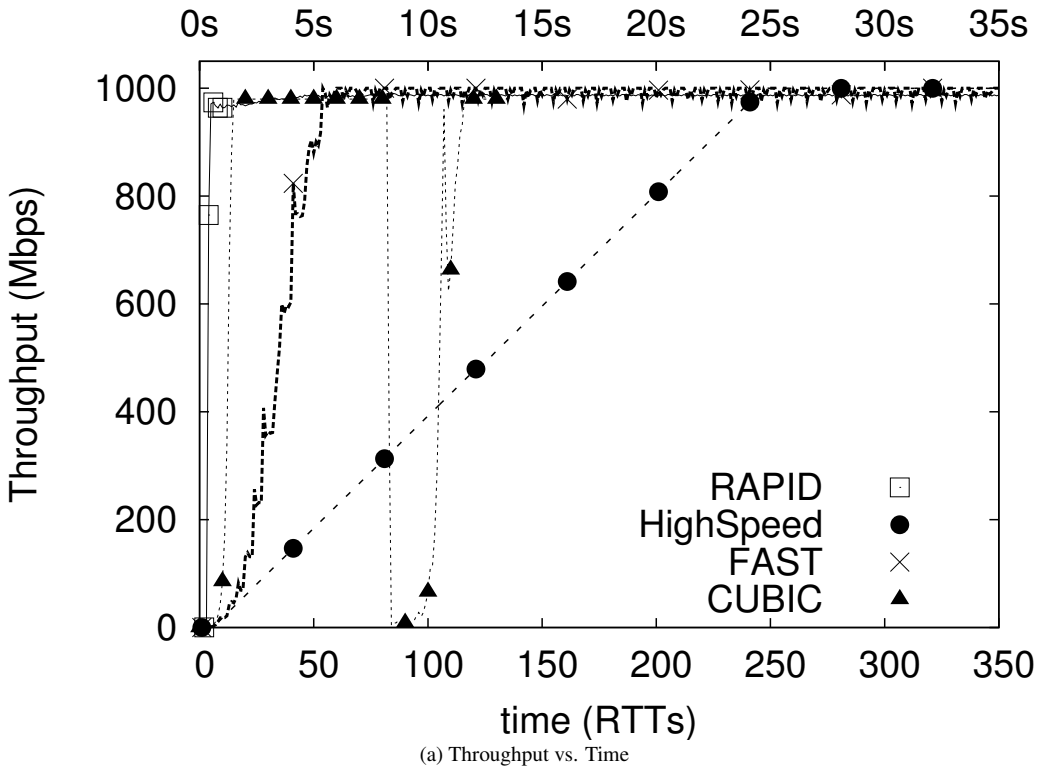
In the second set of experiments, we simulate a network for 500 seconds, and introduce 4 constant-bit-rate (cbr) traffic streams on the bottleneck link according to the following schedule: *cbr-1* exists from 50-400 seconds, *cbr-2* exists from 100-150 seconds, *cbr-3* exists from 250-350 seconds, and *cbr-4* exists for a small duration from 460-462 seconds. Each cbr stream has a bit-rate of 200 Mbps. The spare bandwidth left on the network is plotted in the top-row plots of Figs 4(a)-(d) using a faint dotted line.

We use this setup to run a set of experiments in which we introduce a single long-lived transfer at time 1 second, and respectively, run it over CUBIC, HighSpeed, FAST, and RAPID. The throughput obtained by the transfers and the router queue sizes are plotted, respectively, in the top and bottom rows of Fig 4. We find that:

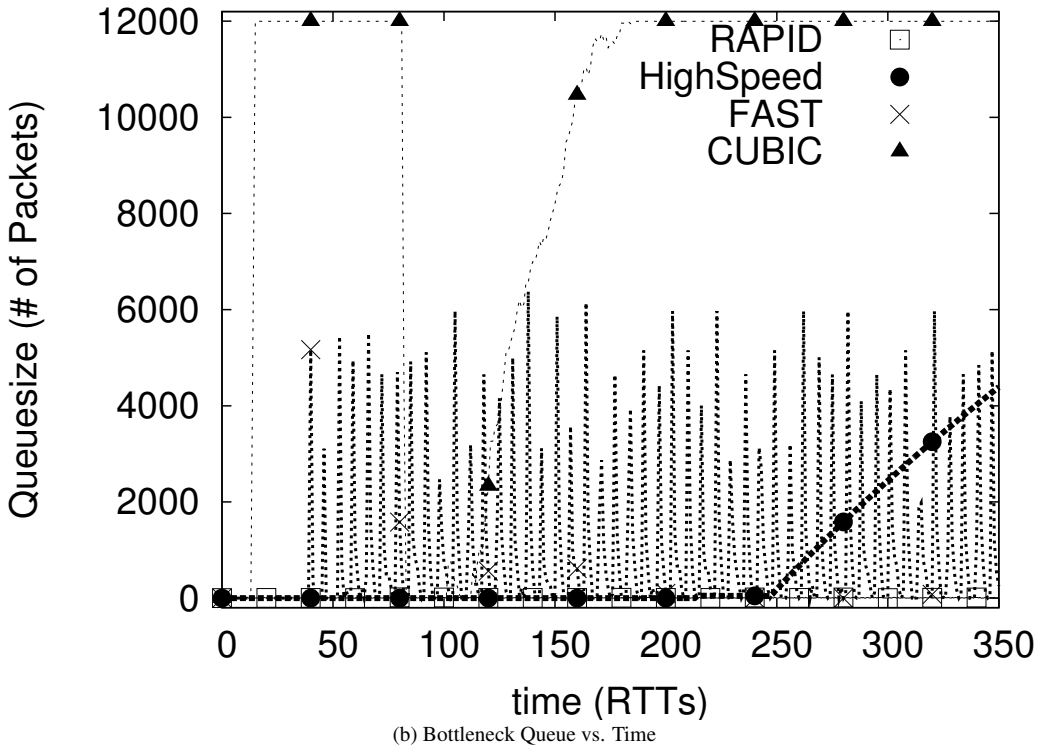
1. HighSpeed and CUBIC, which are both loss-based protocols, experience heavy packet losses when the avail-bw reduces suddenly (and even when it does not change). This is because a loss-based protocol induces persistent queuing in the bottleneck buffers—any sudden decrease in AB overflows the buffers causing multiple packet losses. When this happens,

[12] An experiment with multiple bottlenecks can be found in Appendix C

[13] We also ran this experiment with the NewReno protocol, which took around 7400 RTTs to acquire the 1 Gbps avail-bw.

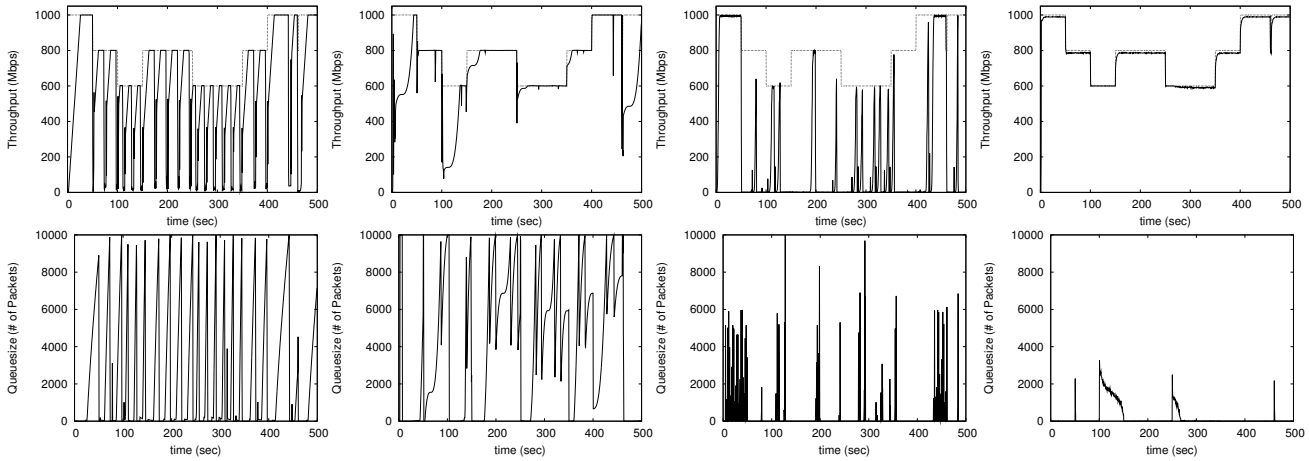


(a) Throughput vs. Time



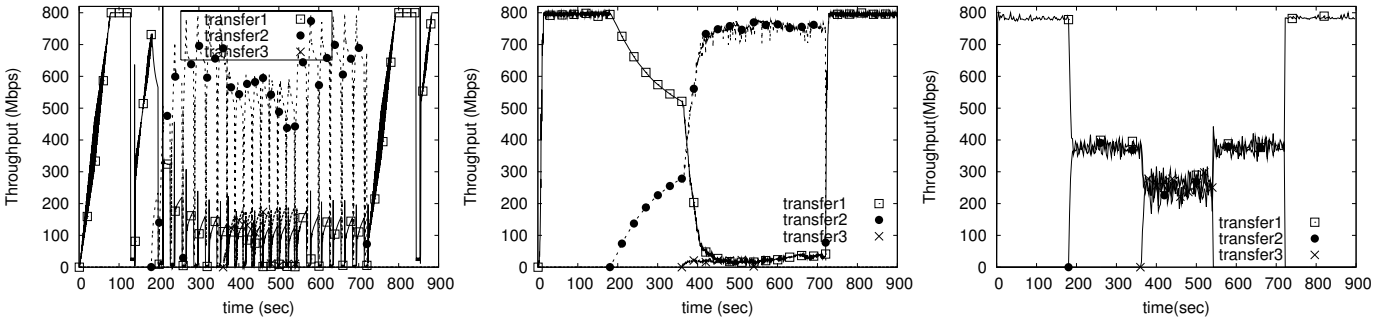
(b) Bottleneck Queue vs. Time

Figure 3: Performance of Slow Start on a 1 Gbps Network



(a) HighSpeed (b) CUBIC (c) FAST (d) RAPID

Figure 4: Performance in a Dynamic Bandwidth 1 Gbps Network



(a) HighSpeed (b) FAST (c) RAPID

Figure 5: Intra-protocol Fairness

the throughput of the transfer drops significantly and each protocol then takes tens-to-hundreds of RTTs for regaining[11] throughput.

HighSpeed and CUBIC are also slow in acquiring spare bandwidth when the avail-bw increase suddenly (for example, at 150s, 250s, and 350s)—this is simply due to the slow AB-search speed of these protocols. Note that if these protocols already have a large number of packets in the router buffers, they may be able to utilize spare bandwidth immediately—loss-based protocols are known to trade-off low latency for high throughput performance.

2. Delay-based FAST maintains smaller router queues in steady-state. However, when the avail-bw decreases suddenly, FAST is unable to react quickly and causes buffer overflow. After the first time this happens, unfortunately, the FAST transfer keeps overflowing the router buffers and is unable to converge to a stable sending rate.[14]

3. The RAPID transfer maintains very low queuing and is able to avoid packet losses, even when the avail-bw decreases suddenly. More importantly, though, the protocol is able to very quickly acquire additional spare bandwidth that becomes available—and it does so without maintaining large queues in routers!

4.2 Intra-protocol Fairness

We next evaluate fairness when multiple RAPID transfers share the bottleneck link. Our objective is to specifically evaluate fairness in heterogeneous RTT environments.

4.2.1 Dynamics Among Co-existing Transfers

Our first set of experiments is inspired by a similar experiment presented in [5], which evaluates the ability of a new transfer to acquire fair share of bandwidth from a pre-existing transfer—in that paper, FAST was shown to be better than other protocols in achieving fairness (when compared against HighSpeed as well as BIC). A bottleneck capacity of 800Mbps was used in that experiment—we use the same by setting the transmission capacity of R_1–R_2 in Fig 2 to 800Mbps.

For the experiment, we simulate three transfers—the first lasts from 0-900s and has an RTT of 200ms, the second lasts from 180-720s and has an RTT of 100 ms, and the third lasts from 360-540s and has an RTT of 150ms. Fig 5 plots the throughput obtained by the three transfers with different underlying protocols.

We find that both HighSpeed and CUBIC (latter not plotted here to improve visualization of Fig 5) can be quite unfair in the throughput allocated among the three transfers—the second transfer (which has the smallest RTT) dominates over the other two transfers. This has also been observed in [5]. Surprisingly, FAST also exhibits the same unfair behavior—the connection with the smallest RTT obtains a significantly high fraction of the avail-bw. This observation differs from the results presented in [5] that illustrate better fairness properties—unfortunately, [5] does not clearly specify the parameter settings used for the experiment. At the very least, our experiment illustrates that the performance of FAST is fairly sensitive to its parameter settings.

Fig 5(c) shows that RAPID allocates similar throughput to all transfers, irrespective of their RTTs—RAPID thus does not suffer from RTT-unfairness. Furthermore, the second transfer is able to acquire a fair share even though the pre-existing first transfer had a high throughput of 1 Gbps; this illustrates that the filter mechanism added in Section 3.2 successfully eliminates from RAPID any bias due to rate-proportional feedback frequency.

Fig 5(c) also illustrates that the convergence of the transfers to a fair share occurs at a fairly small timescale—we study the fairness timescale in more detail below.

4.2.2 Fairness Among Large Number of Transfers

We next evaluate how well the RAPID intra-protocol fairness scales when a larger number of connections are aggregated. For this, we conduct experiments in which a 1 Gbps bottleneck link is shared by n long-lived RAPID transfers for 600 seconds—we vary n from 2 to 100 across experiments. The RTTs of the n connections are selected uniformly randomly from two ranges: 60-80 ms and 135-165 ms. The start-times of the connections are selected uniformly randomly between 0-20s.

Starting at 20 seconds, the time-series of throughput obtained by each transfer are logged at several different timescales ranging from 500ms–128s. For each timescale, and for each logging instance of the associated time-series, we compute the Jain Fairness Index [17] of the throughput achieved by the n transfers. Fig 6 plots the median, 10th, and 90th percentiles (latter

[14] Our experiments in Appendix A show that providing the bottleneck router with thrice the number of buffers does help FAST in completely avoiding losses and utilizing avail-bw—however, such large buffers are also likely to improve the performance of HighSpeed and CUBIC.

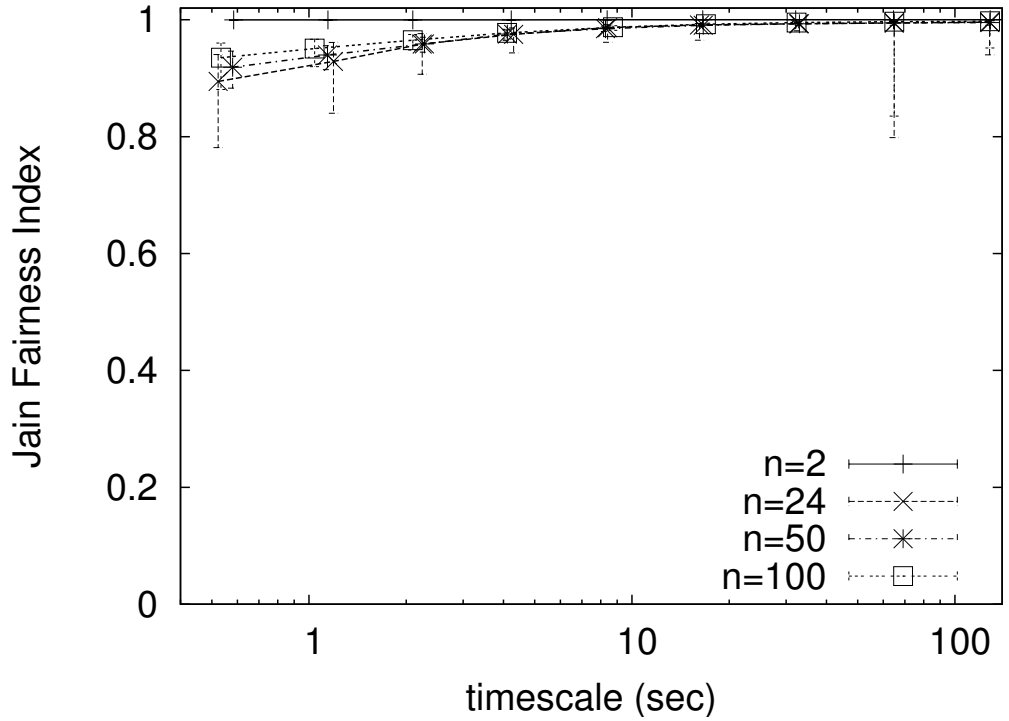


Figure 6: Intra-protocol Fairness (Large Number of Transfers)

plotted as error bars) of the distribution of the fairness index, as a function of the timescale at which the throughput data is observed (for $n = 2, 24, 50, 100$).

We find that RAPID ensures excellent fairness (median fairness index > 0.8) among a large number of co-existing transfers with heterogeneous RTTs—this is true even at timescales as small as 500ms. Further, even the 10-percentile values of the indices are fairly high, indicating that it is quite rare for even transient "unfair" episodes—in which some connections obtain much less throughput than others—to occur.

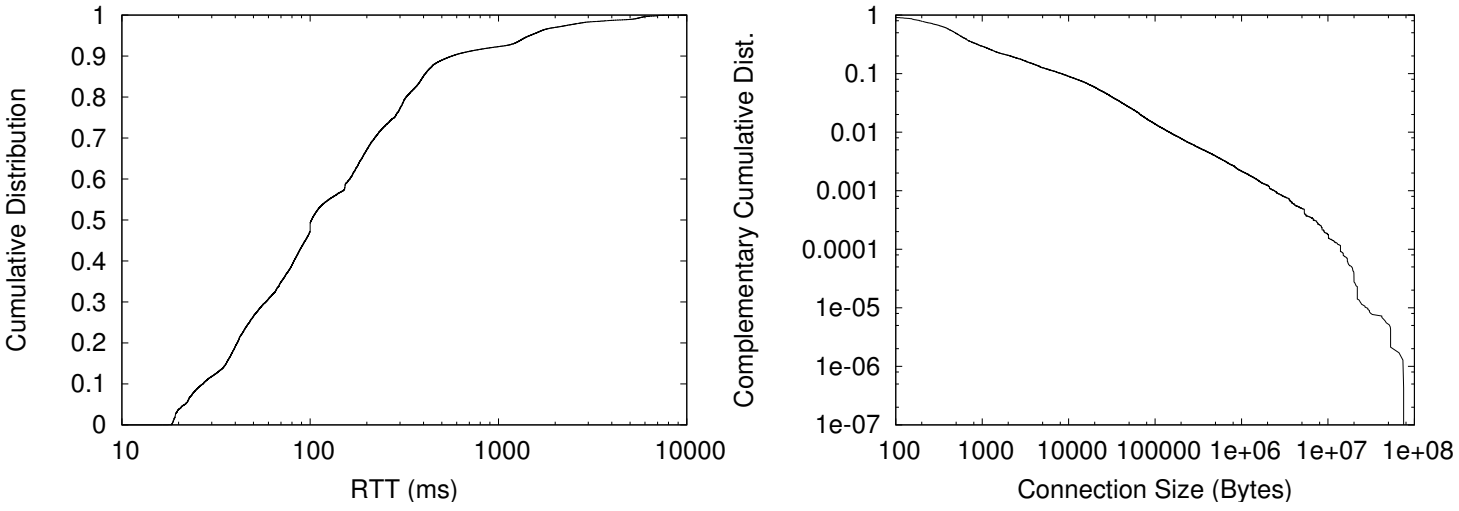
4.3 Co-existence with Low-speed TCP Traffic

Finally, we evaluate the impact of high-throughput RAPID transfers on a realistic mix of regular TCP transfers that co-exist on a bottleneck link. For this, we use the publicly-available Tmix traffic generator code for ns-2, which generates an empirically-derived TCP traffic mix that is representative of TCP traffic aggregates observed on production Internet links [18]. TMIX simulates application-level socket-writing behavior and runs over the TCP protocol—it, consequently, generates response TCP traffic. Our objectives are to study: (i) how a high-throughput transfer might impact the performance of connections in such a traffic mix, and (ii) how effectively can a high-throughput transfer utilize the avail-bw with such dynamic (and realistic) cross-traffic.

We use TMIX to generate TCP traffic at an average offered load of 70Mbps for 30 minutes and drive it through a bottleneck link (R_1–R_2 in Fig 2) of 100 Mbps—the bottleneck buffers are set to 750 packets (based on the RTT of the $\mathcal{L}$ transfer described below). For the set of TCP connections simulated in our TMIX experiment, Fig 7(a) plots the distributions of per-connection RTTs—we find that these are fairly diverse (ranging from 10ms LANs to long-distance transfers). Fig 7(b) plots the (complementary) distribution of the number of bytes transferred in each connection—the traffic mix is typical of that found on the Internet (a majority of small *mice* connections, but a few heavy *elephants*). All statistics presented below are collected from roughly the middle 20 minutes of the experiments (to exclude the ramp-up and ramp-down behavior of the traffic generator).

We first simulate only the TMIX traffic and observe the aggregate throughput and queue sizes at the bottleneck router at a timescale of 1s—these are plotted respectively in Fig 8(a) and 8(b). We find that although the average offered load of the TMIX aggregate is 70Mbps, the traffic is fairly bursty; the short-term load can vary from 50-100Mbps. This causes the router queues to vary rapidly from 0$\sim$80 packets (and occasionally to larger amounts). Thus, the TMIX aggregate represents *bursty cross-traffic* that causes the avail-bw on the bottleneck router link to vary dynamically.

We then re-run the Tmix experiment and add a single bulk transfer (henceforth, referred to as $\mathcal{L}$) that shares the bottleneck



(a) Distribution of Transfer RTTs

(b) Distribution of Transfer Sizes

Figure 7: Statistics of TMIX Transfers

link and has an RTT of 60ms—we repeat this experiment three times with $\mathcal{L}$ running over HighSpeed, FAST, and RAPID, respectively. In each of these experiments, we also log the throughput observed by $\mathcal{L}$, the queue sizes at the bottleneck router, as well as the aggregate throughput of the bottleneck router. Figs 8(c), 8(e), and 8(g) plot the throughput of $\mathcal{L}$ observed with HighSpeed, FAST, and RAPID, respectively. Figs 8(d), 8(f), and 8(h) plot the router queue sizes, respectively. We find that:

1. Fig 8(d) shows that the loss-based HighSpeed transfer fills up (and overflows) the router queues at a frequent rate—consequently, the responsive TCP connections in the TMIX cross-traffic suffer packet losses and reduce their sending rates. The HighSpeed $\mathcal{L}$ transfer is, thus, able to obtain a throughput much higher than the spare bandwidth available in the TMIX-only experiment.

2. The $\mathcal{L}$ transfer using FAST TCP is barely able to obtain any throughput at all (less than 1Mbps throughout the experiment). This suggests that FAST is unable to effectively utilize the dynamically-varying avail-bw on the bottleneck link—indeed, the utilization of the bottleneck link in this experiment is similar to that observed in the TMIX-only experiment. Fig 8(f) shows that the presence of the FAST-based $\mathcal{L}$ has a negligible impact on the router queues.

3. The RAPID-based $\mathcal{L}$ transfer is able to utilize the spare bandwidth fairly effectively—the fast AB-search speed of RAPID is crucial in achieving this behavior. Fig 8(h) shows that RAPID does do while increasing the router queue buildup by only a small amount. The bottleneck link is nearly-100% utilized throughout this experiment.

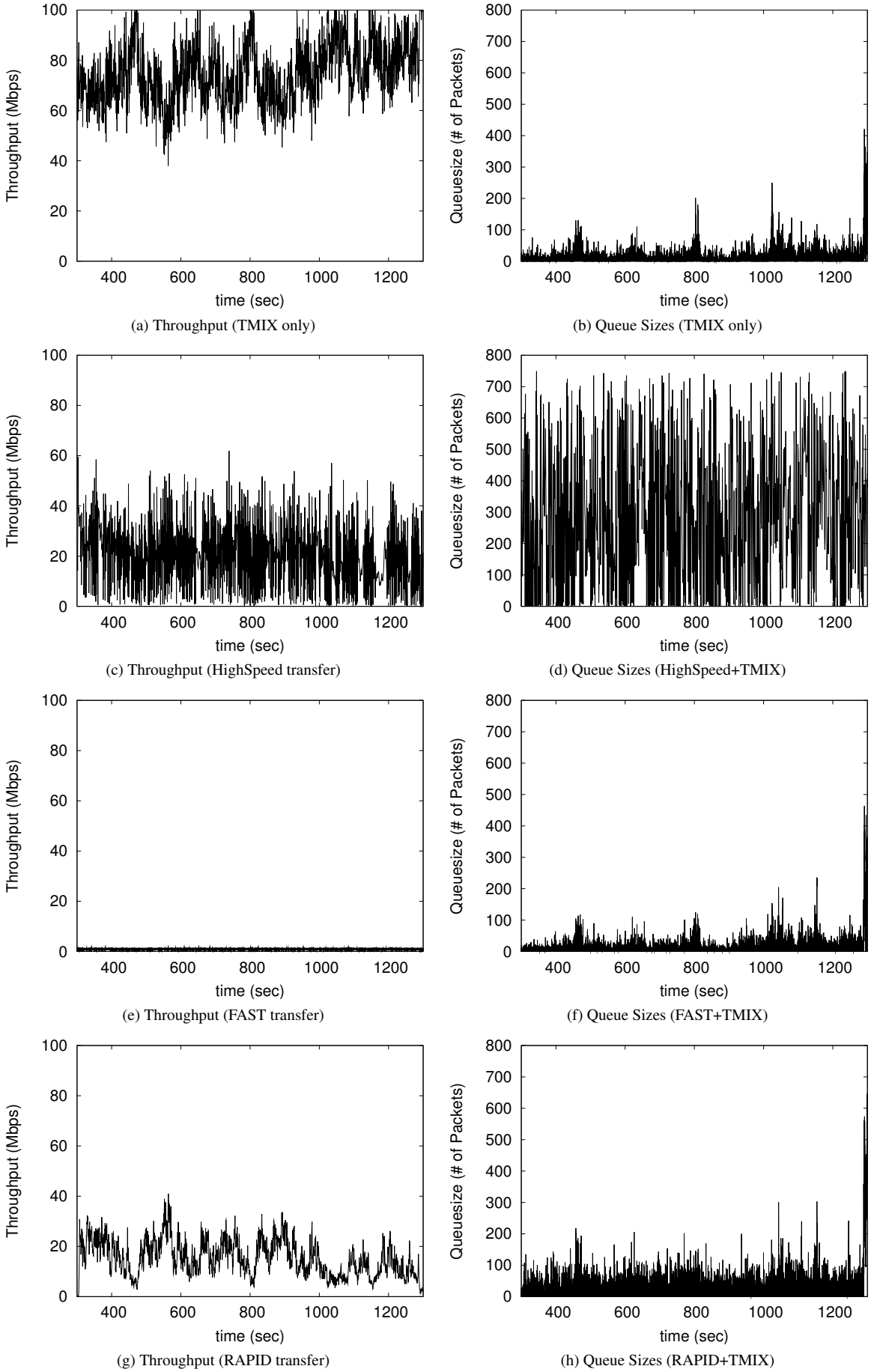
5 Concluding Remarks

In this paper, we propose a novel protocol, RAPID, that reduces the timescales at which congestion-control operates by orders of magnitude—this enables the protocol to efficiently utilize spare bandwidth in high-speed and dynamic bandwidth environments. RAPID does so while: (i) providing excellent intra-protocol fairness in heterogeneous RTT environments, and (ii) ensuring friendly co-existence with regular TCP transfers and router queues.

Perhaps one of the key challenges to deploying RAPID in multi-gigabit networks is related to the high-precision packet time-stamping and packet-spacing (of the order of a few microseconds) that it would need to rely on. Fortunately, since RAPID only relies on detecting *increasing trends* in the inter-packet spacing, the need for accuracy is more relaxed. Consequently, we believe that existing high-end PC platforms should be able to support RAPID for up to 1 Gbps speeds. We are also exploring the use of FPGA-based network cards to enable a RAPID implementation to scale up to multi-gigabit capabilities. In addition, we are currently designing mechanisms to help auto-tune RAPID as it scales up to multi-gigabit and multi-hop networks.

References

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Apr. 1999, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt



(a) Throughput (TMIX only)

(b) Queue Sizes (TMIX only)

(c) Throughput (HighSpeed transfer)

(d) Queue Sizes (HighSpeed+TMIX)

(e) Throughput (FAST transfer)

(f) Queue Sizes (FAST+TMIX)

(g) Throughput (RAPID transfer)

(h) Queue Sizes (RAPID+TMIX)

Figure 8: Statistics of TMIX Transfers

[2] S. Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649 (Experimental), Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[3] I. Rhee, L. Xu, and S. Ha, "CUBIC for fast long-distance networks," August 2007, Internet Draft.

[4] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast, long distance networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[5] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[6] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zoharjan, "PCP: Efficient endpoint congestion control," in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.

[7] M. Fukuhara, F. Hirose, T. Hatano, H. Shigeno, and K. Okada, "SRF TCP: A TCP-friendly and fair congestion control method for high-speed networks," in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, ser. Lecture Notes in Computer Science, vol. 3544. Springer, 2005, pp. 169–183.

[8] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," in *First International Workshop on Protocols for Fast Long-distance Networks*, February 2003.

[9] S. Floyd and M. Allman, "Specifying New Congestion Control Algorithms," RFC 5033, 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5033.txt

[10] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Passive and Active Measurement Workshop* , April 2003.

[11] A. Shriram and J. Kaur, "Empirical evaluations of techniques for measuring available bandwidth," in *Proceedings of IEEE INFOCOM 2007*, May 2007.

[12] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," RFC 3742 (Experimental), Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3742.txt

[13] S. Gorinsky and H. Vin, "Additive increase appears inferior," Dept. of Computer Sciences, University of Texas at Austin, Tech. Rep., 2000.

[14] M. Vojnovic, J. L. Boudec, and C. Boutremans, "Global fairness of additive-increase and multiplicative- decrease with heterogeneous round-trip times," in *Proceedings of IEEE INFOCOM*, March 2000.

[15] "Network simulator-2 ns2 (http://www.isi.edu/nsnam/ns/)."

[16] "URL http://www.cubinlab.ee.unimelb.edu.au/ns2fasttcp/."

[17] R. Jain, D. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource alloca tion in shared computer system," September 1984, dEC Research Report TR-301.

[18] M. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. Smith, "Tmix: A tool for generating realistic application workloads in ns-2," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 67–76, 2006.

A Experiments with Fast with different parameters

In this section, We perform 3 sets of experiments that show the effect of parameters α, β and minimum threshold (mi) on FAST [5].

The set of values used (shown in table 5) are according to the test scripts that came with the FAST source code for ns.

α	β	mi
10	10	0.0001
100	100	0.00075
100	100	0.0006
200	200	0.00075
800	800	0.00075
10000	10000	0.00015

Table 5: Values of α, β and mi

A.1 Slowstart

First we examine the behaviour of FAST during slow start for different values of α, β and minimum threshold (mi).Fig 9 plots as a function of time, the throughput obtained by a single long lived transfer on a 1GB link with FAST for different values of parameters. We can observe that

- for some parameter settings (figs 9(b) and 9(c)) slow start causes losses.
- the ramp up time depends on the values of the parameters. It is best when α = 10000, β = 10000 and mi = 0.00015 (figs 9(f)).

A.2 Dynamic Bandwidth

Next, we simulate a network for 500 seconds, and introduce 4 constant-bit-rate (cbr) traffic streams on the bottleneck link according to the following schedule: *cbr-1* exists from 50-400 seconds, *cbr-2* exists from 100-150 seconds, *cbr-3* exists from 250-350 seconds, and *cbr-4* exists for a small duration from 460-462 seconds. Each cbr stream has a bit-rate of 200 Mbps. Fig 10 plots as a function of time, the throughput obtained by the fast transfer.

- Figs 9(b), 9(c) and 9(d) show losses.
- The performance is best when α = 800, β = 800 and mi = 0.00075 (fig 9(e)).

A.3 Fairness Experiment

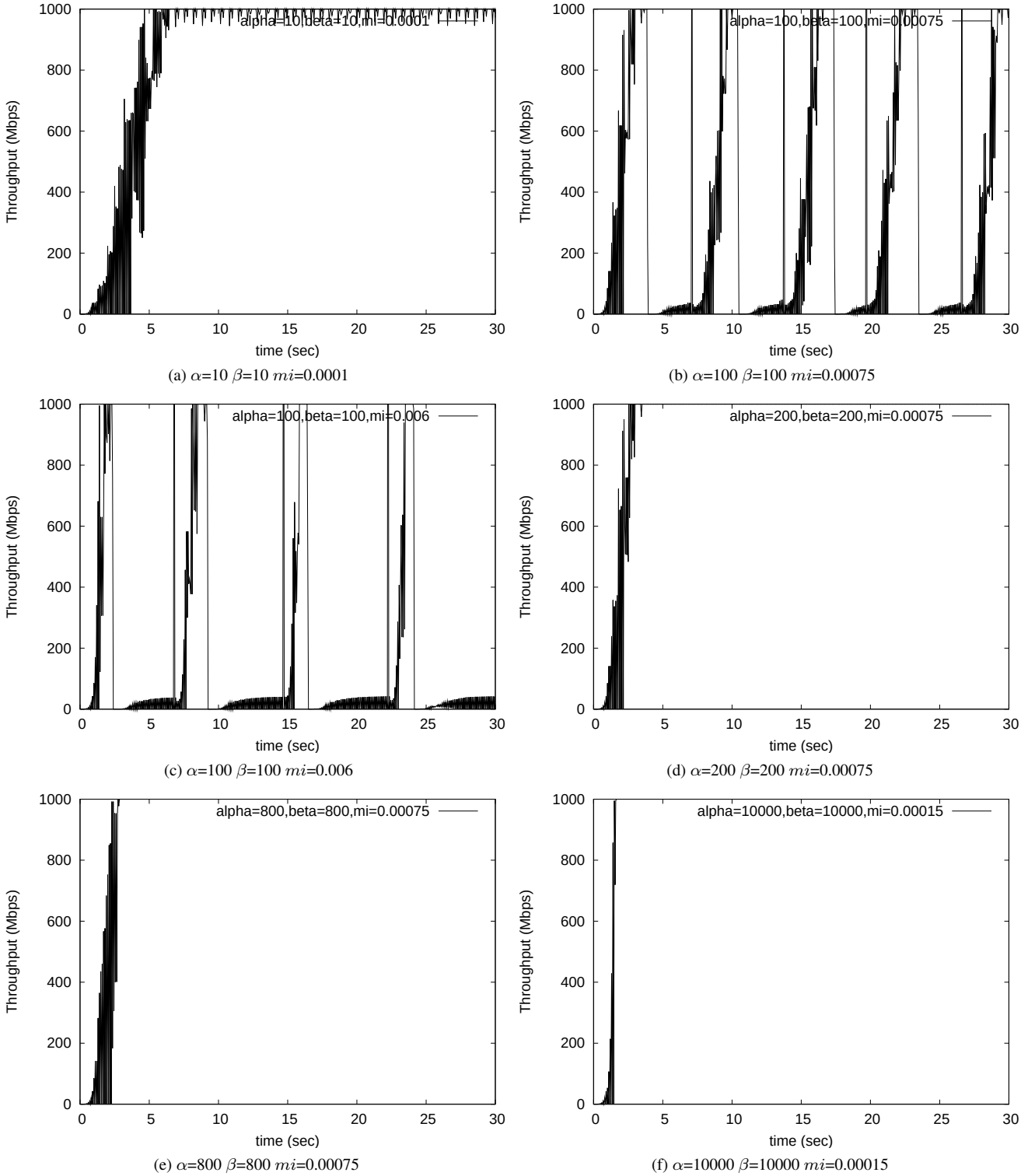
Lastly, we simulate three transfers—the first lasts from 0-900s and has an RTT of 200ms, the second lasts from 180-720s and has an RTT of 100 ms, and the third lasts from 360-540s and has an RTT of 150ms. Fig 11 plots the throughputs obtained by the three transfers with FAST for different values of the parameters α, β and minimum threshold(mi).

- From fig 11(a), where α = 10, β = 10 and mi = 0.0001, the third transfer do not get any throughput. The first transfer dies down when the second transfer starts.
- Fast gives best fairness when α=100, β=100 and mi=0.00075
- Fig 11(c) and Fig 11(f) show high unfairness towards the third transfer.
- Fig 11(d), where α = 200, β = 200 and mi = 0.00075, and fig 11(e), where α = 800, β = 800 and mi = 0.00075, show slight bias towards the second transfer initially. But, they converge towards their fair share when the third transfer starts.

This shows that the fairness of FAST is highly dependent on values of α, β and mi.
In general, we can see that the performance of FAST depends on the values of the parameters. Some key observations are

- For some parameter settings, even slow start causes losses.
- The parameter settings for best fairness (fig 11(b)) gives worst peformance with CBR cross traffic fig 10(b).
- The parameter settings which perform best with CBR cross traffic (fig 10(e)) does not do well with respect to fairness (fig 11(e)).

A.4 SlowStart simulations with Fast



(a) α=10 β=10 mi=0.0001

(b) α=100 β=100 mi=0.00075

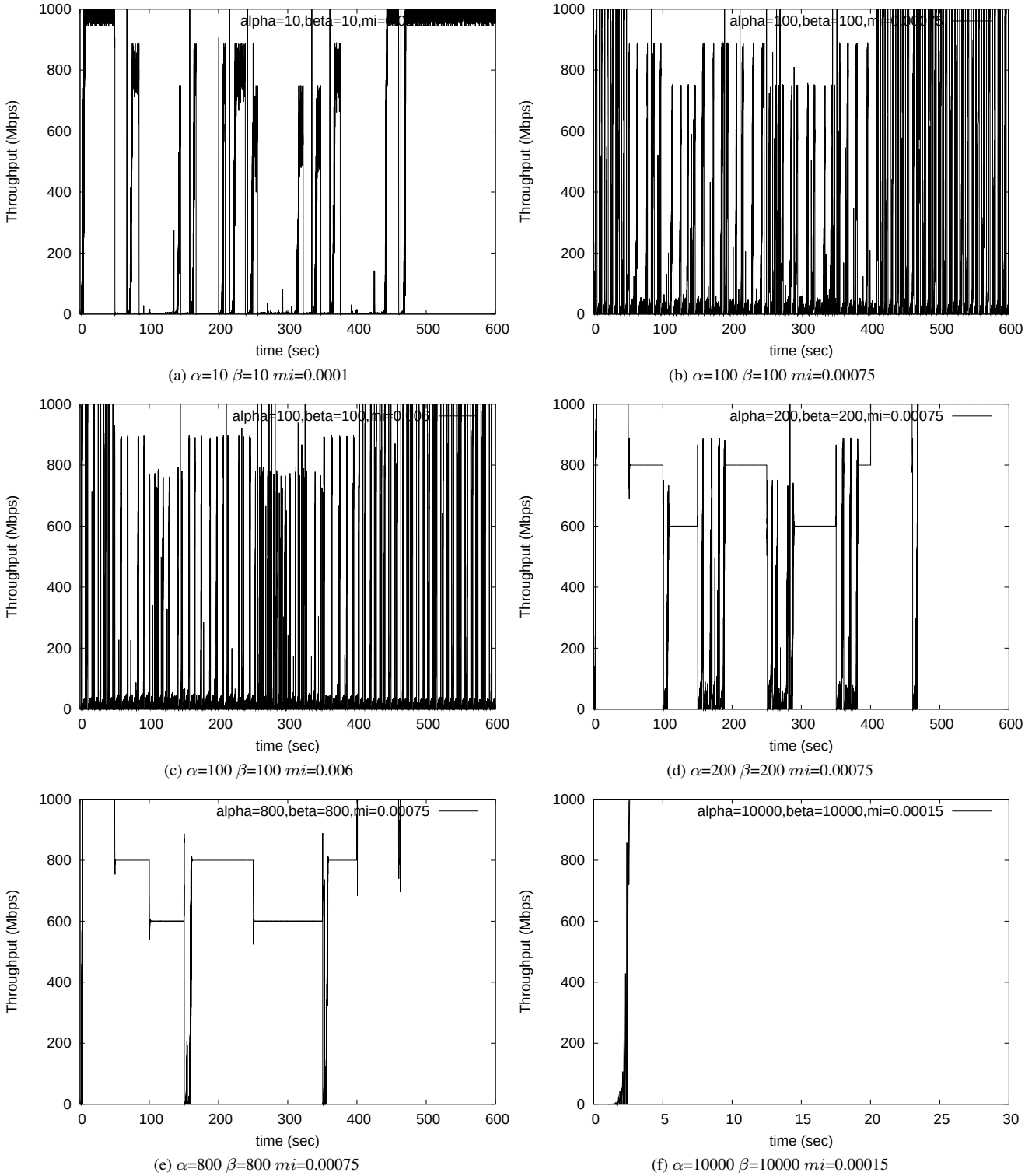
(c) α=100 β=100 mi=0.006

(d) α=200 β=200 mi=0.00075

(e) α=800 β=800 mi=0.00075

(f) α=10000 β=10000 mi=0.00015

Figure 9: Effect of α, β and mi on slow-start



(a) α=10 β=10 mi=0.0001

(b) α=100 β=100 mi=0.00075

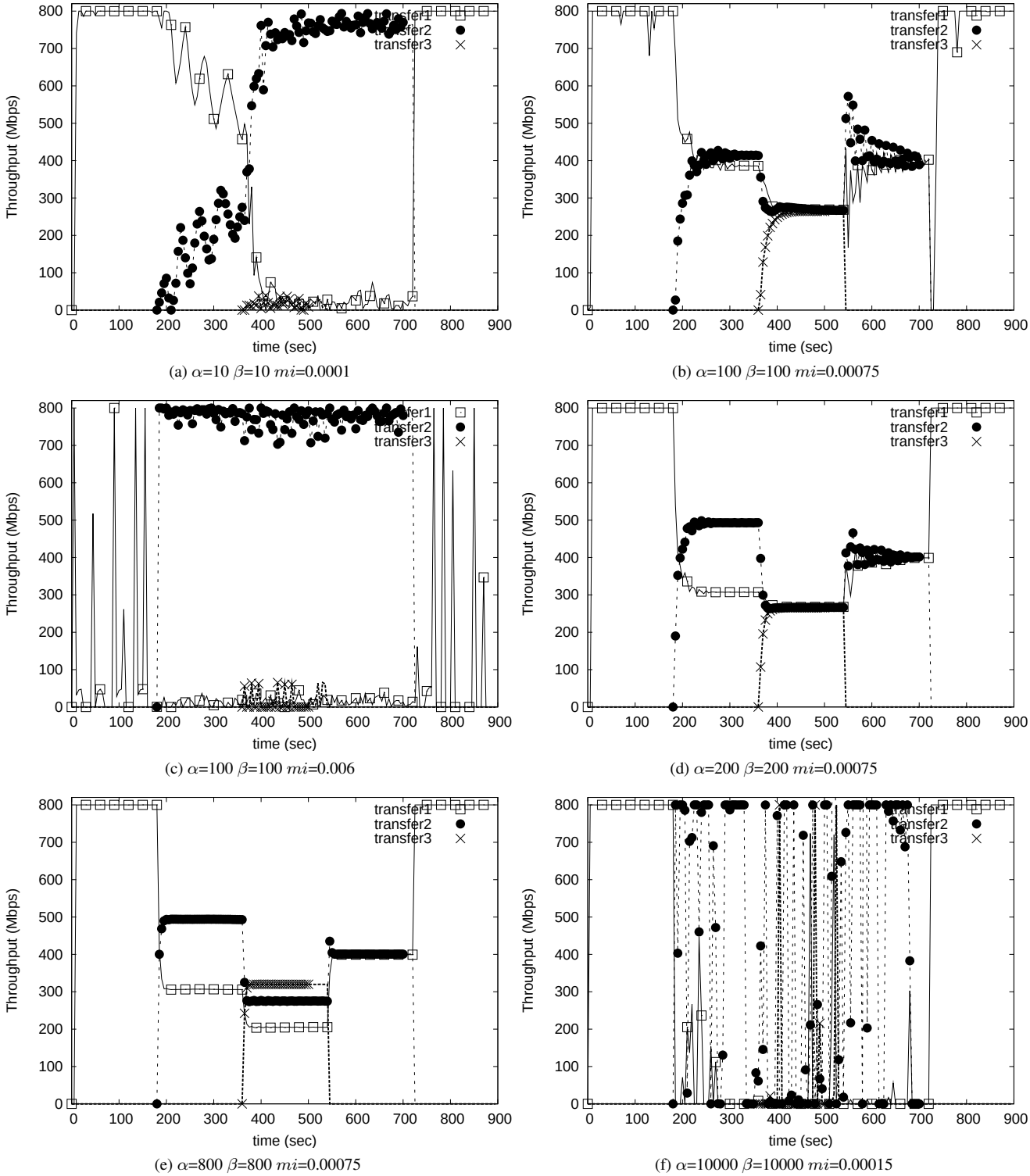
(c) α=100 β=100 mi=0.006

(d) α=200 β=200 mi=0.00075

(e) α=800 β=800 mi=0.00075

(f) α=10000 β=10000 mi=0.00015

Figure 10: Effect of α, β and mi with CBR cross traffic



(a) α=10 β=10 mi=0.0001

(b) α=100 β=100 mi=0.00075

(c) α=100 β=100 mi=0.006

(d) α=200 β=200 mi=0.00075

(e) α=800 β=800 mi=0.00075

(f) α=10000 β=10000 mi=0.00015

Figure 11: Effect of α, β and mi

B Sensitivity Analysis

This section attempts to study the sensitivity of RAPID to parameters m, N and the ratio r ($\frac{r_1}{r_a vg}$). We study the effect of these parameters on queue build up, fairness and utilisation. For all these simulations, we rely on a simple dumbbell topology in which multiple sources are aggregated at a single bottleneck link (R_1–R_2 in Fig 2)—this bottleneck link is the only link shared by co-existing transfers. All links other than the bottleneck link have a transmission capacity of 10 Gbps—the bottleneck link capacity is set by default to 1 Gbps. We vary N from 15 to 100 and m from 1.01 to 1.15. We vary the number of connections n sharing the link from 2 to 100. The RTTs of the n connections are selected uniformly randomly from two ranges: 60-80 ms and 135-165 ms. The start-times of the connections are selected uniformly randomly between 0-20s.

Next we explain the impact of each of these factors on the three performance metrics.

For queue analysis, we plot the cdf of the queuesize. For fairness analysis, in each simulation, we plot starting at 20 seconds, the time-series of throughput obtained by each transfer are logged at several different timescales ranging from 500ms–128s. For each timescale, and for each logging instance of the associated time-series, we compute the Jain Fairness Index [17] of the throughput achieved by the n transfers. We then plot the median, 10th, and 90th percentiles (latter plotted as error bars) of the distribution of the fairness index, as a function of the timescale at which the throughput data is observed (for $n = 2, 4, 24, 50, 100$). Similar analyis is done for utilisation to calculate utilisation index. Three different set of plots are generated for each of these perfomance metrics.

- the first set from fig 12 to fig 29 where N is held constant
- the second set from fig 30 to fig 44 where m is held constant
- the third set from fig 45 to fig 62 where r is held constant

B.1 Queue build up

- For a given value of N, we change both m and r. We observe that decreasing r (or increasing m) increase queuesizes.
- For a given value of m, increasing N (or decreasing r) cause larger queues.
- For a given value of r, as N increases, queuesizes slightly increase. And, when r is too small, large value of N causes losses as the number of connections increase.

From the above observations, we conclude that with decreasing r the max rate of the chirp increases leading to larger queuesizes. Also, increasing N increases the number of rates per chirp leads to more bursty traffic (more rates above the average abestimate) thus increasing queuesizes.

B.2 Fairness

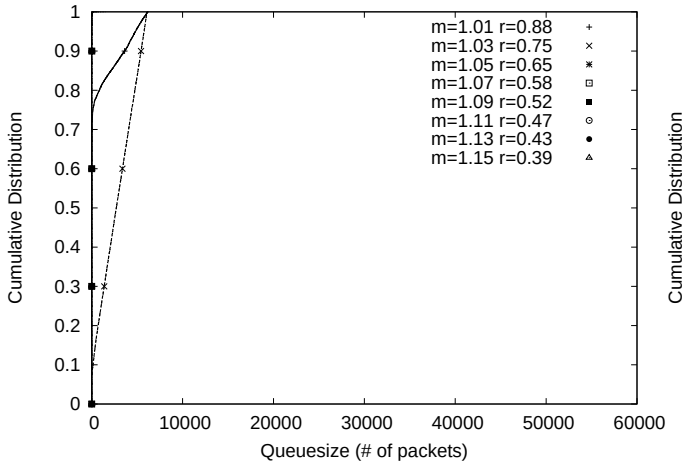
- For a given value of N, increasing m (or decreasing r) improves fairness.
- For a given value of m, increasing n (or decreasing r) improves fairness.
- For a given value of r, changing N(or m) does not have any impact.

From the above observations, we conclude that with decreasing r, the ranges of rates that different connections try, overlap thus giving chance to get the same estimate (fairshare).Thus, fairness improves with decreasing r.

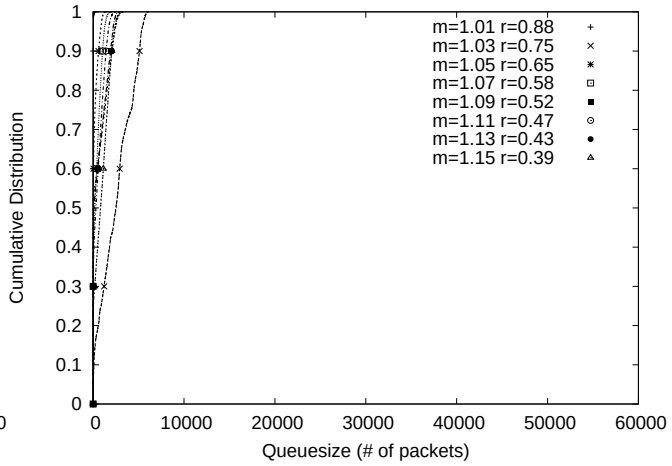
B.3 Utilisation

- For a given value of N, decreasing m gives better utilisation when the number of connections is less. But when number of connections increasing m does not have much impact.
- For a given value of m, decreasing r decreases the utilisation. Again, with increasing number of connections, r doesnot have much impact on utilisation.
- For a given value of r, changing N do not have any impact.

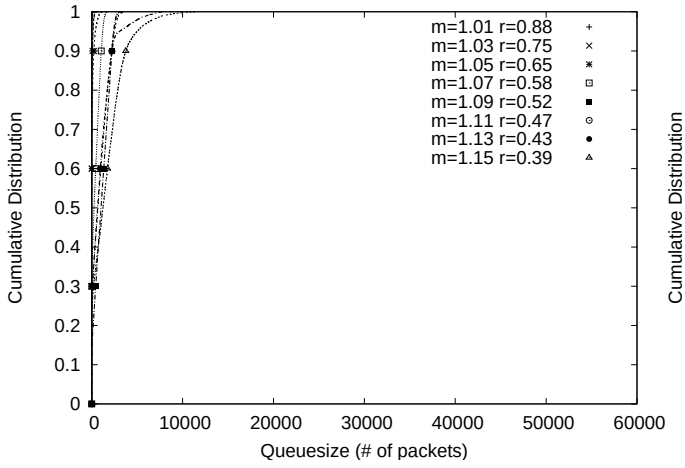
In general, we conclude that utilisation increases as the number of connections increase and lesser values of r give lesser utilisation.



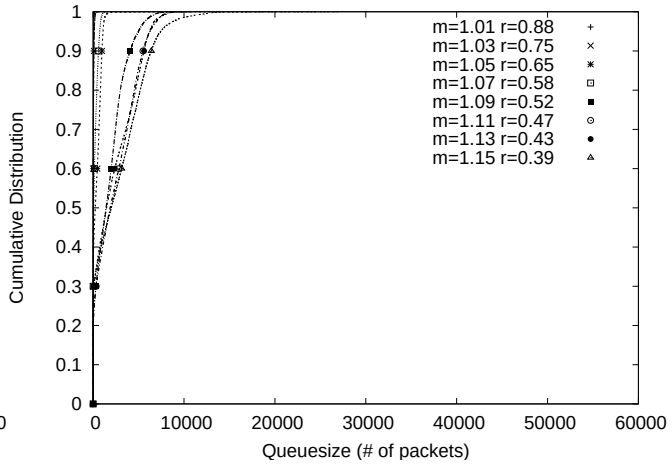
(a) n=2



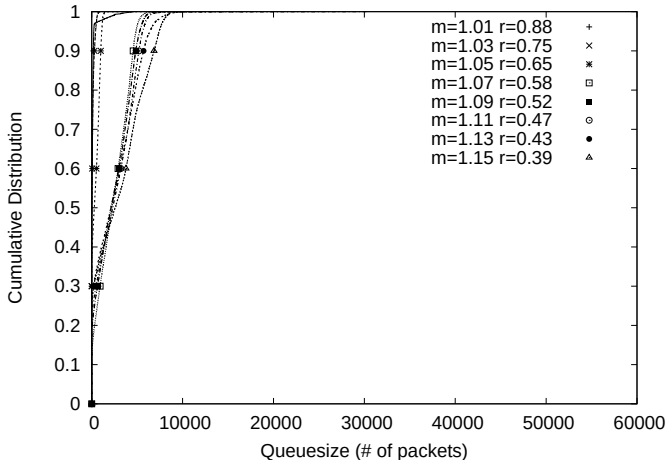
(b) n=4



(c) n=24

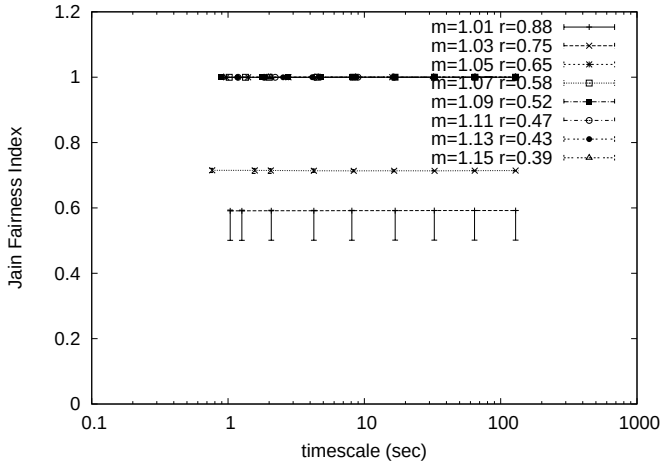


(d) n=50

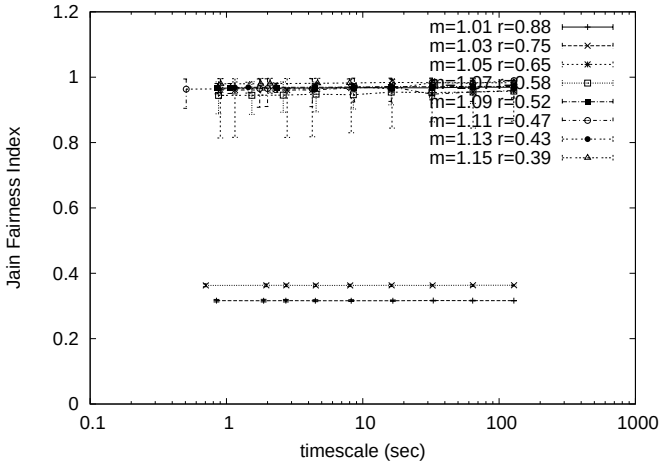


(e) n=100

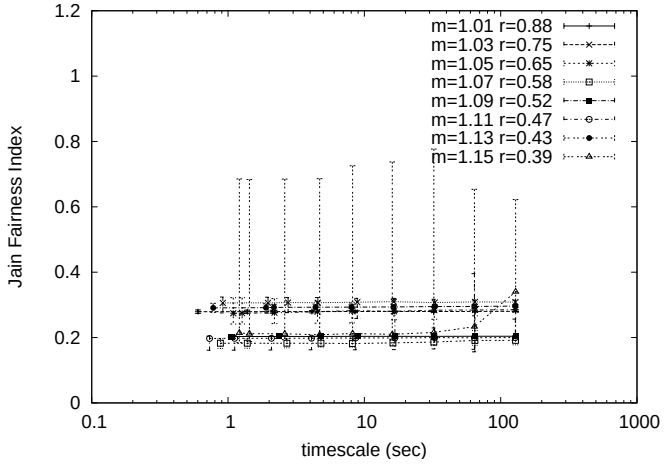
Figure 12: Effect of m on queuesizes with $N = 20$



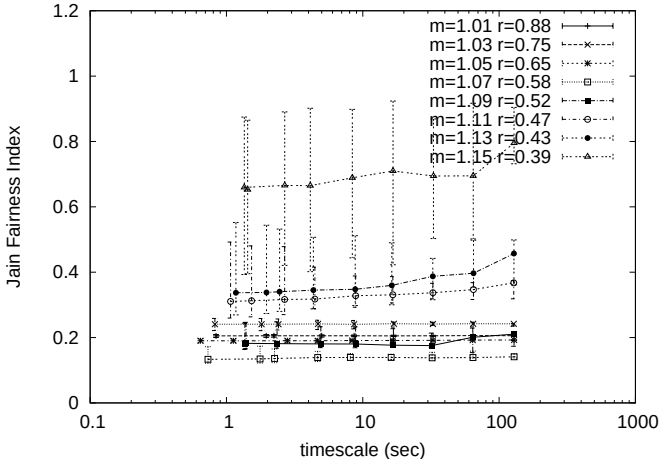
(a) n=2



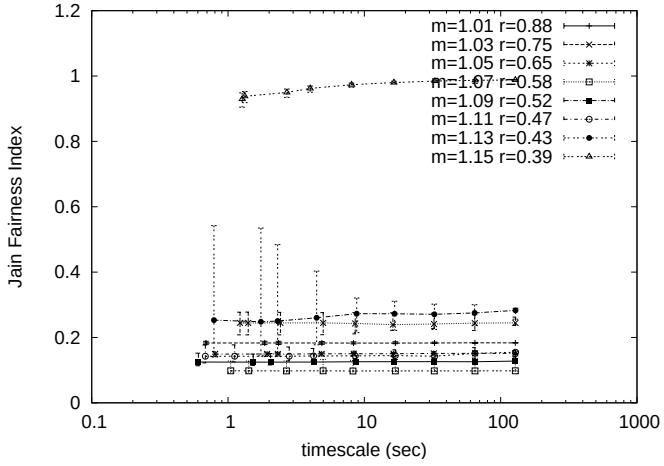
(b) n=4



(c) n=24

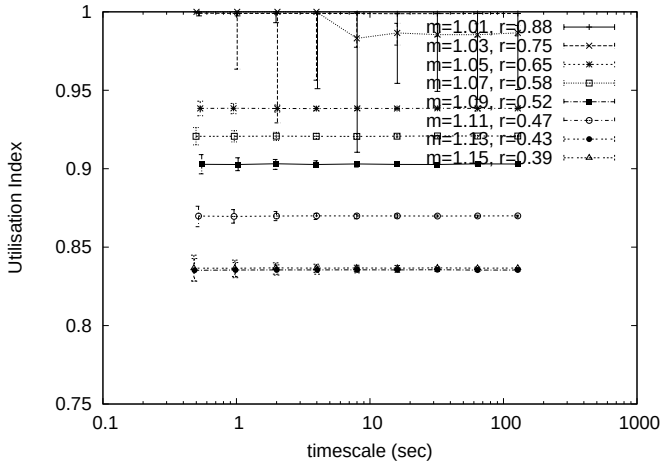


(d) n=50

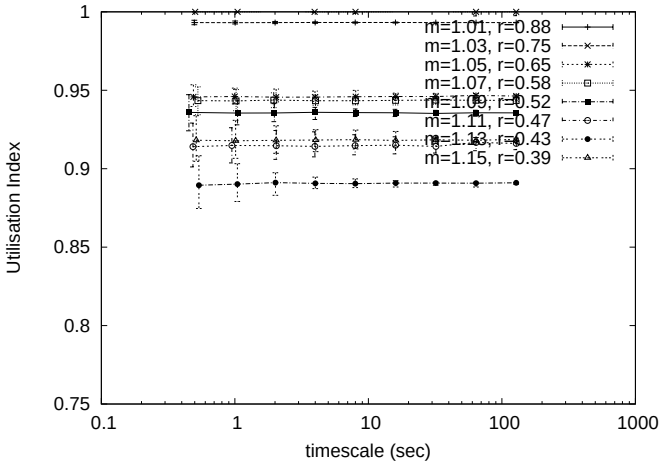


(e) n=100

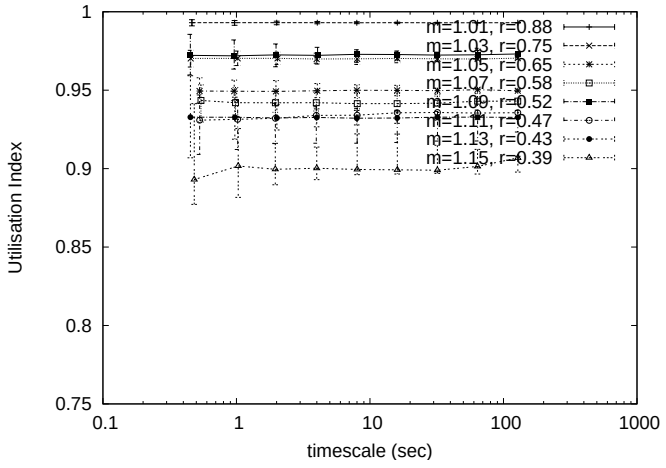
Figure 13: Effect of m on fairness with N = 20



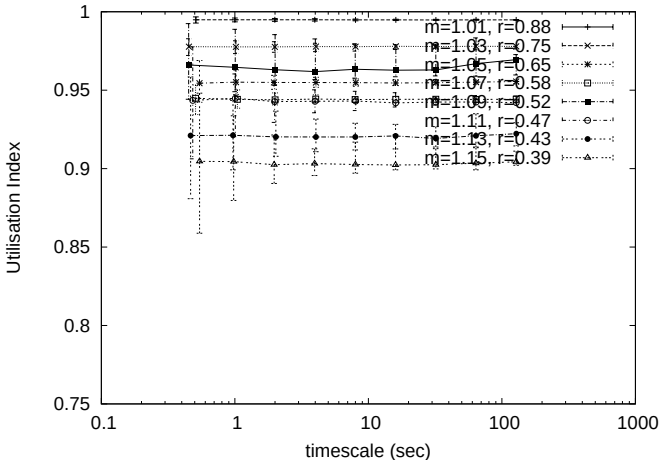
(a) n=2



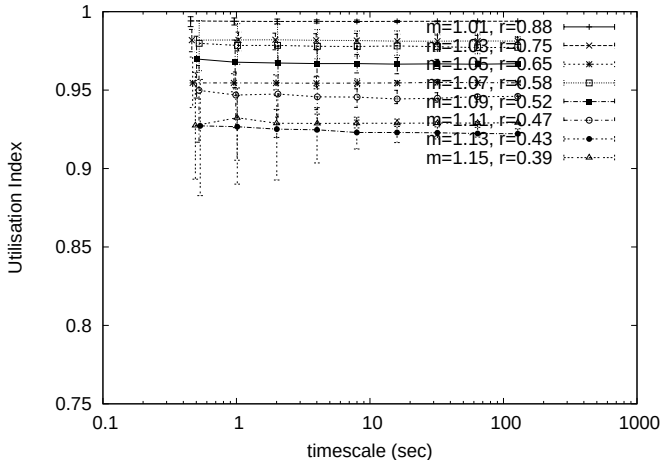
(b) n=4



(c) n=24

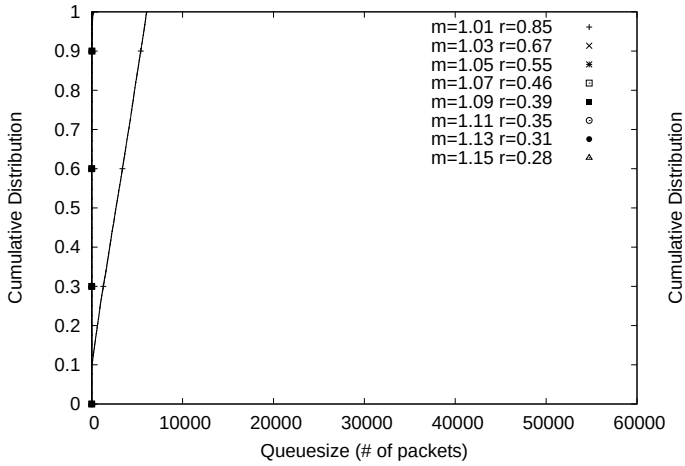


(d) n=50

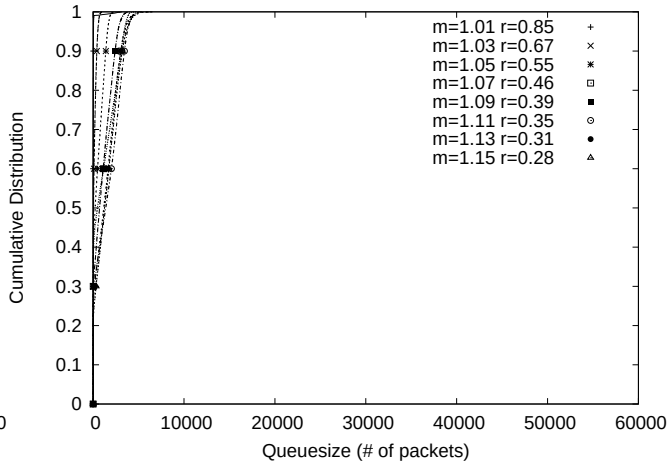


(e) n=100

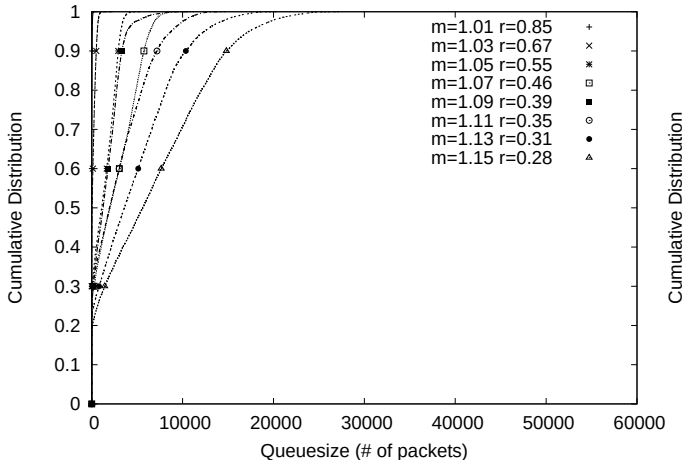
Figure 14: Effect of m on utilisation with $N = 20$



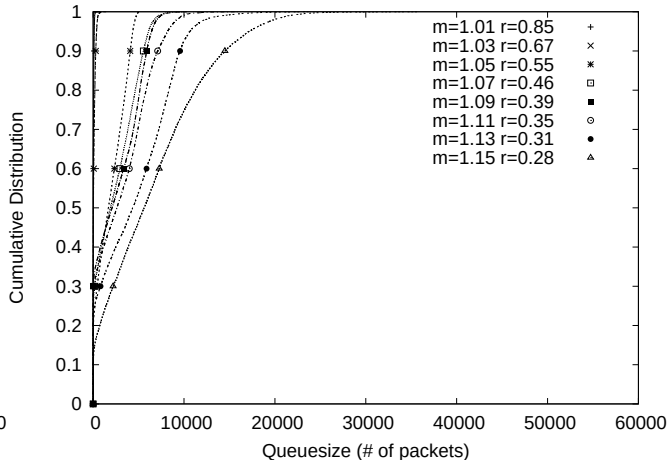
(a) n=2



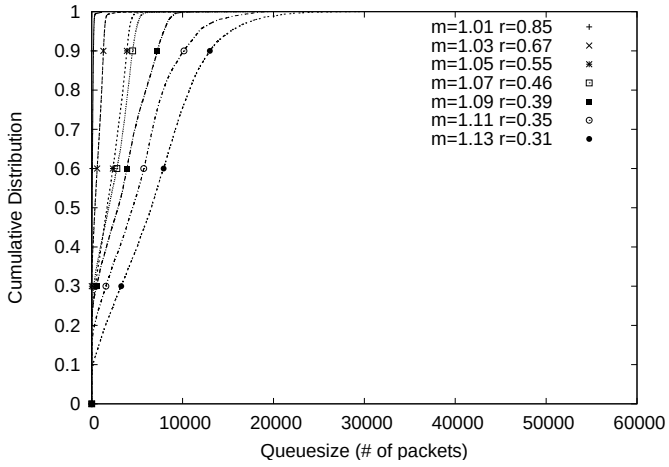
(b) n=4



(c) n=24

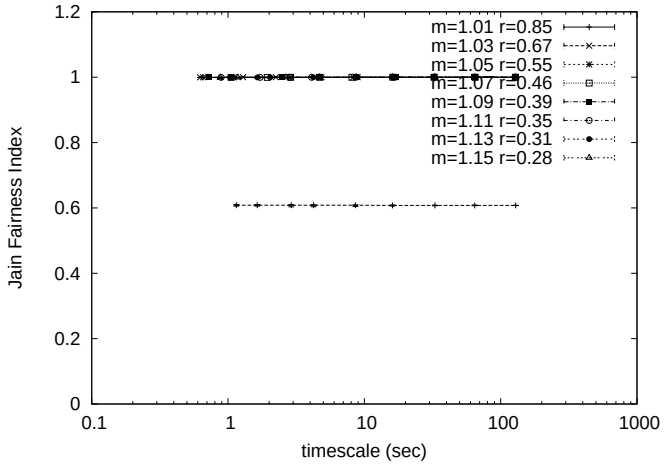


(d) n=50

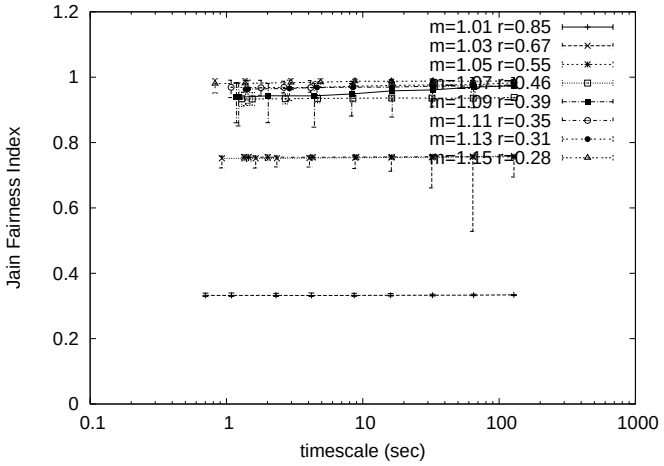


(e) n=100

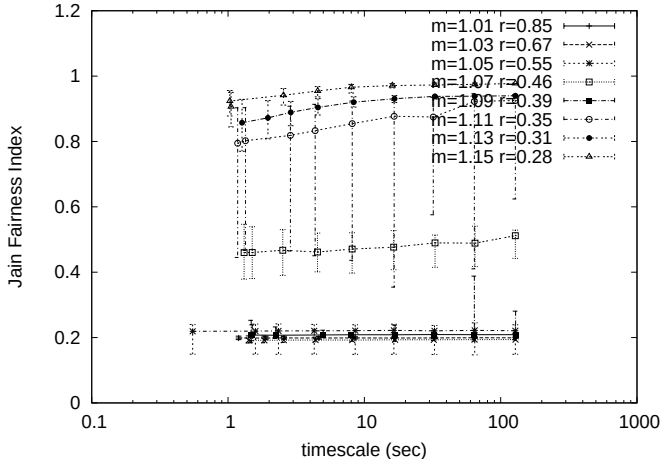
Figure 15: Effect of m on queuesizes with N = 30



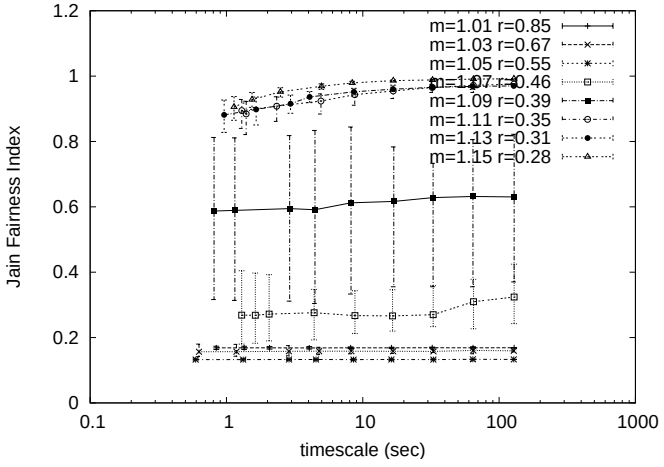
(a) n=2



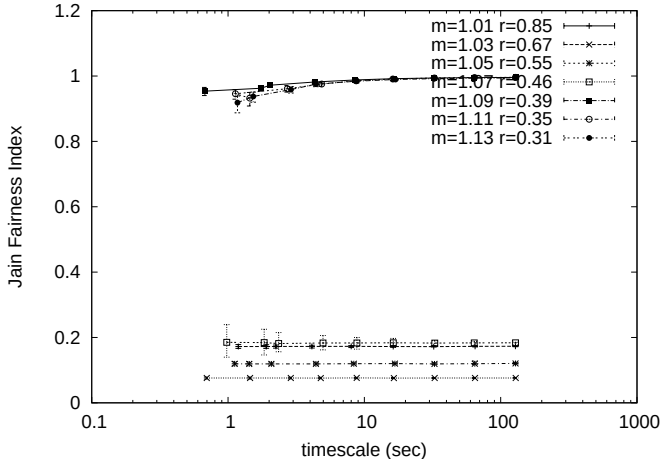
(b) n=4



(c) n=24

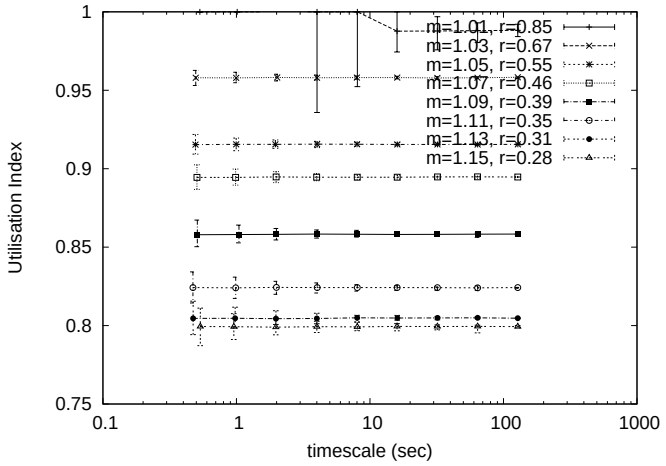


(d) n=50

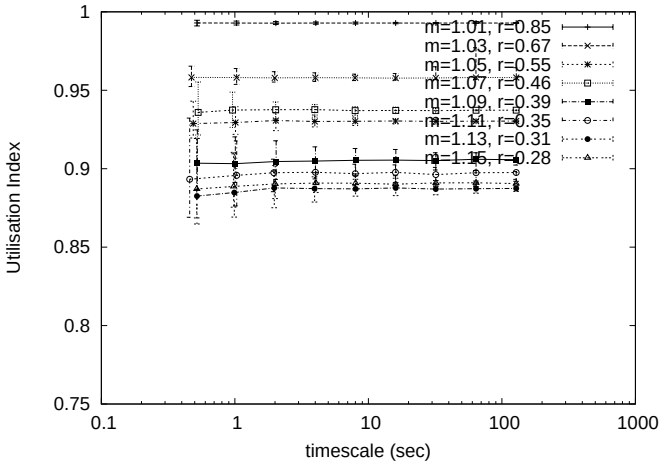


(e) n=100

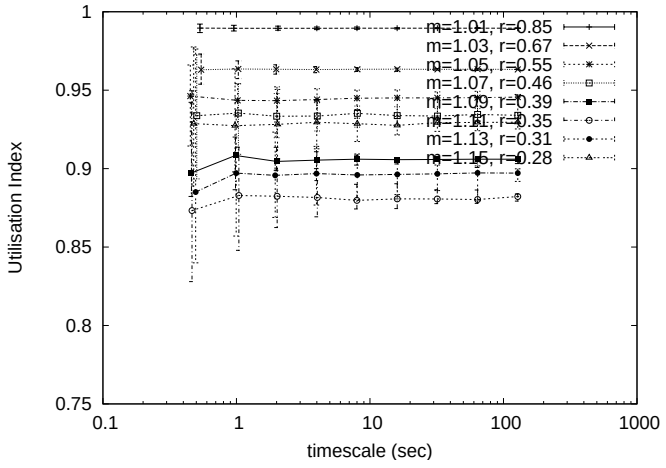
Figure 16: Effect of m on fairness with N = 30



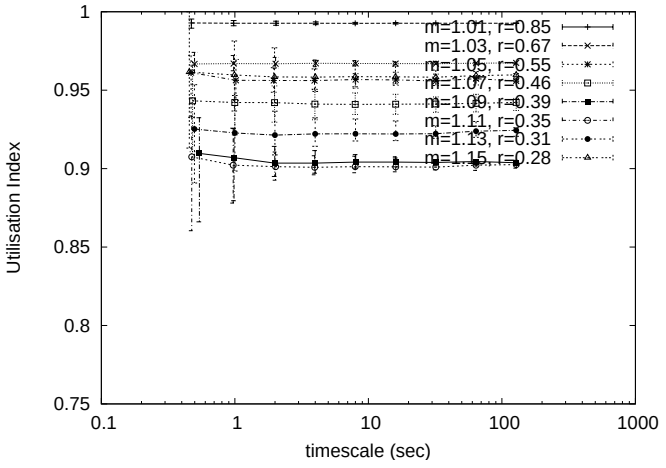
(a) n=2



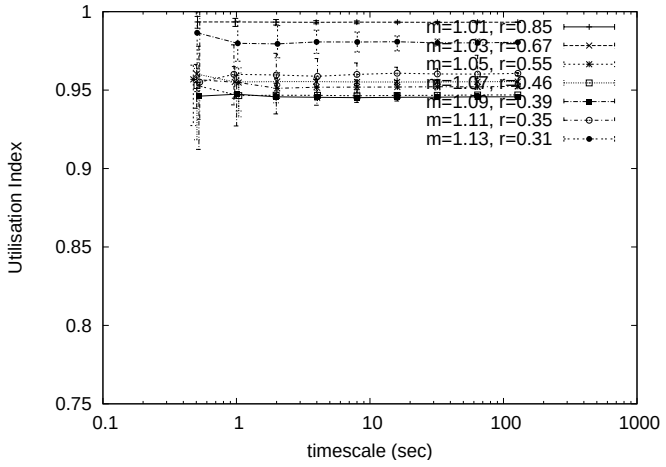
(b) n=4



(c) n=24

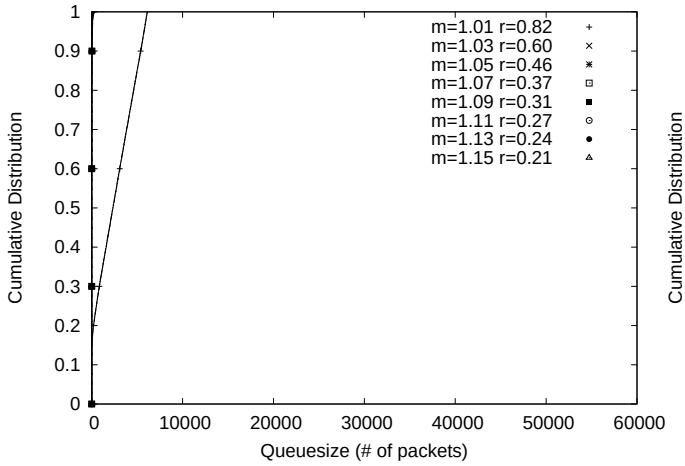


(d) n=50

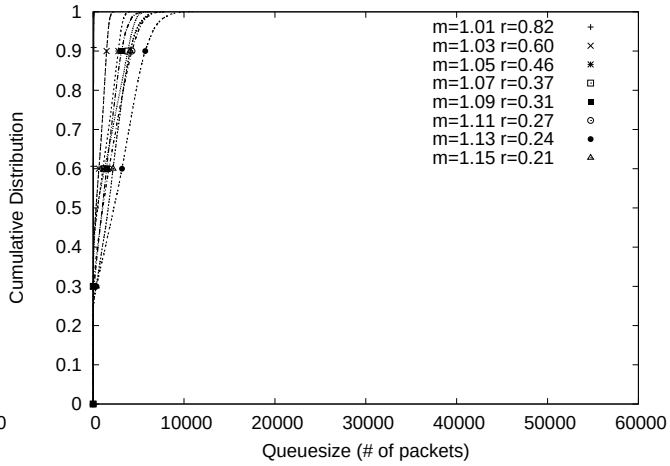


(e) n=100

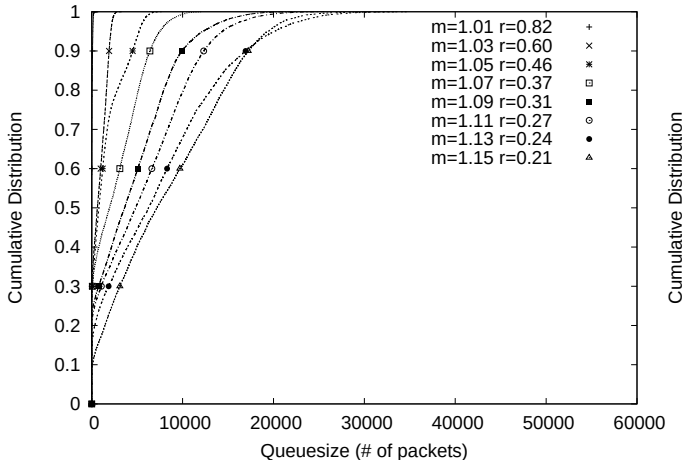
Figure 17: Effect of m on utilisation with N = 30



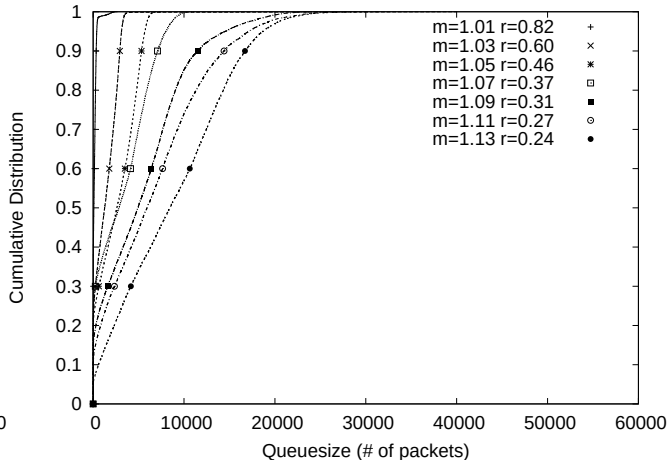
(a) n=2



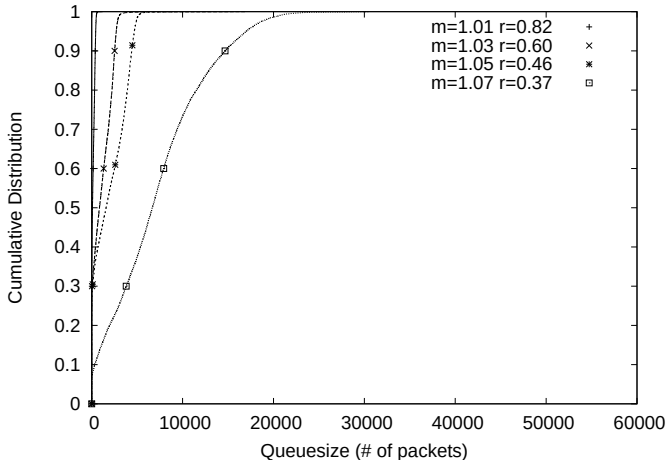
(b) n=4



(c) n=24

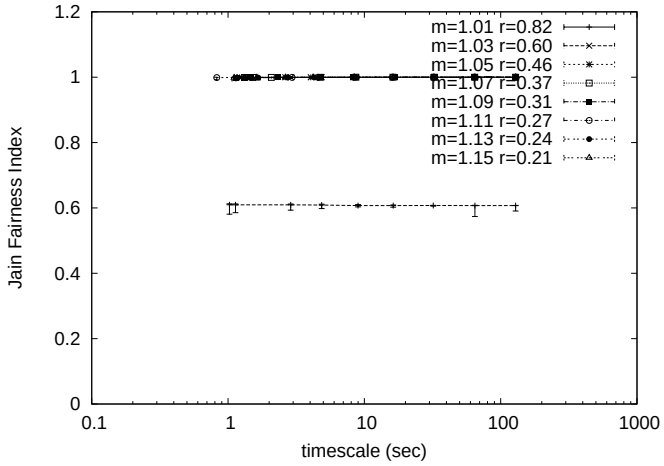


(d) n=50

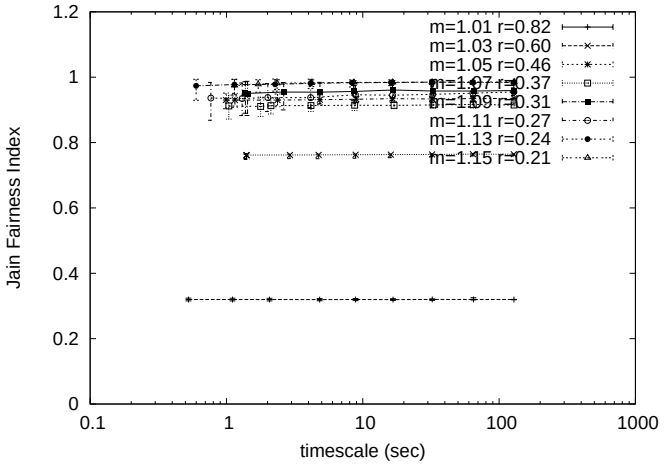


(e) n=100

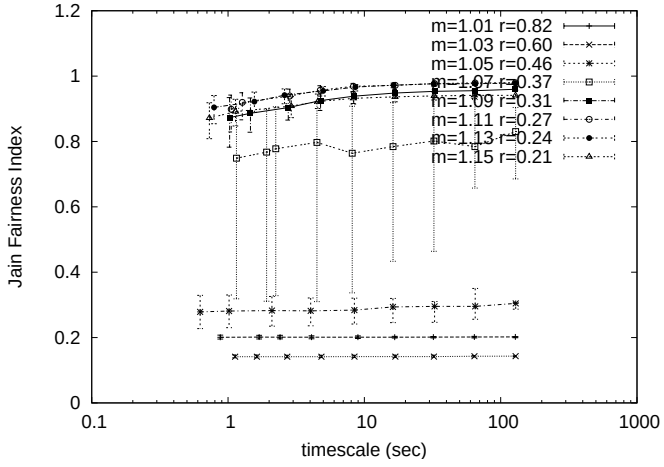
Figure 18: Effect of m on queuesizes with $N = 40$



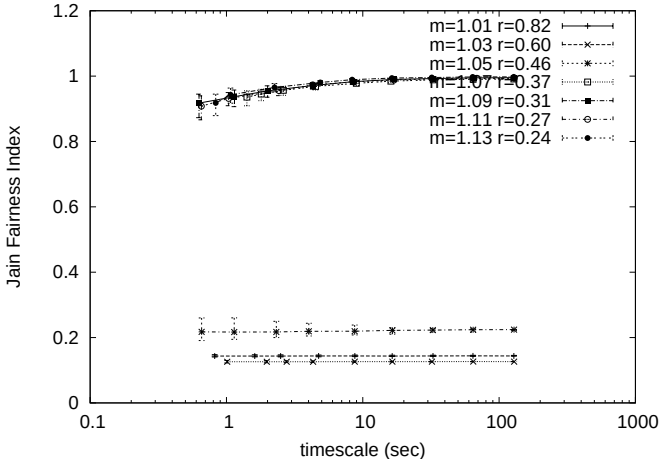
(a) n=2



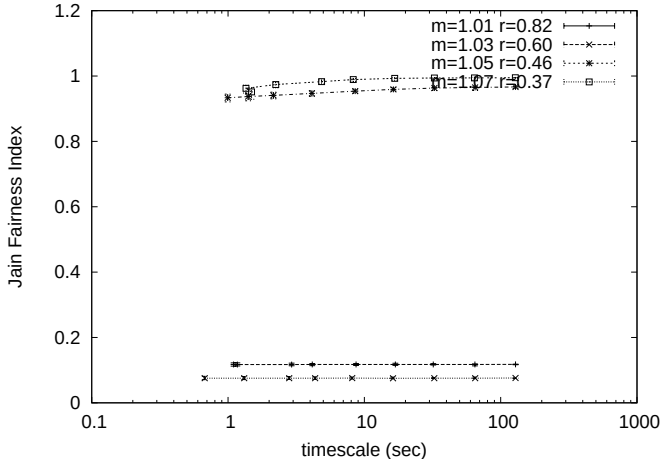
(b) n=4



(c) n=24

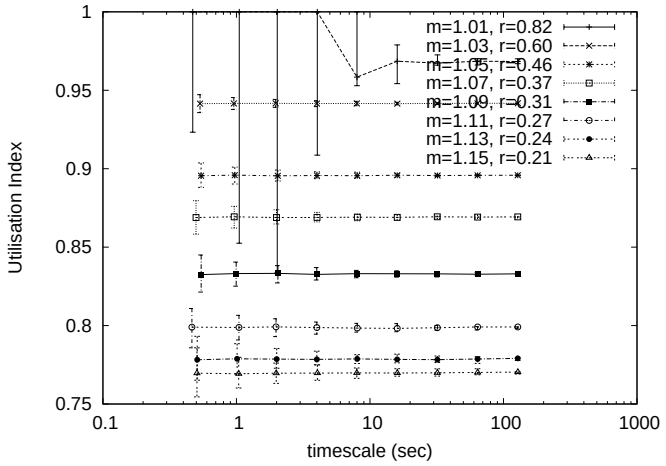


(d) n=50

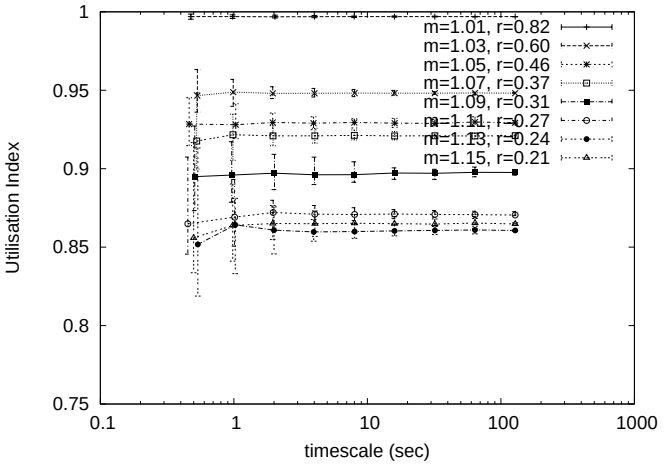


(e) n=100

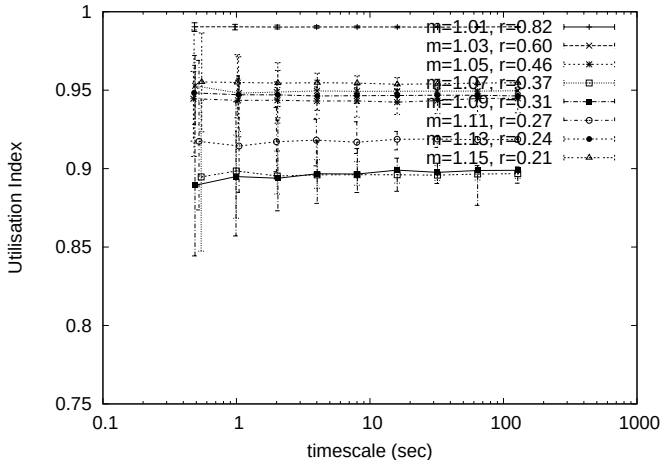
Figure 19: Effect of m on fairness with $N = 40$



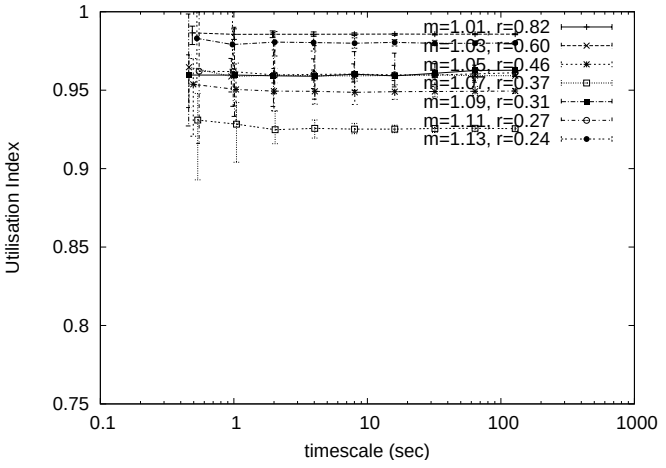
(a) n=2



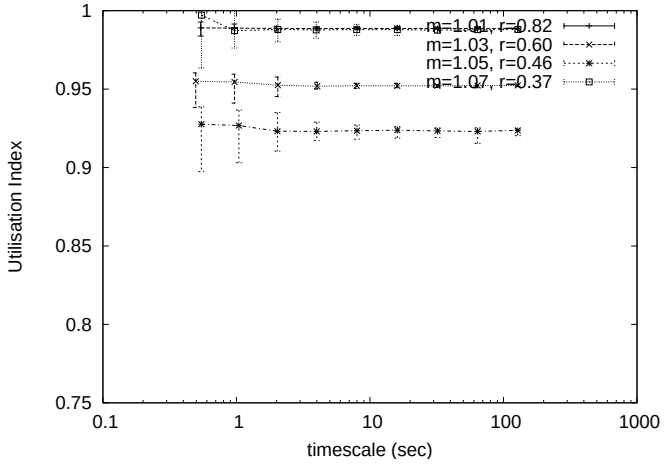
(b) n=4



(c) n=24

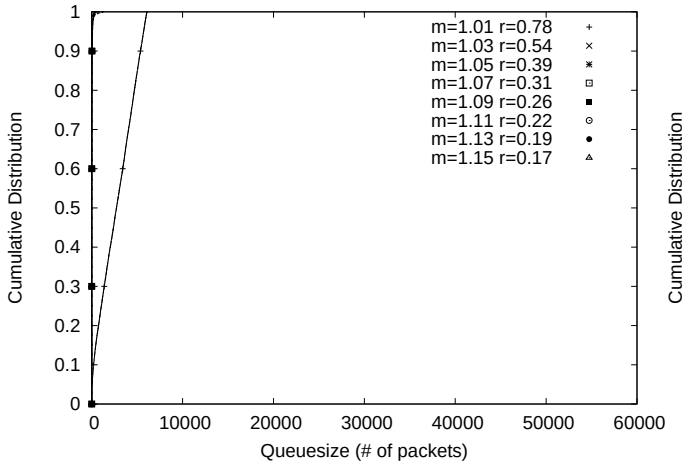


(d) n=50

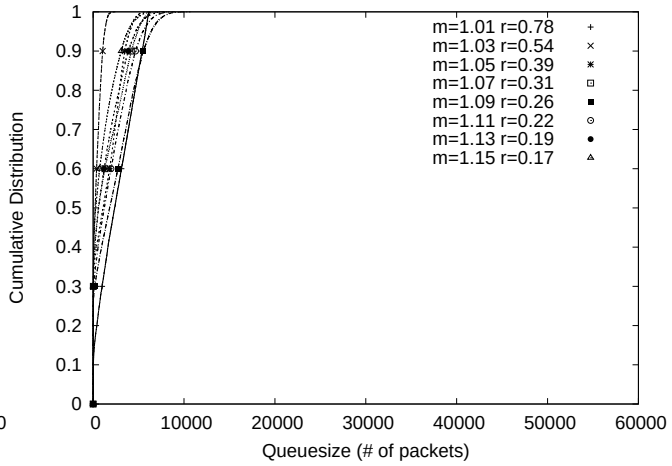


(e) n=100

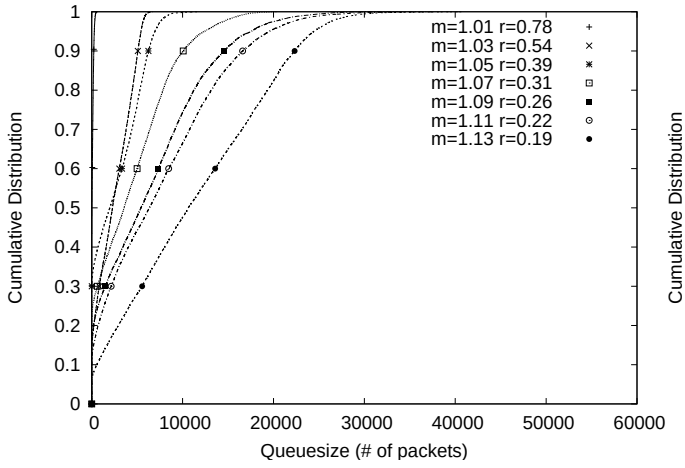
Figure 20: Effect of m on utilisation with $N = 40$



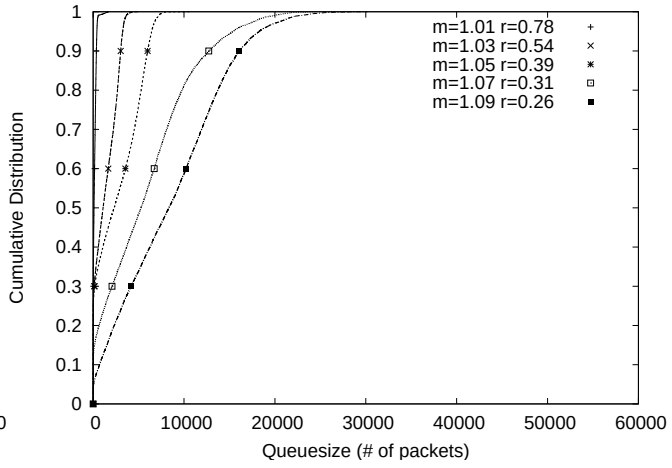
(a) n=2



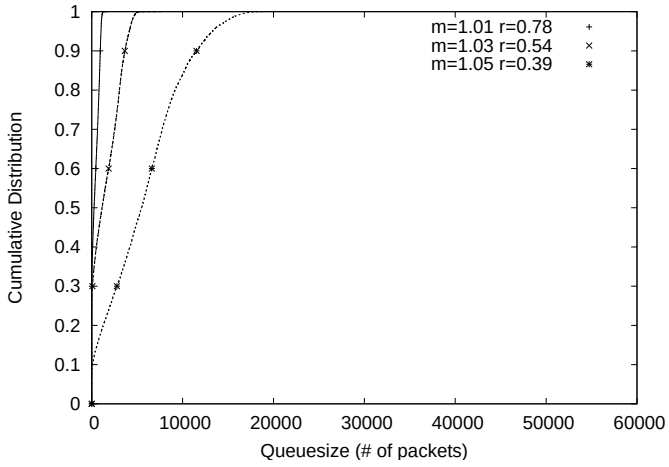
(b) n=4



(c) n=24

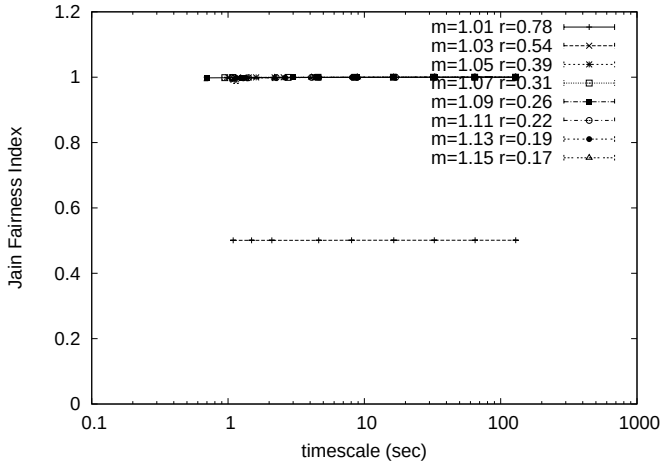


(d) n=50

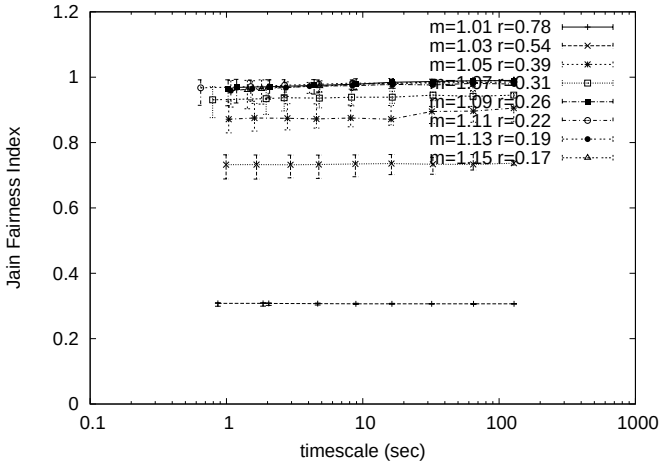


(e) n=100

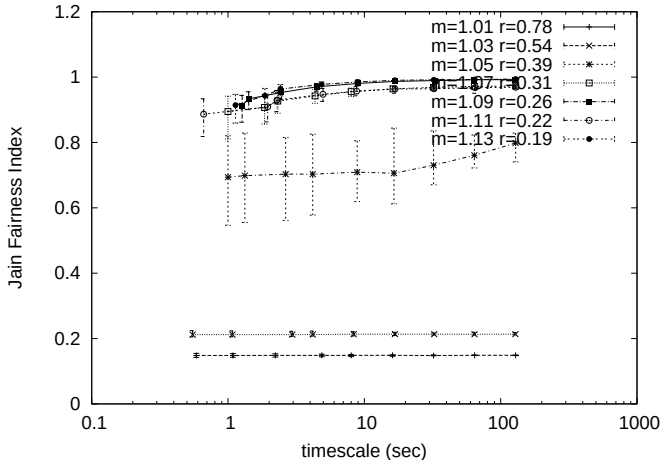
Figure 21: Effect of m on queuesizes with $N = 50$



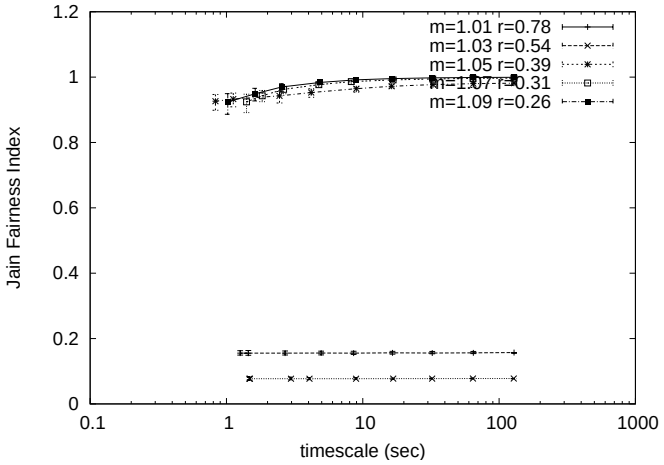
(a) n=2



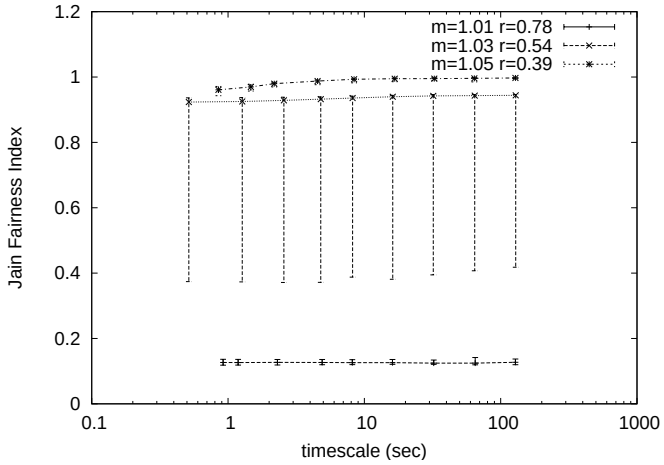
(b) n=4



(c) n=24

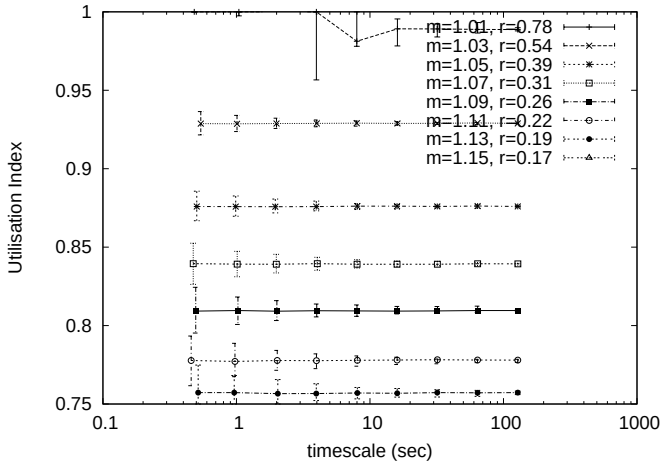


(d) n=50

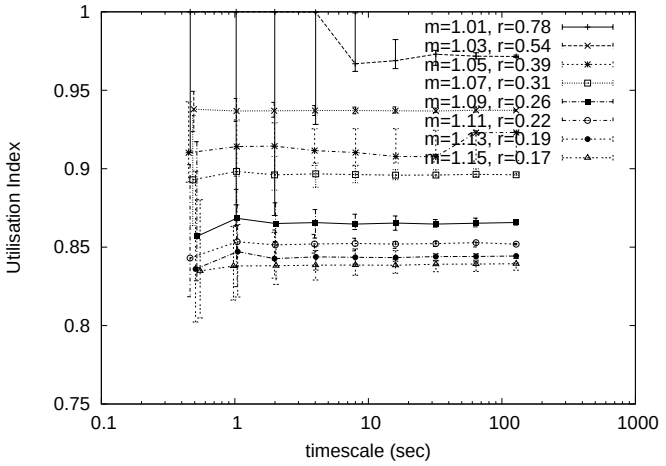


(e) n=100

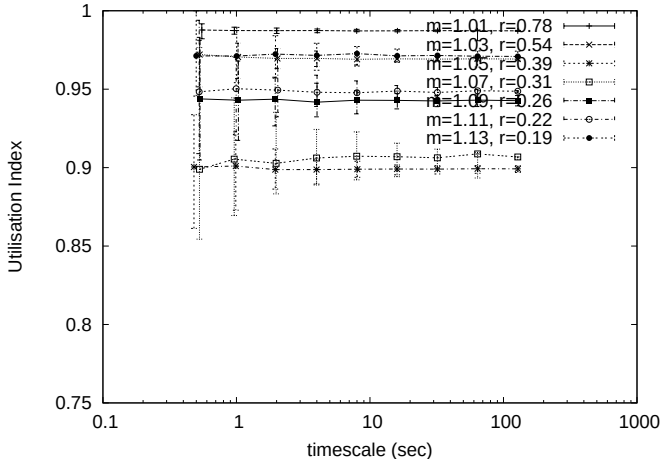
Figure 22: Effect of m on fairness with N = 50



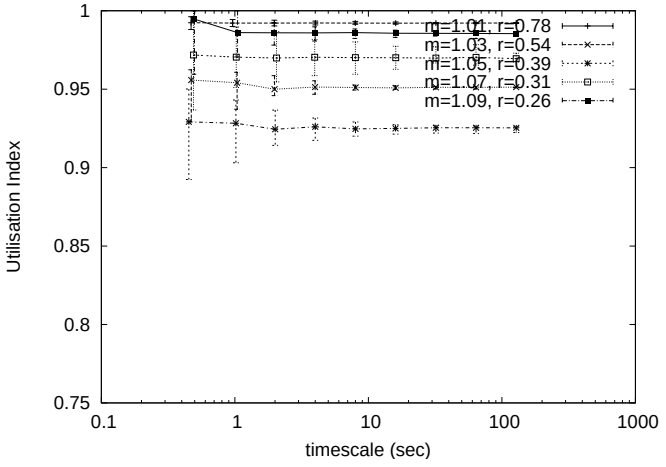
(a) n=2



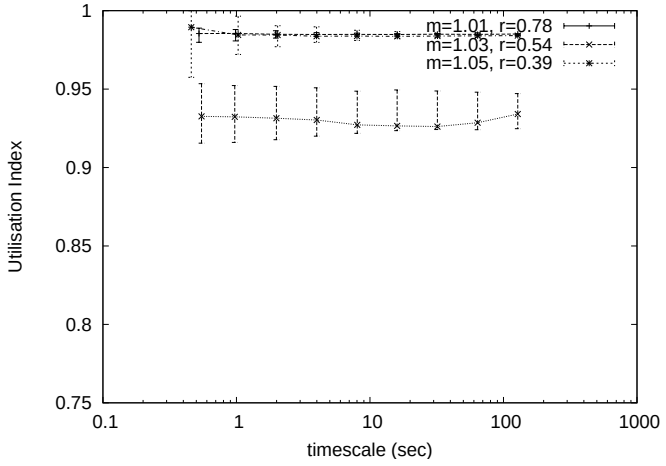
(b) n=4



(c) n=24

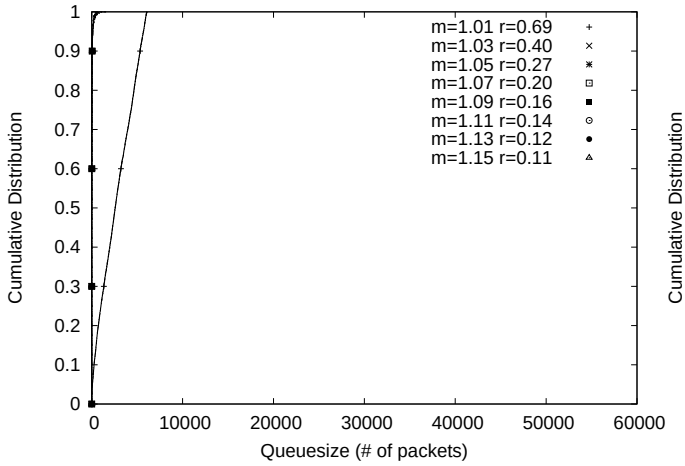


(d) n=50

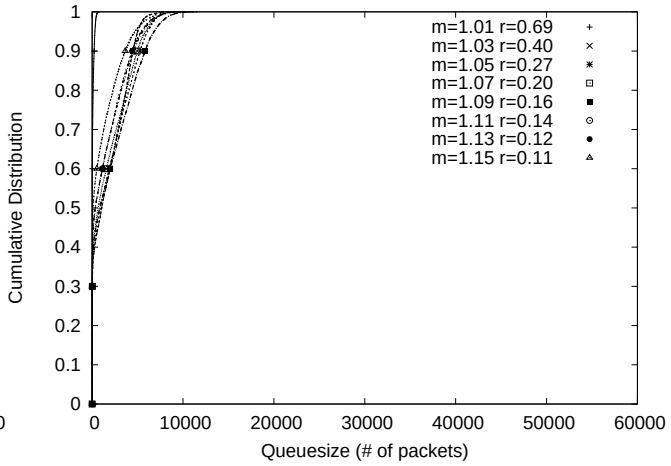


(e) n=100

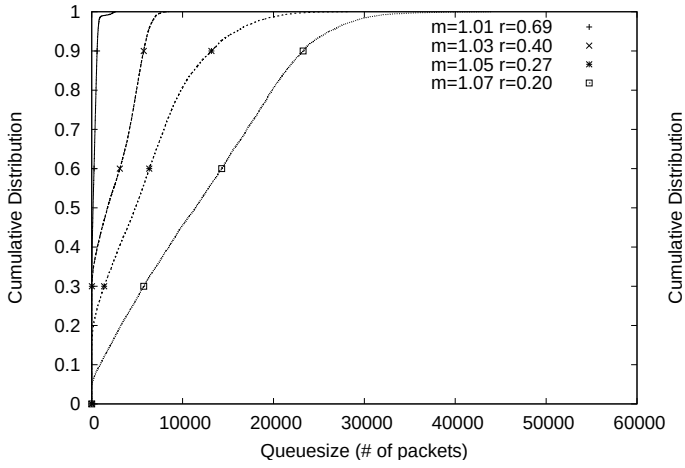
Figure 23: Effect of m on utilisation with N = 50



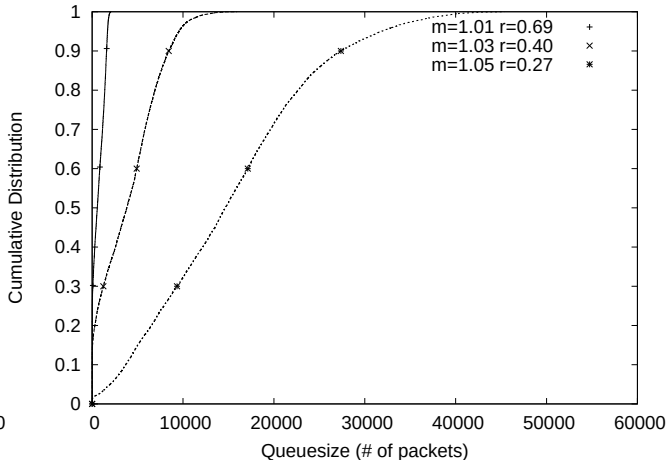
(a) n=2



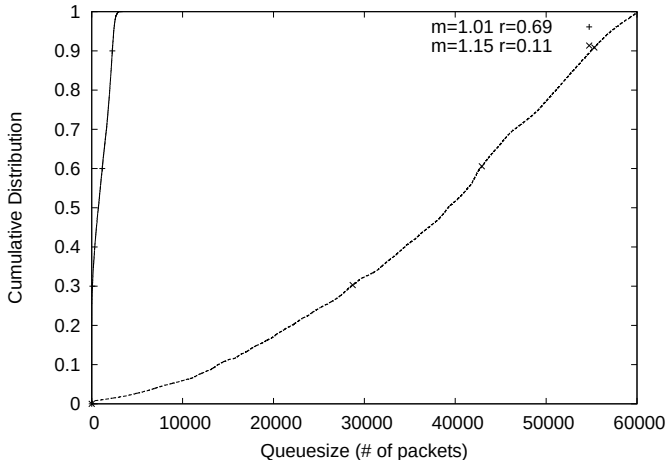
(b) n=4



(c) n=24

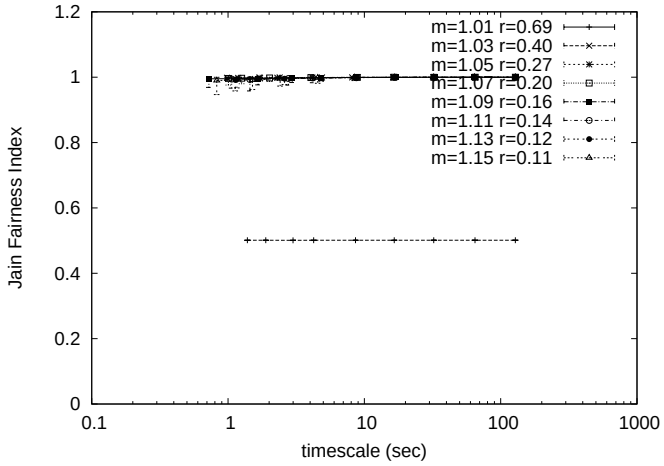


(d) n=50

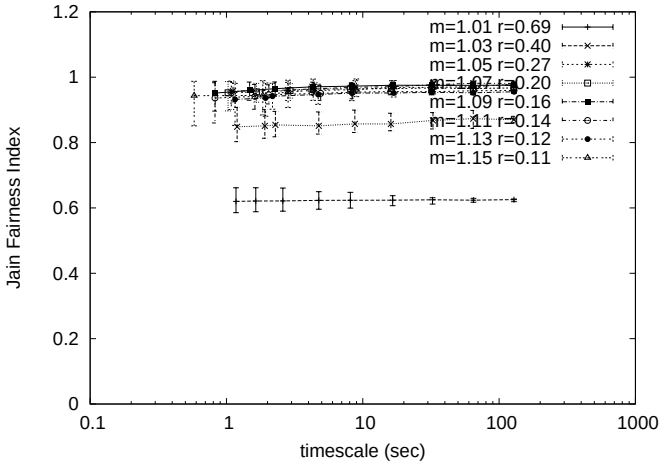


(e) n=100

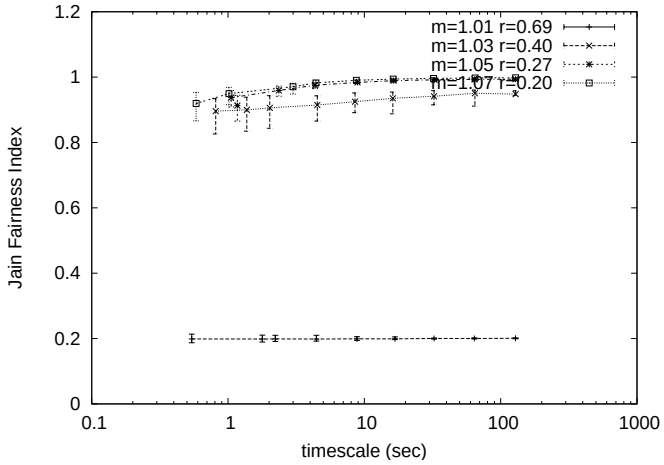
Figure 24: Effect of m on queuesizes with N = 80



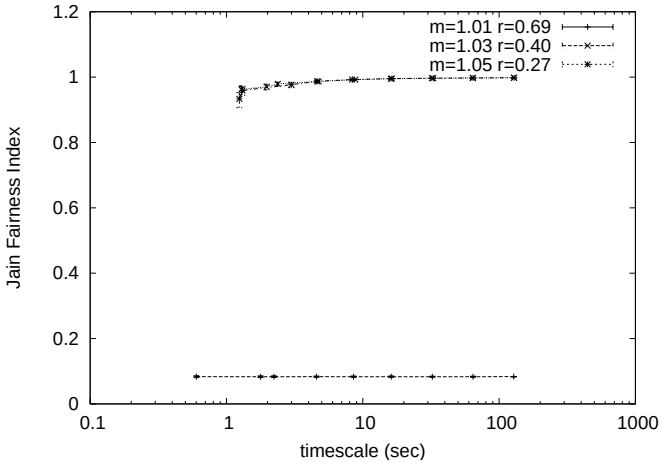
(a) n=2



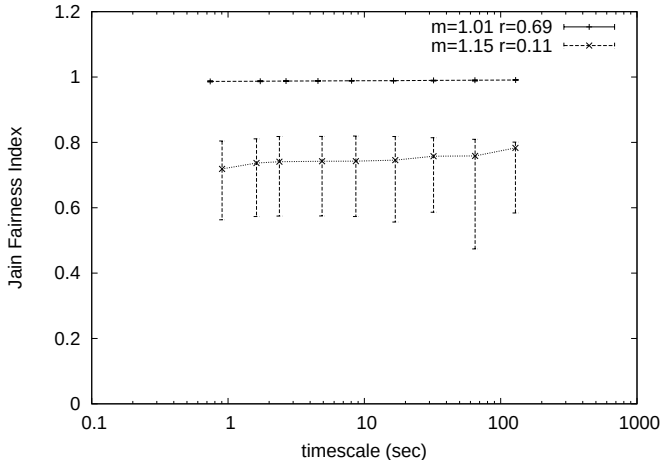
(b) n=4



(c) n=24

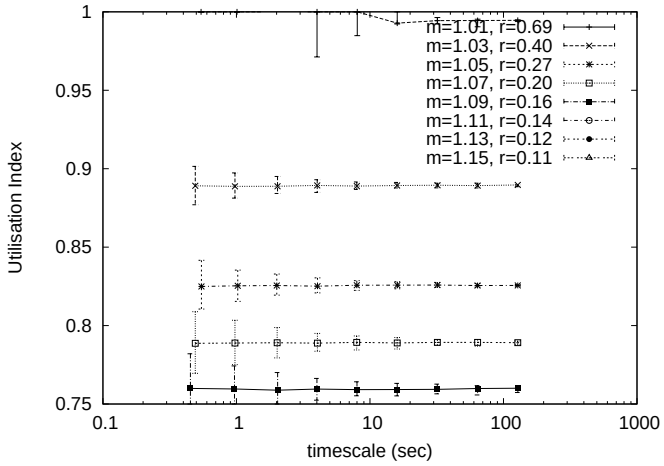


(d) n=50

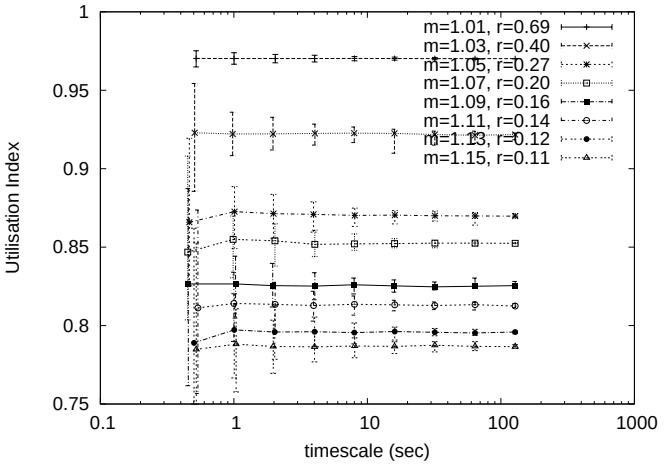


(e) n=100

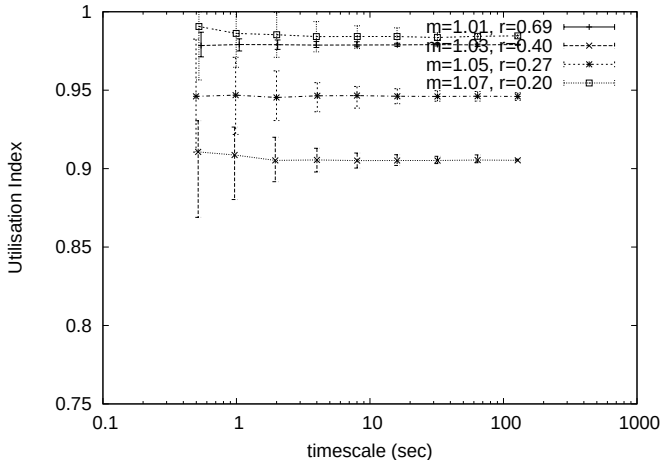
Figure 25: Effect of m on fairness with N = 80



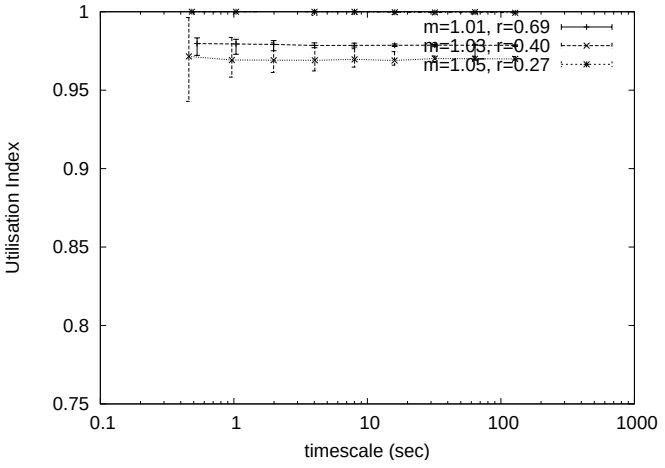
(a) n=2



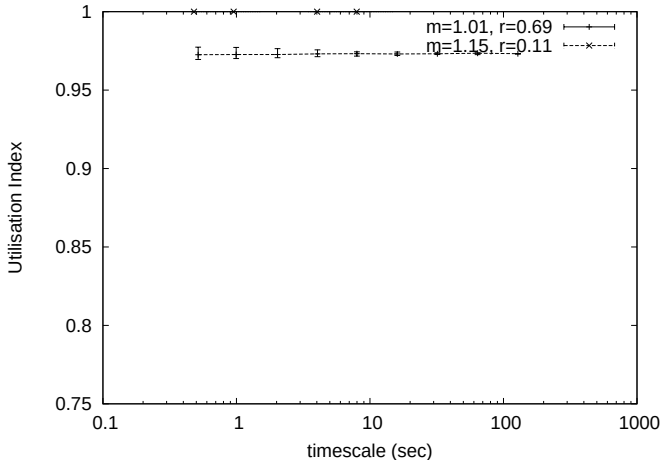
(b) n=4



(c) n=24

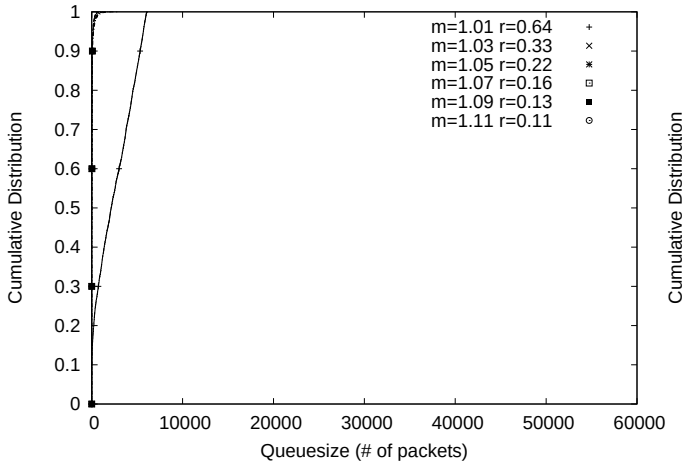


(d) n=50

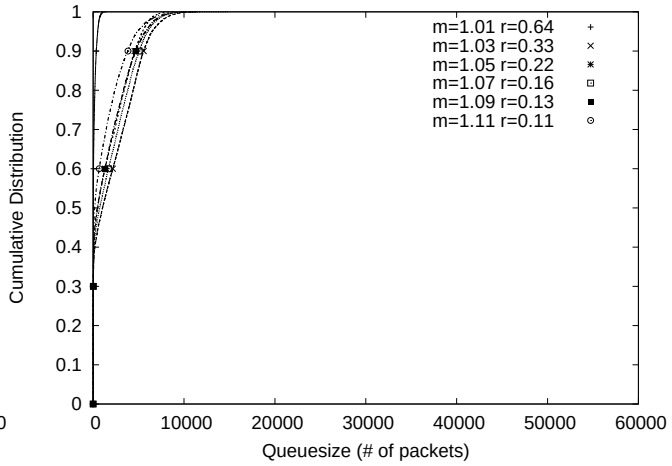


(e) n=100

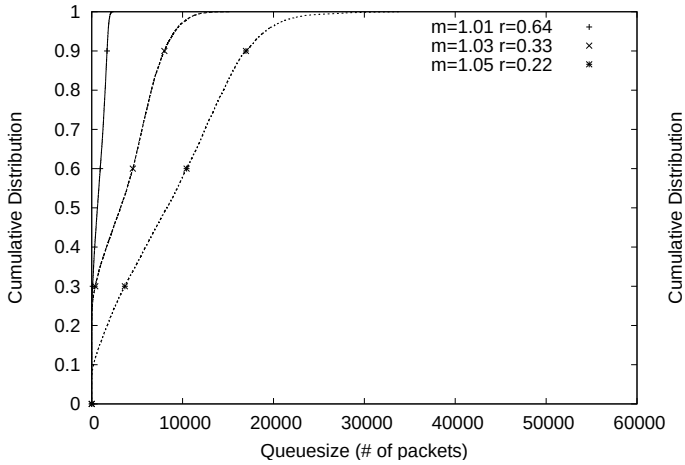
Figure 26: Effect of m on utilisation with N = 80



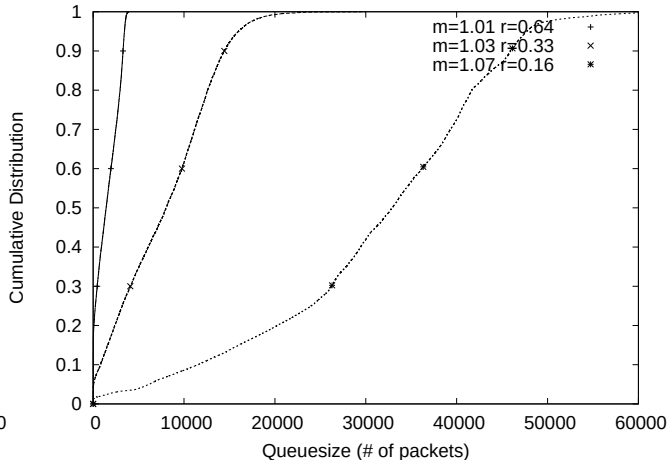
(a) n=2



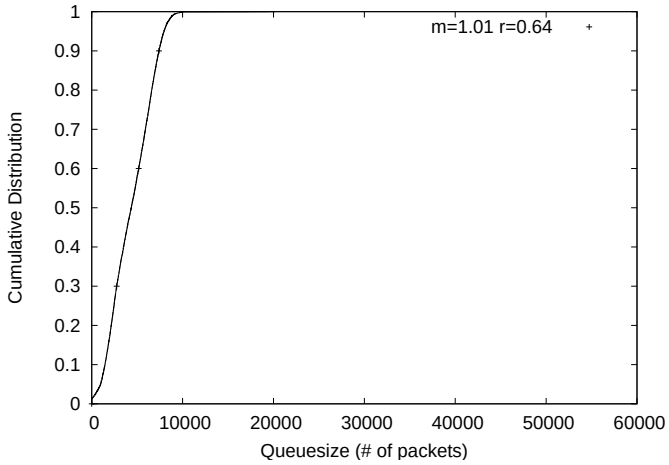
(b) n=4



(c) n=24

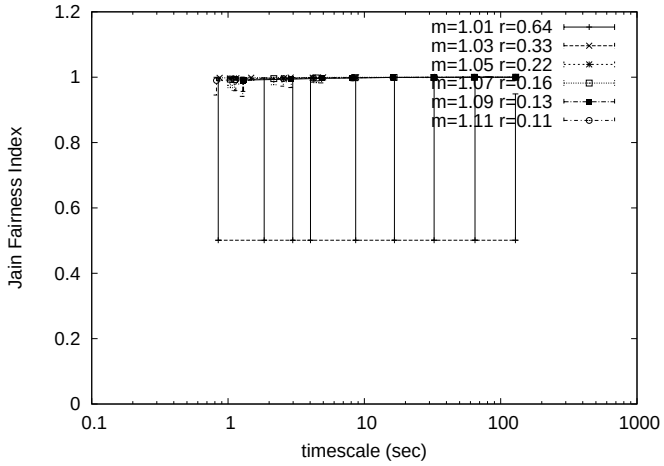


(d) n=50

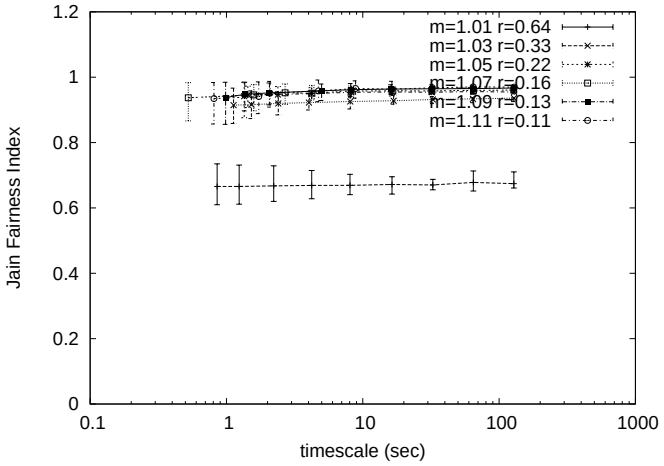


(e) n=100

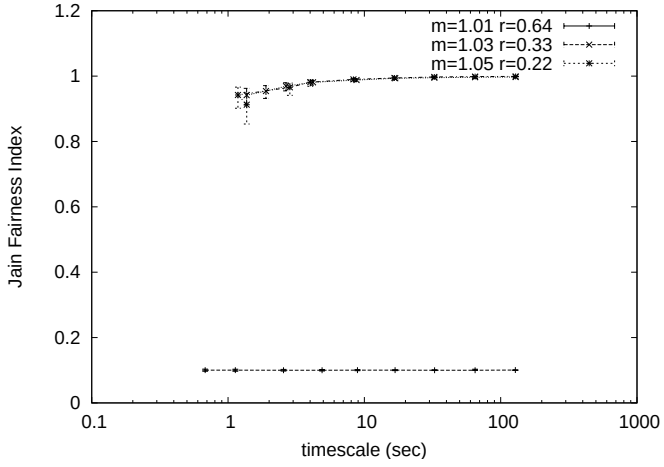
Figure 27: Effect of m on queuesizes with N = 100



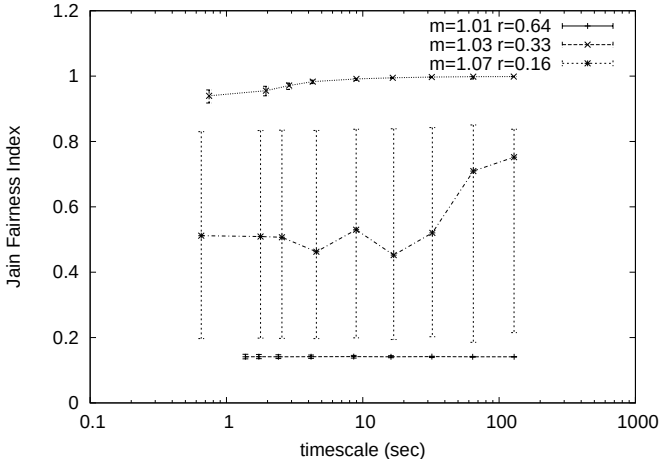
(a) n=2



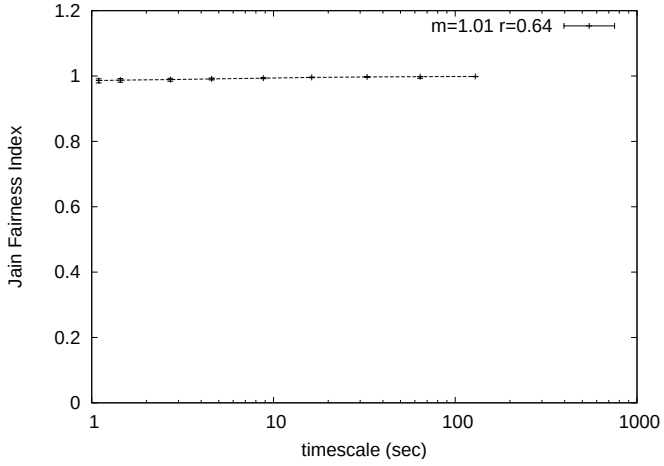
(b) n=4



(c) n=24

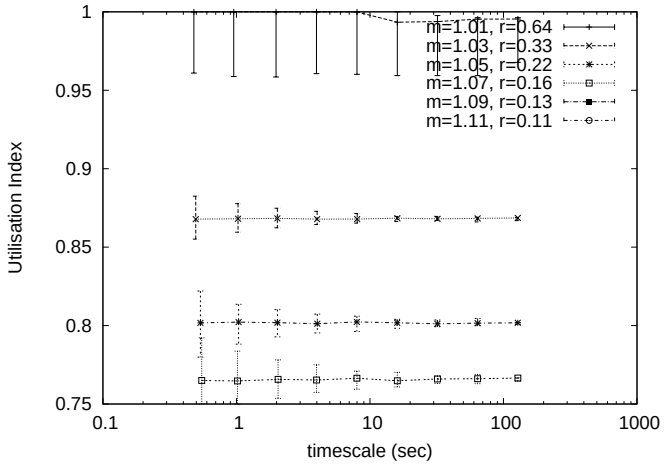


(d) n=50

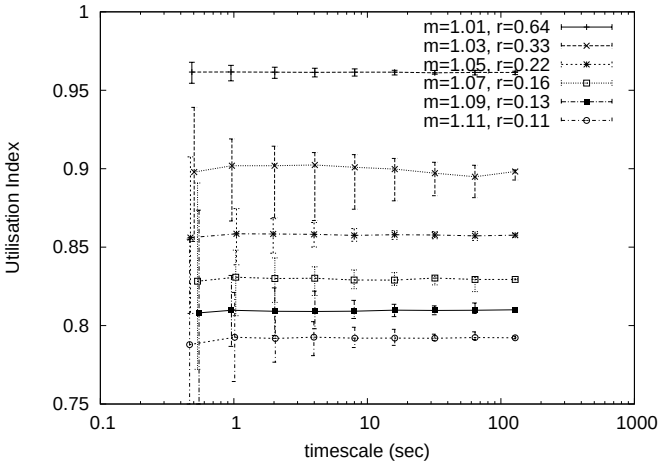


(e) n=100

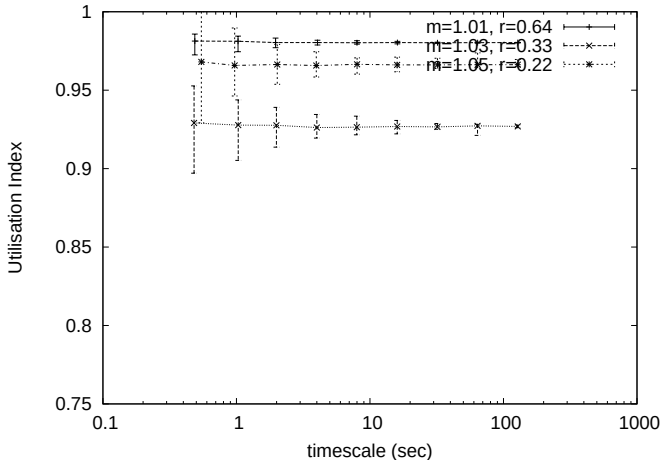
Figure 28: Effect of m on fairness with N = 100



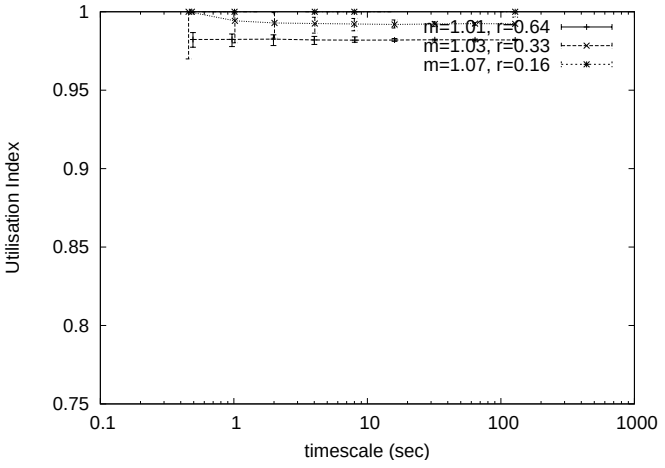
(a) n=2



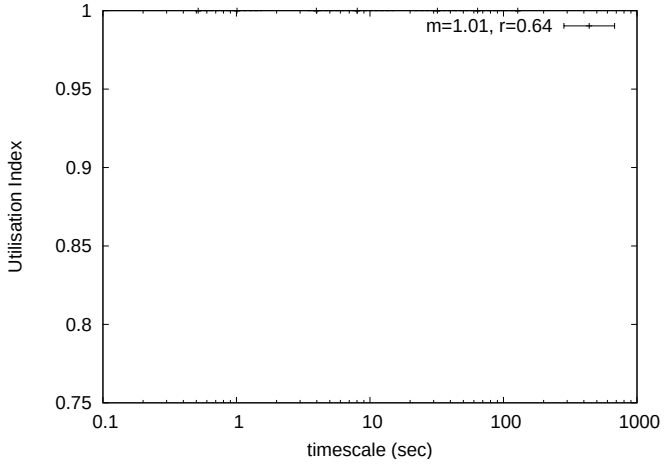
(b) n=4



(c) n=24

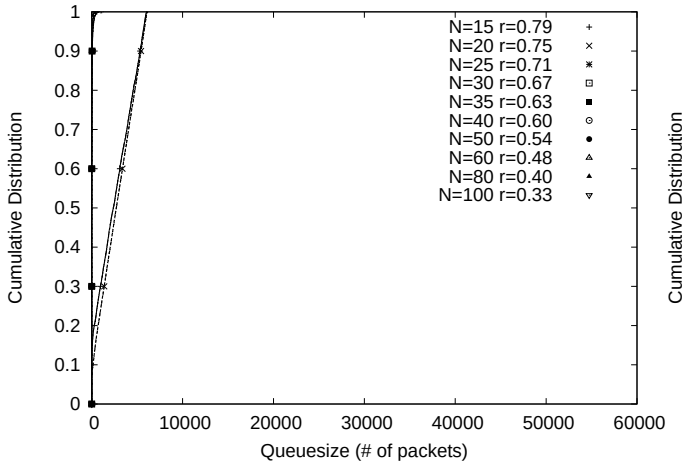


(d) n=50

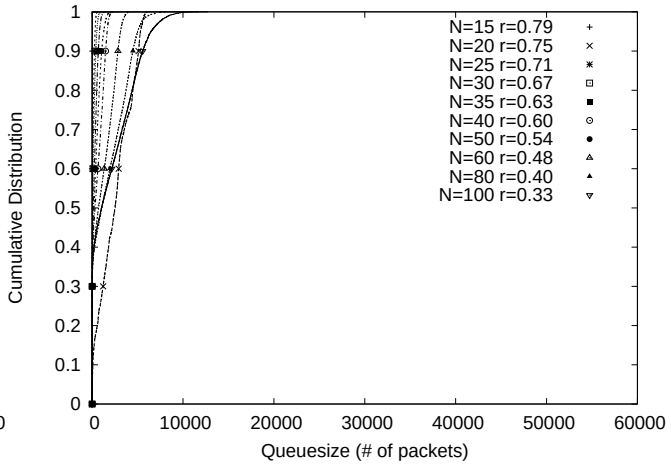


(e) n=100

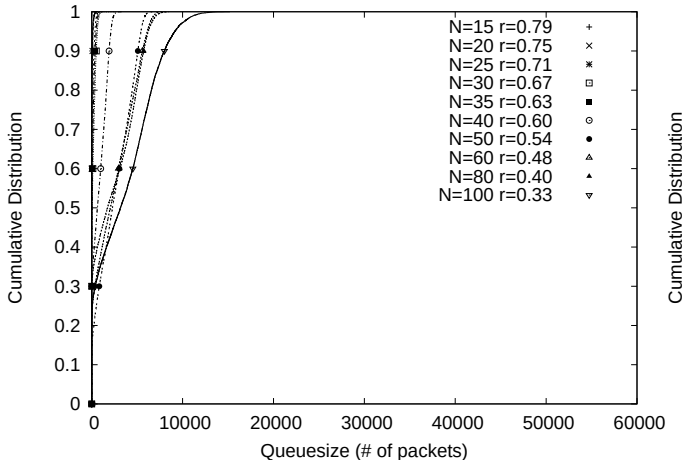
Figure 29: Effect of m on utilisation with $N = 100$



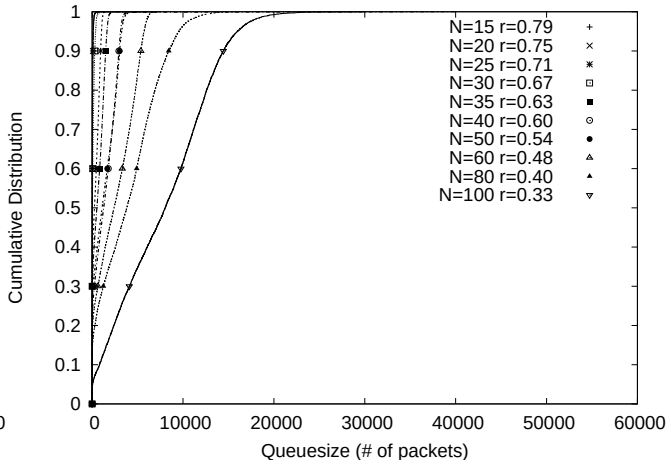
(a) n=2



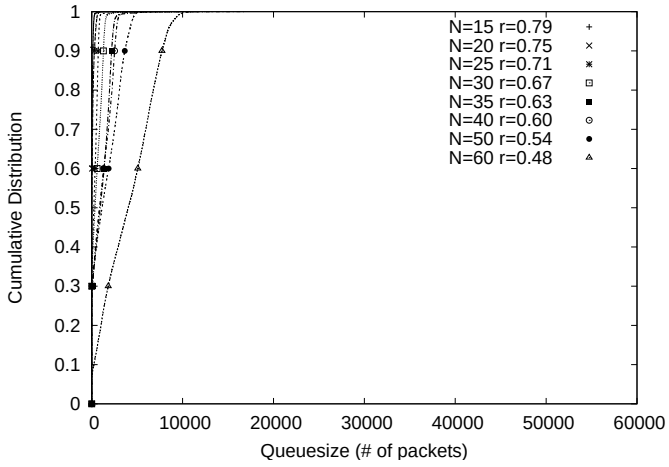
(b) n=4



(c) n=24

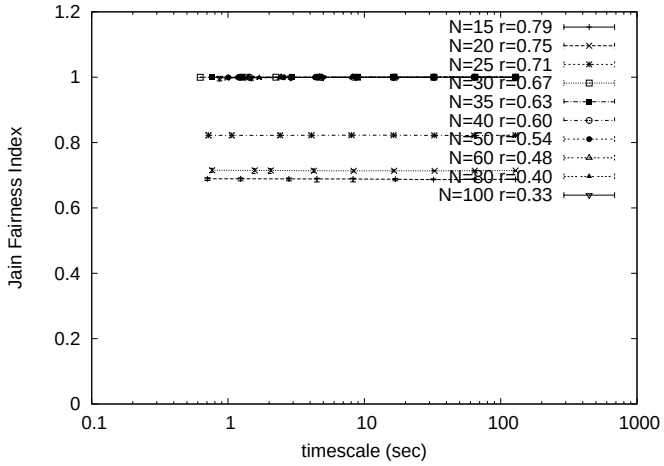


(d) n=50

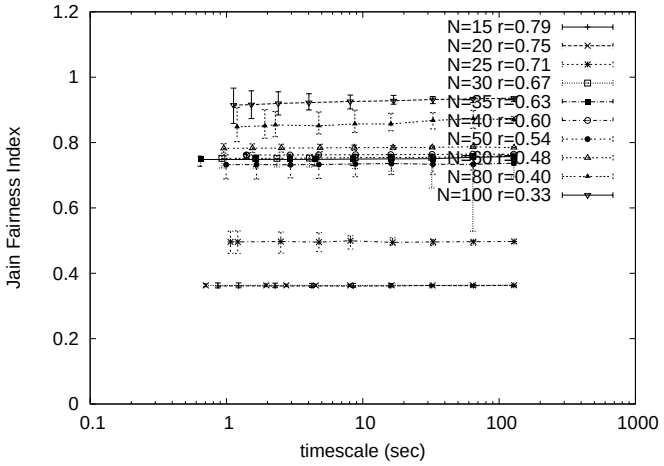


(e) n=100

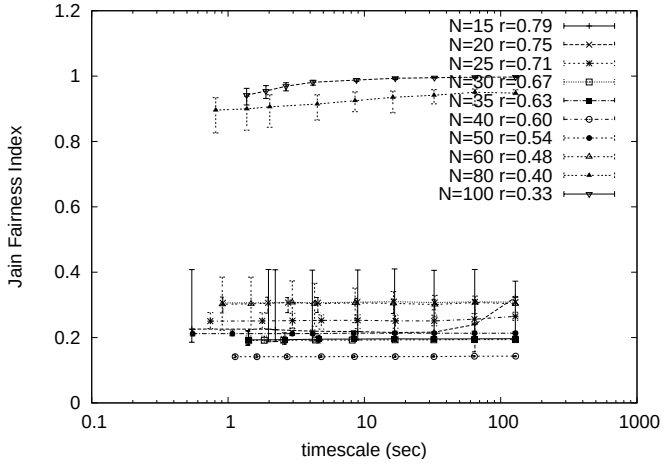
Figure 30: Effect of N on queuesizes with m = 1.03



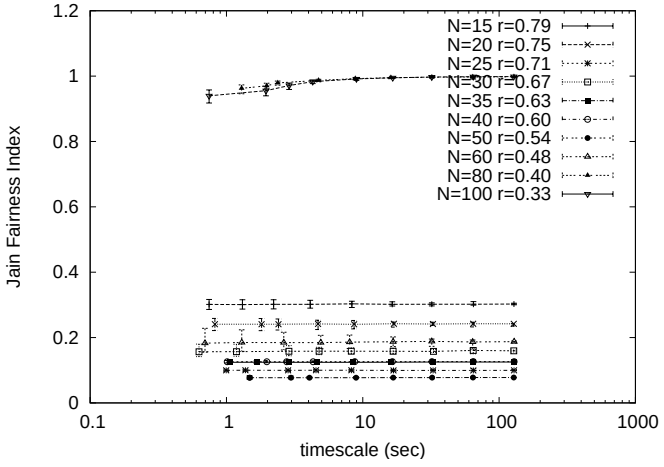
(a) n=2



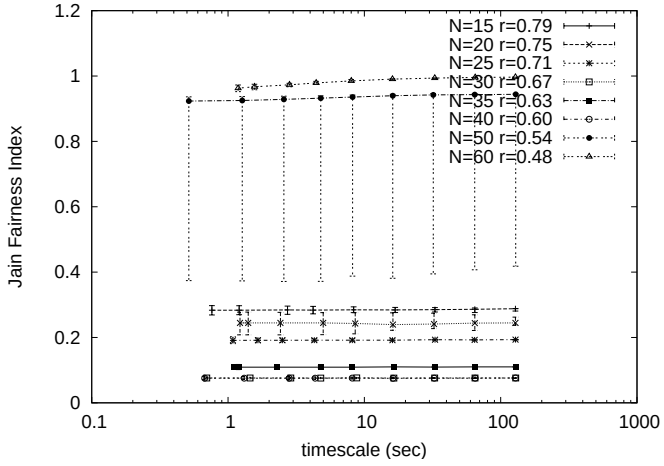
(b) n=4



(c) n=24

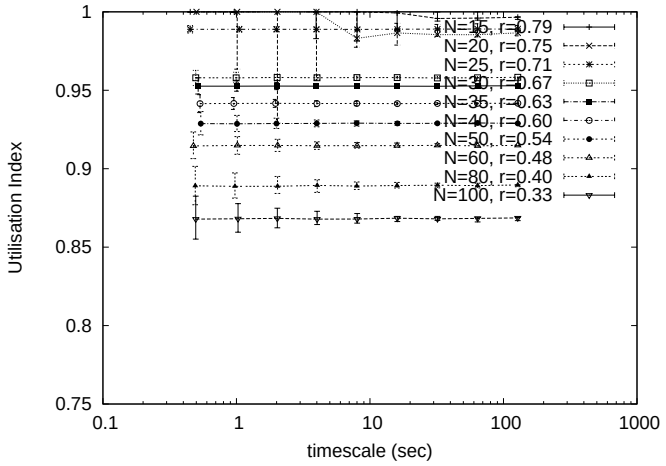


(d) n=50

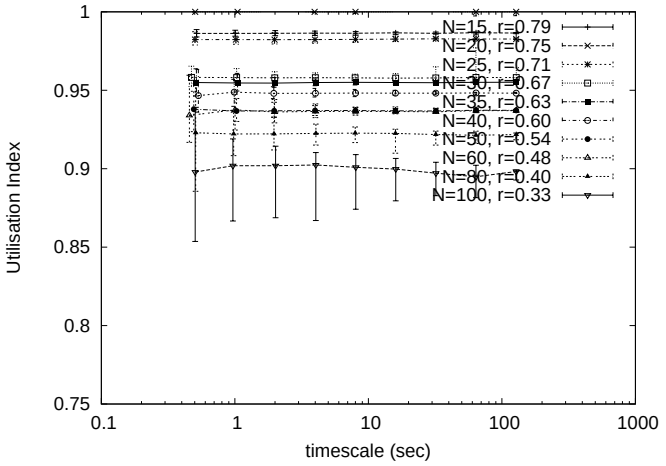


(e) n=100

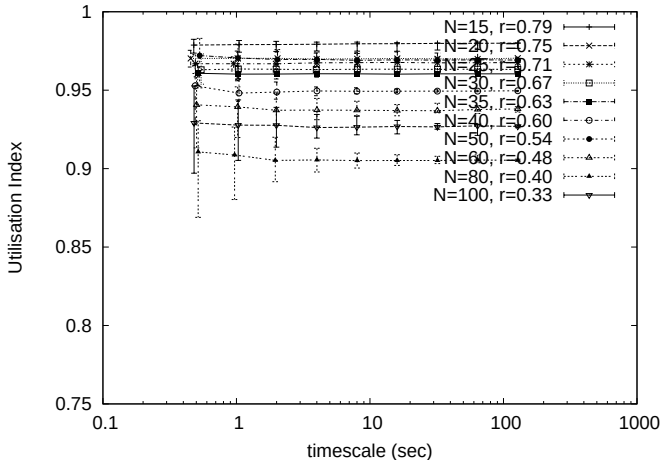
Figure 31: Effect of N on fairness with m = 1.03



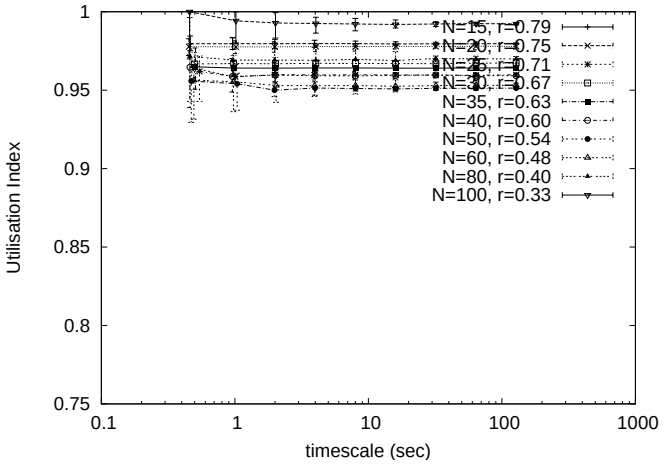
(a) n=2



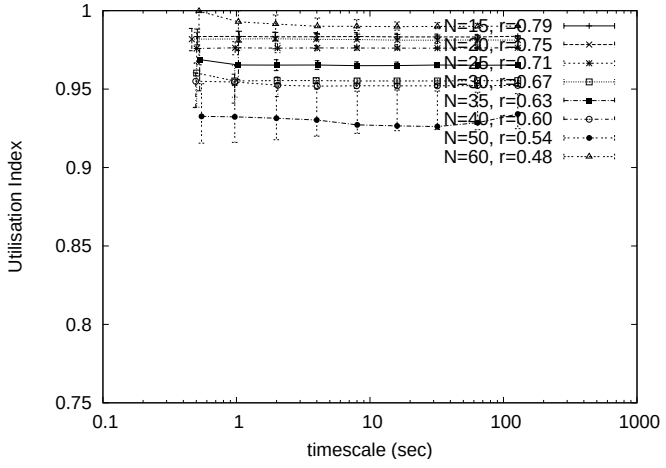
(b) n=4



(c) n=24

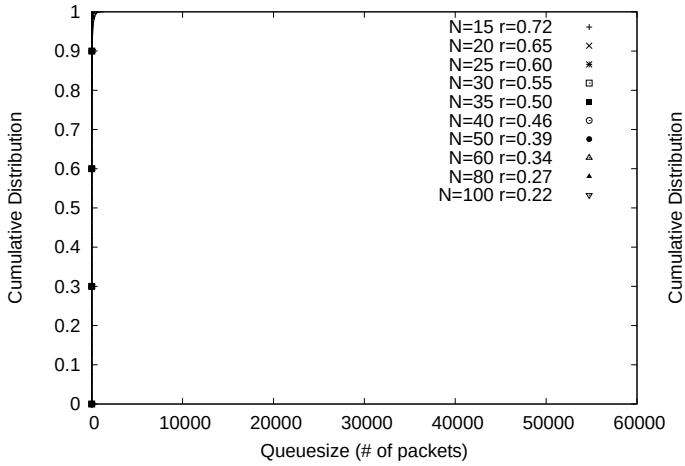


(d) n=50

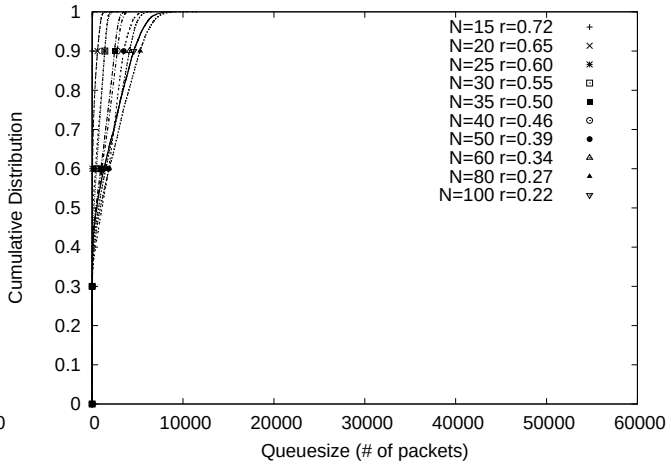


(e) n=100

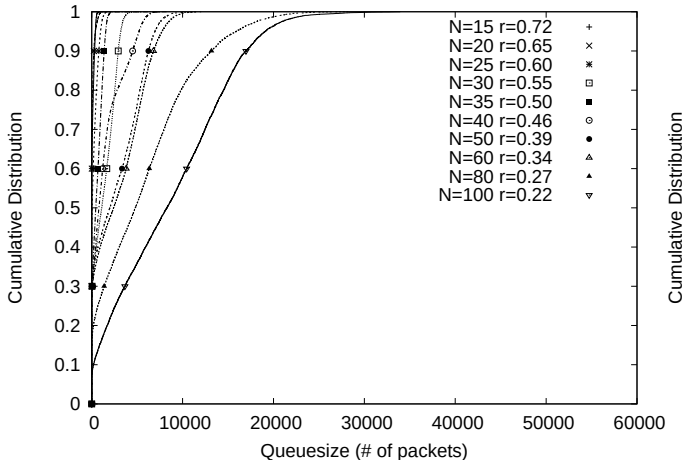
Figure 32: Effect of N on utilisation with m = 1.03



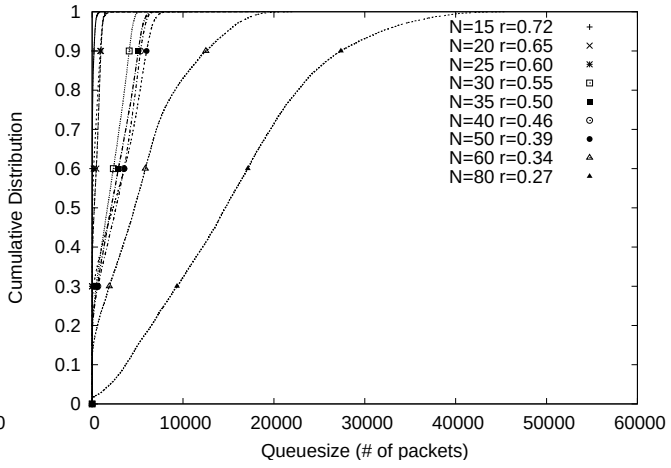
(a) n=2



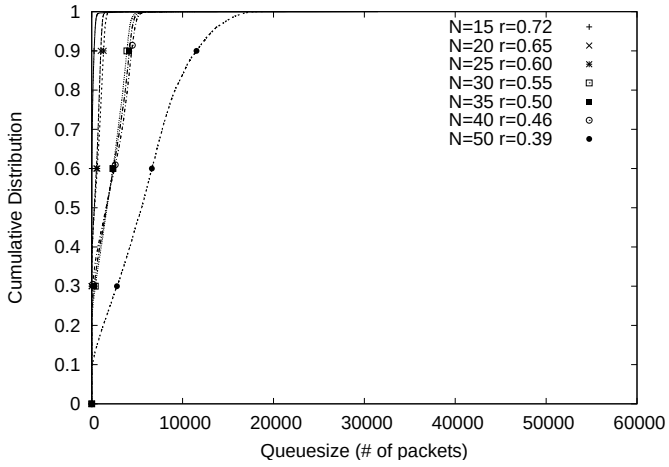
(b) n=4



(c) n=24

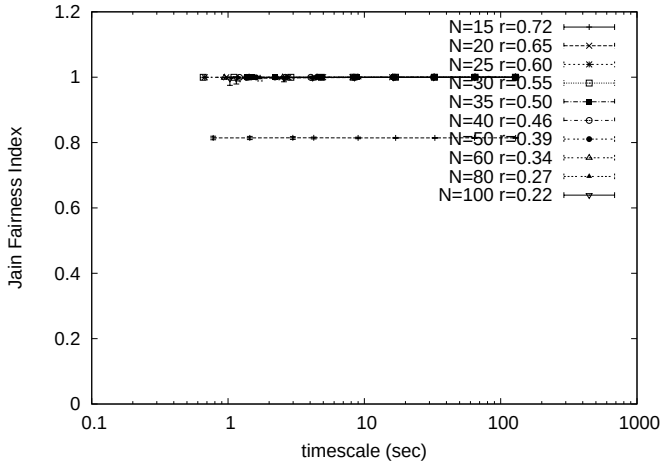


(d) n=50

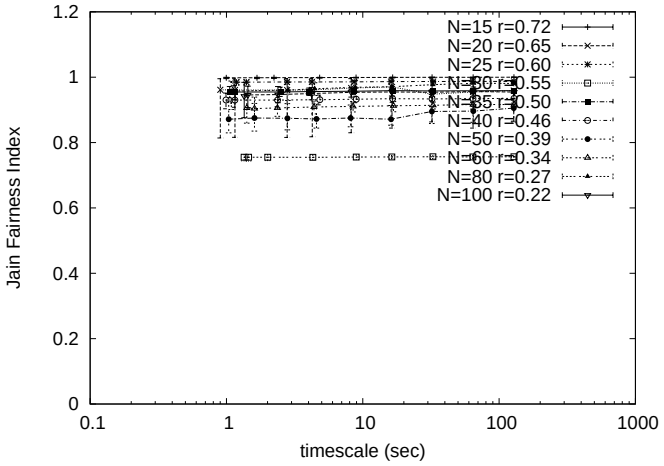


(e) n=100

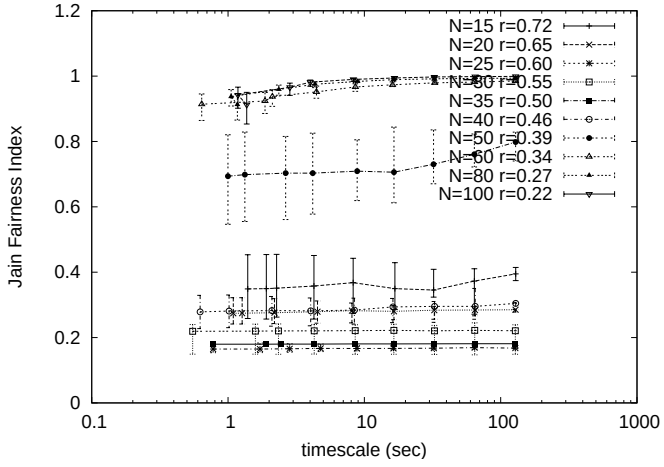
Figure 33: Effect of N on queuesizes with m = 1.05



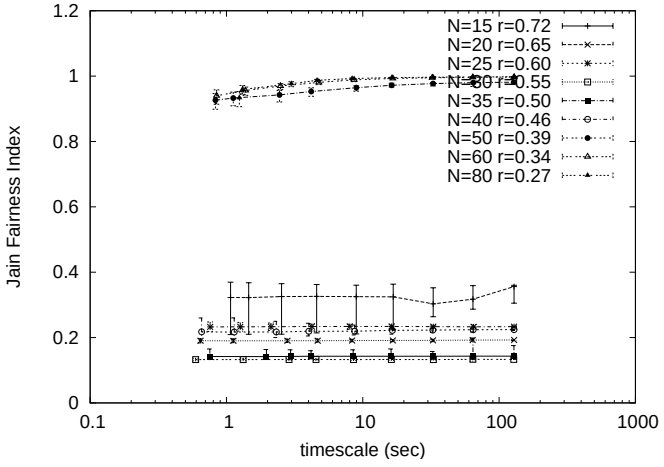
(a) n=2



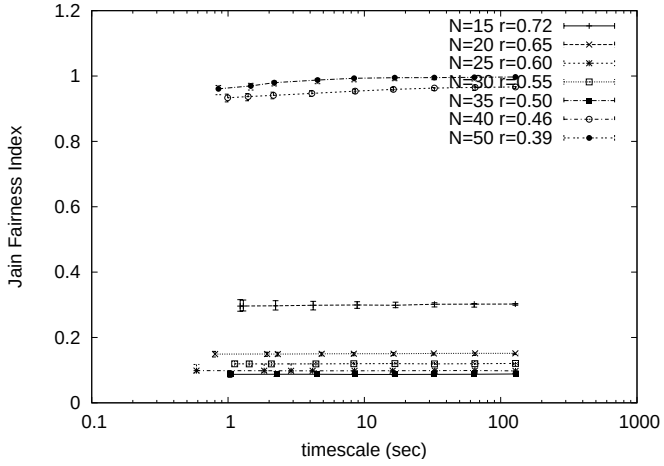
(b) n=4



(c) n=24

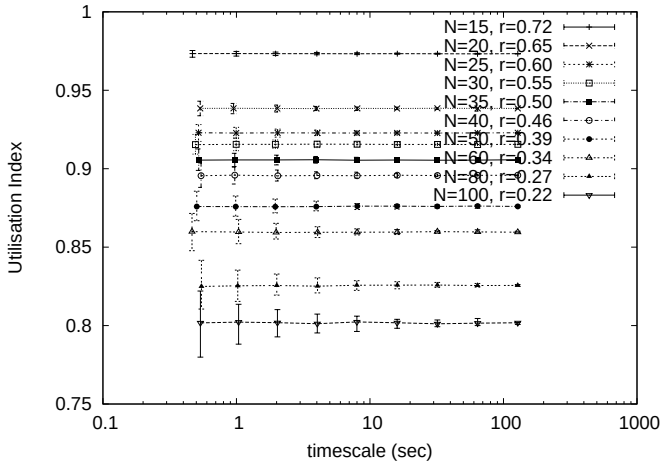


(d) n=50

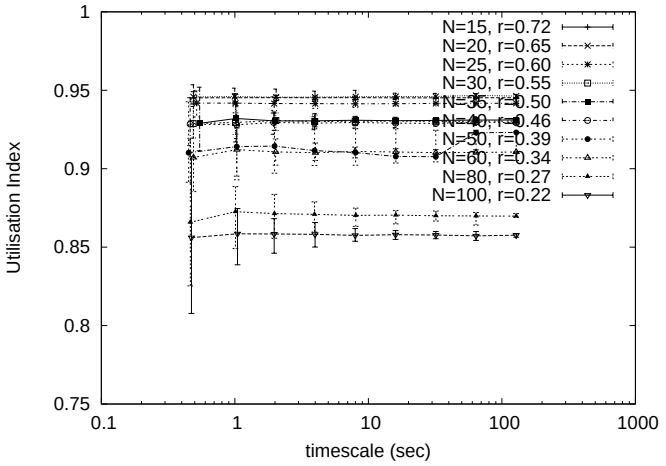


(e) n=100

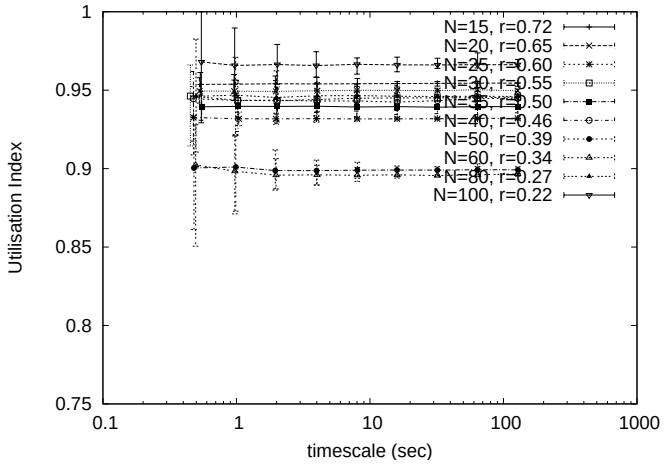
Figure 34: Effect of N on fairness with m = 1.05



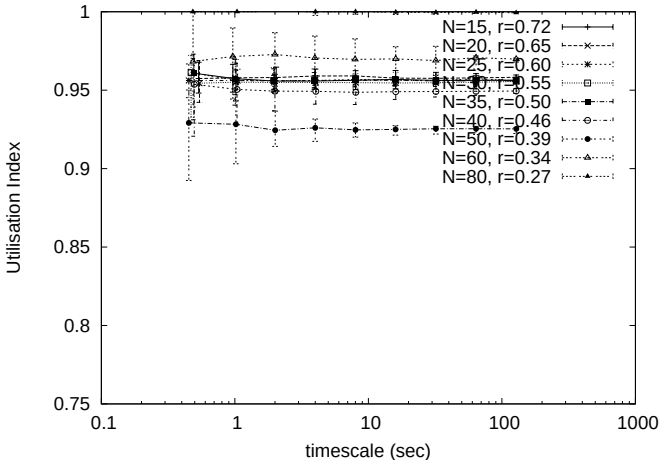
(a) n=2



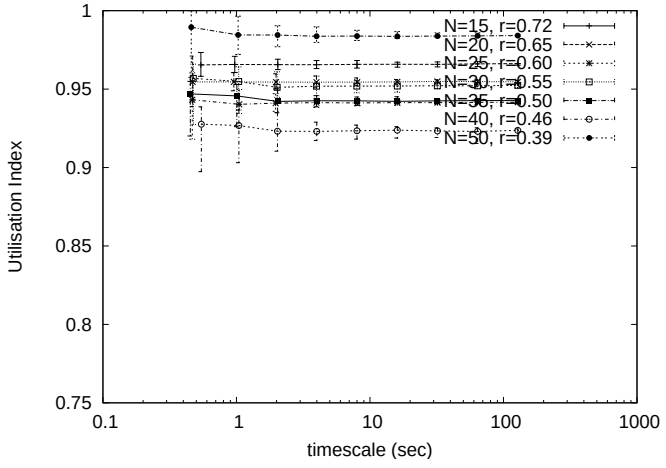
(b) n=4



(c) n=24

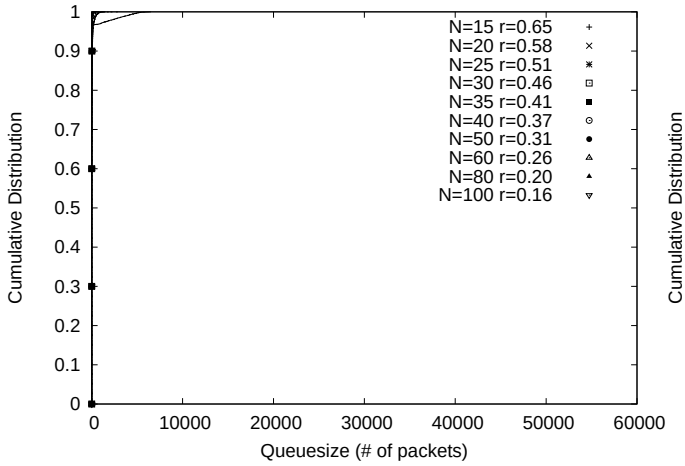


(d) n=50

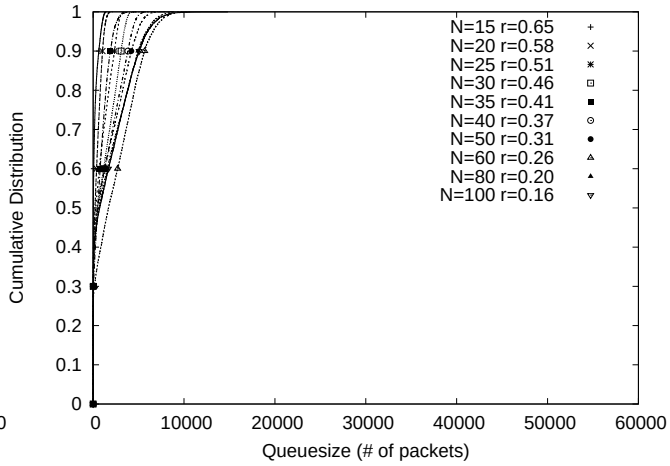


(e) n=100

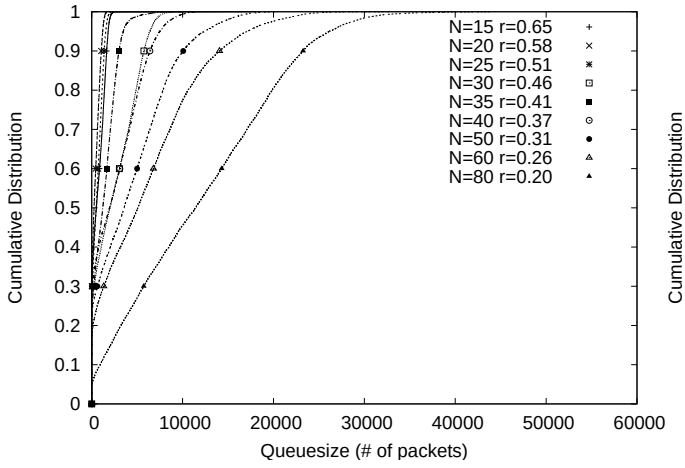
Figure 35: Effect of N on utilisation with m = 1.05



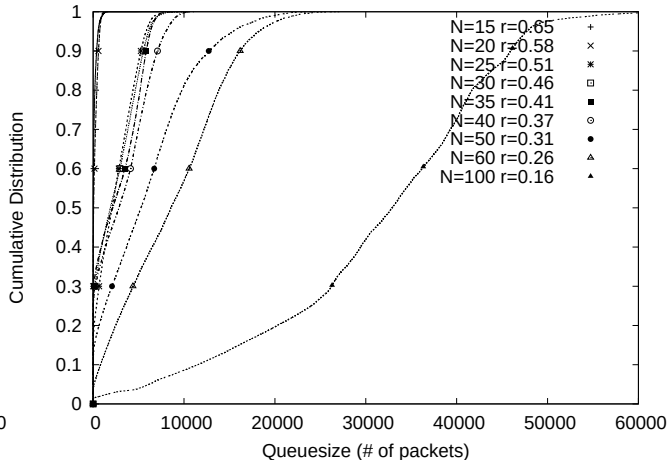
(a) n=2



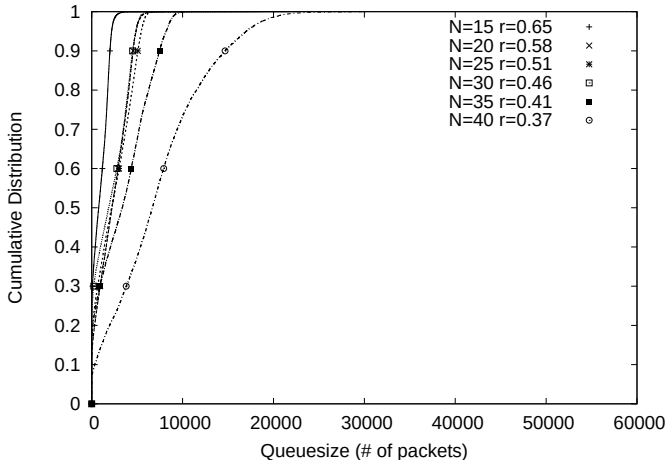
(b) n=4



(c) n=24

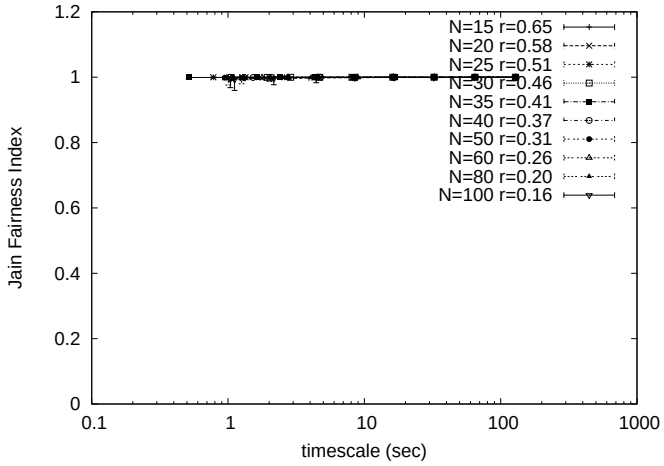


(d) n=50

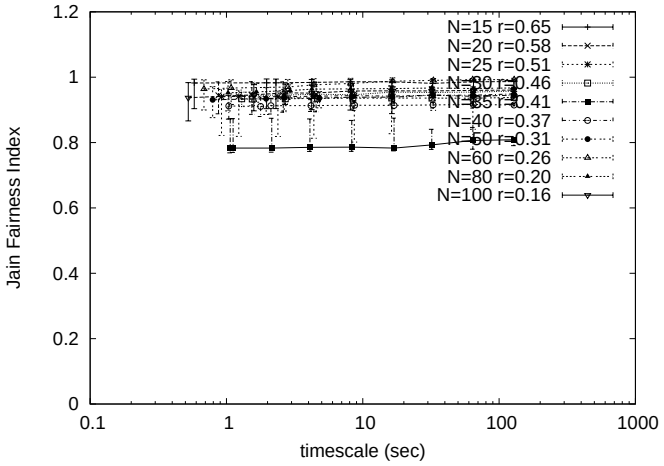


(e) n=100

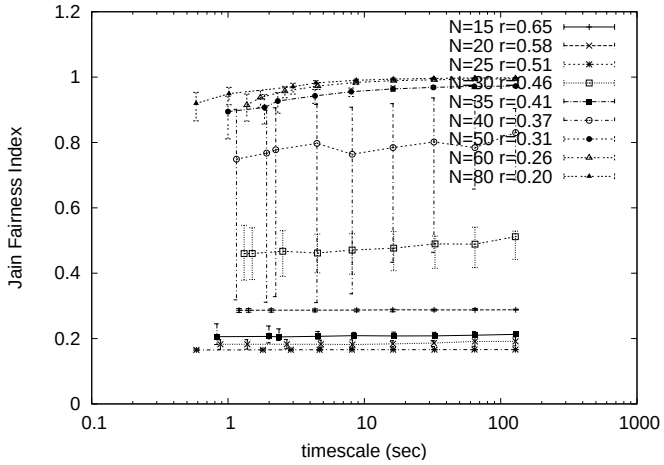
Figure 36: Effect of N on queuesizes with m = 1.07



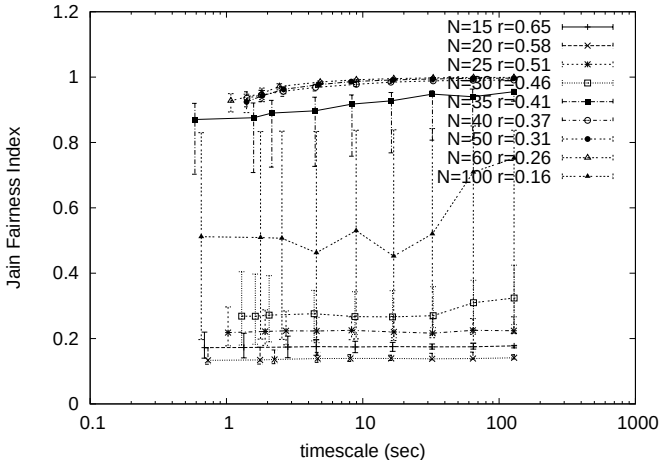
(a) n=2



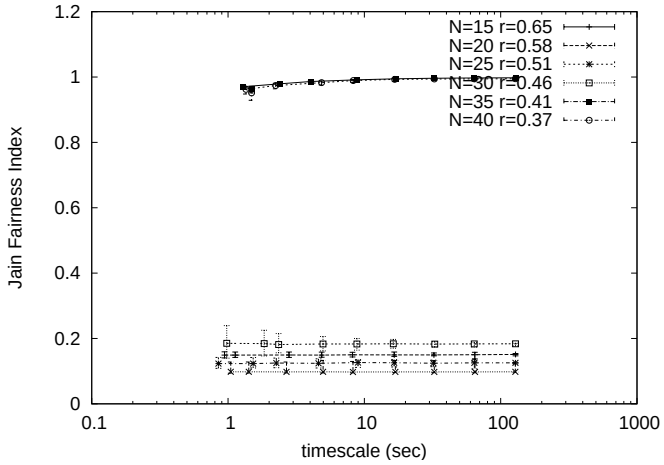
(b) n=4



(c) n=24

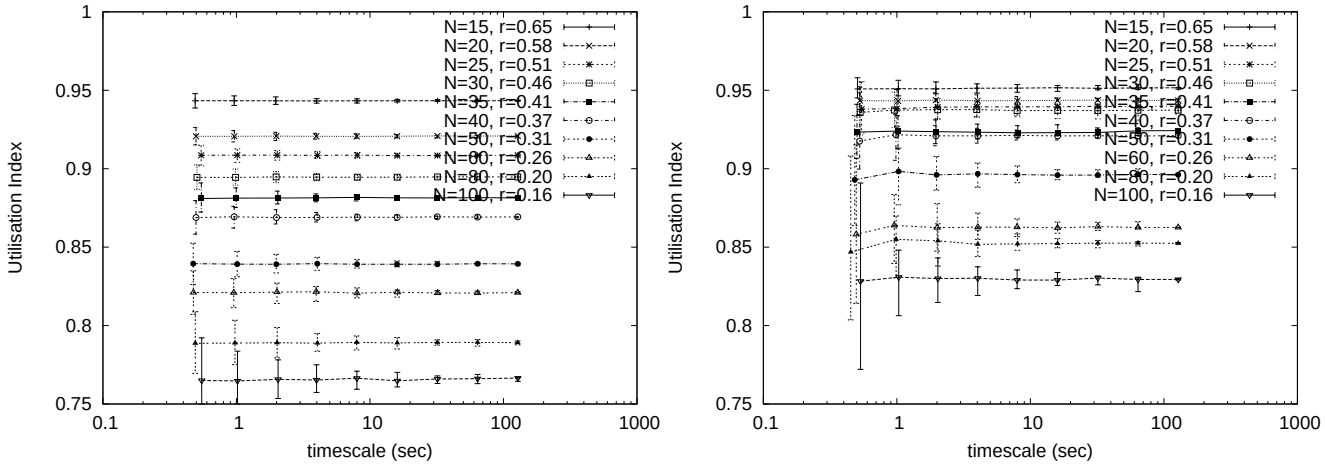


(d) n=50



(e) n=100

Figure 37: Effect of N on fairness with m = 1.07



(a) n=2

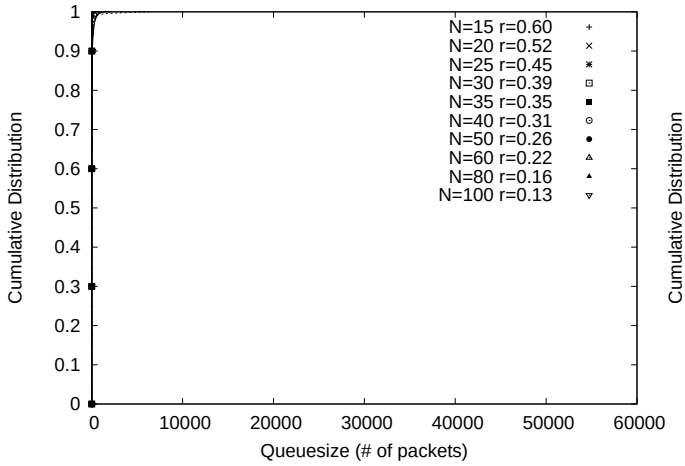
(b) n=4

(c) n=24

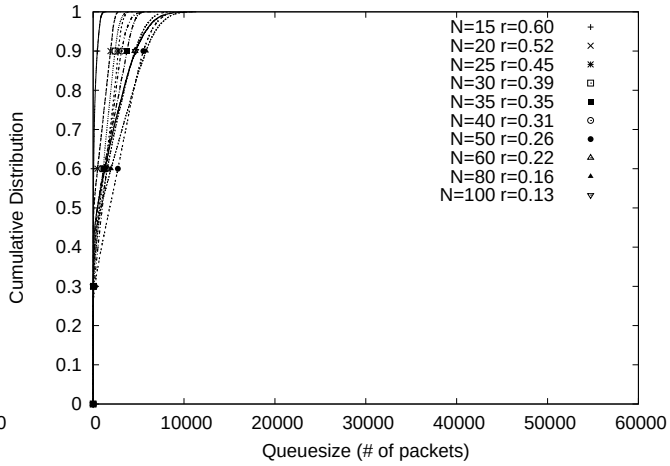
(d) n=50

(e) n=100

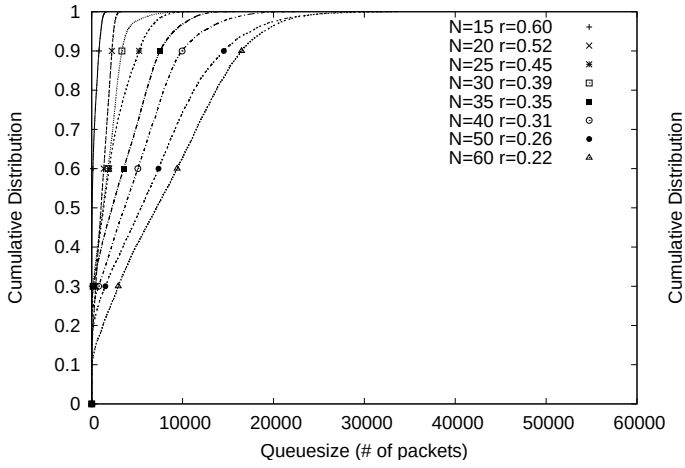
Figure 38: Effect of N on utilisation with m = 1.07



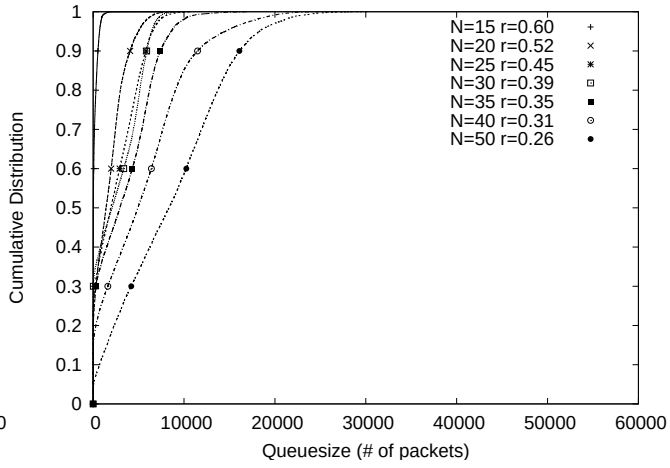
(a) n=2



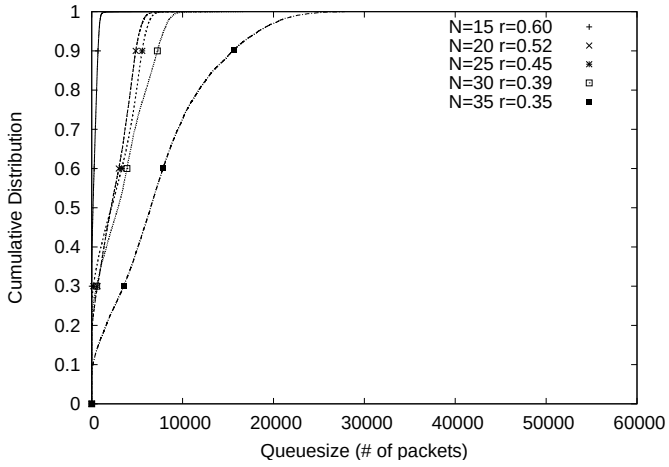
(b) n=4



(c) n=24

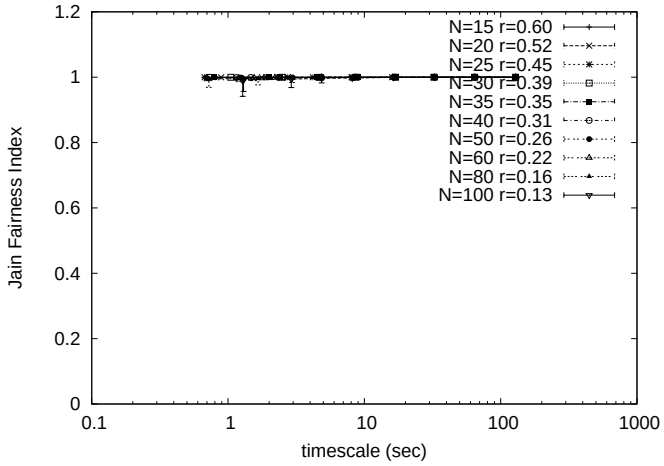


(d) n=50

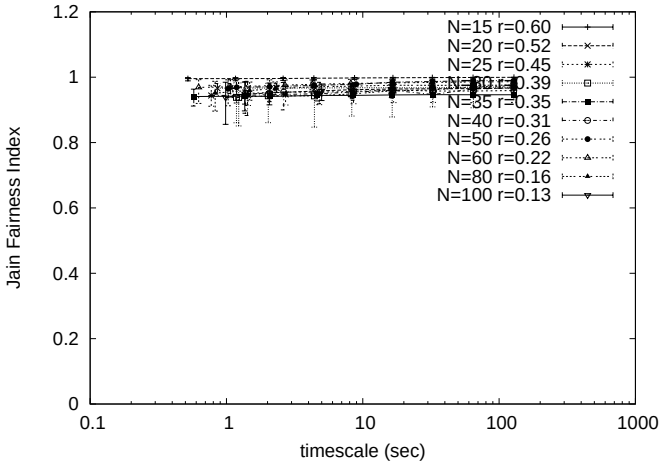


(e) n=100

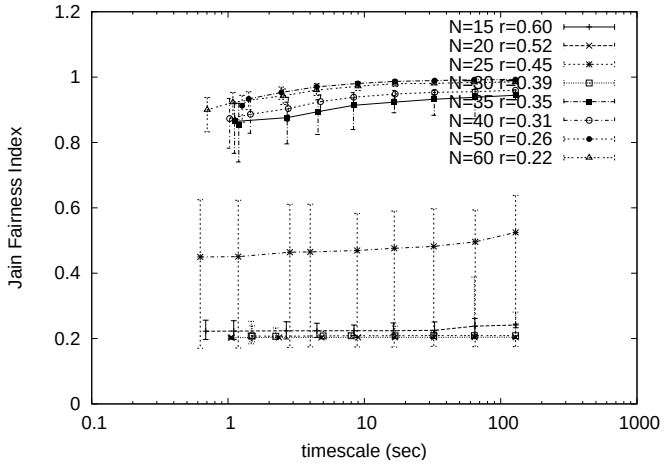
Figure 39: Effect of N on queuesizes with m = 1.09



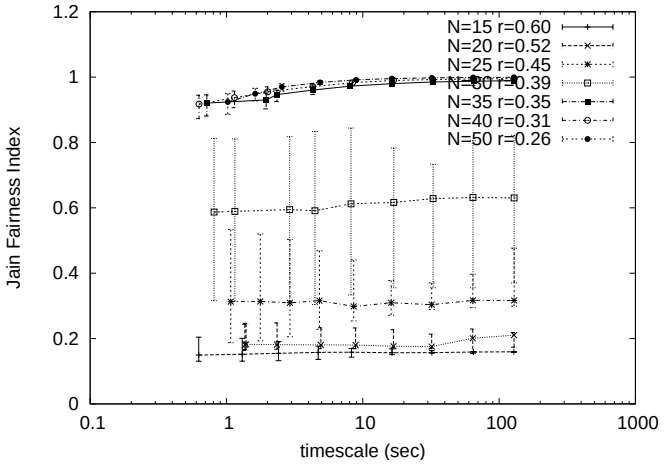
(a) n=2



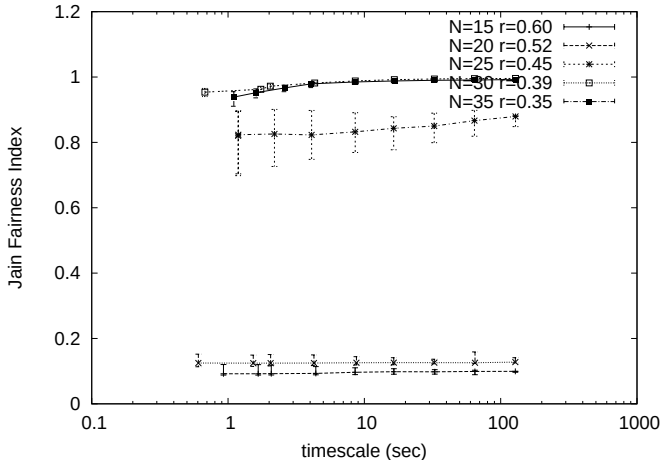
(b) n=4



(c) n=24

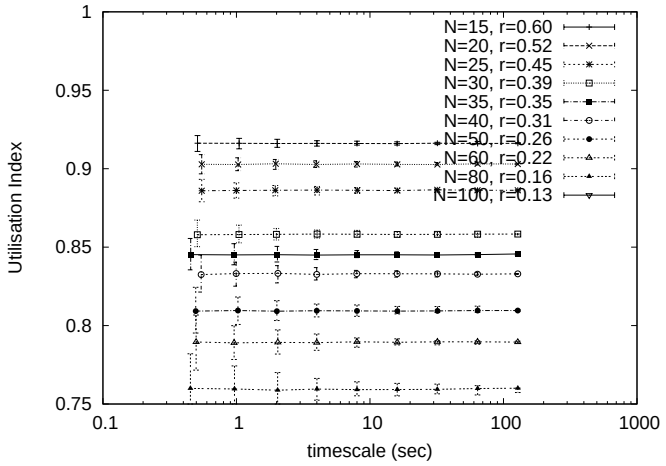


(d) n=50

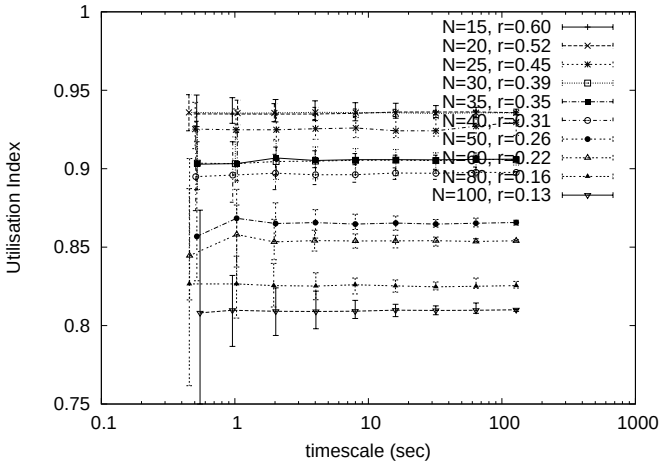


(e) n=100

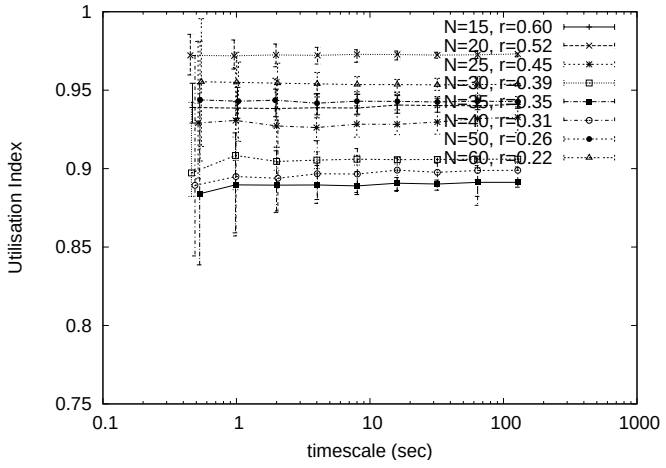
Figure 40: Effect of N on fairness m = 1.05



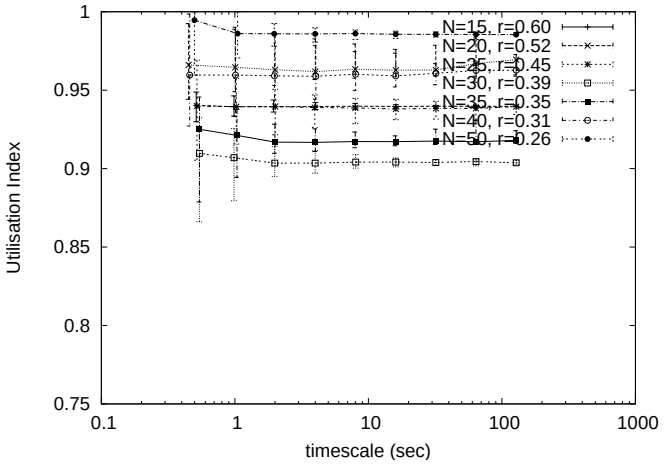
(a) n=2



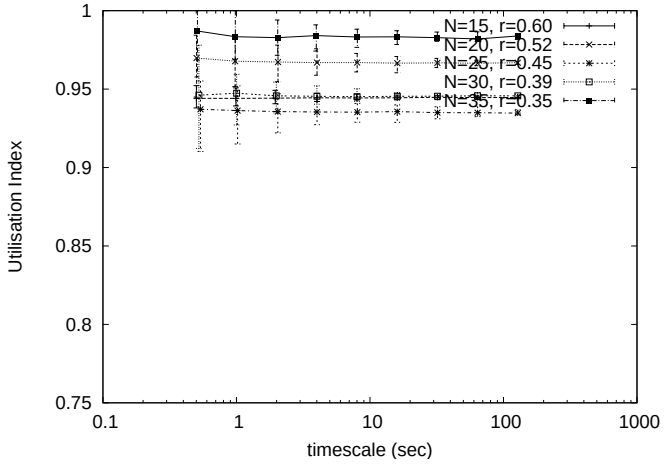
(b) n=4



(c) n=24

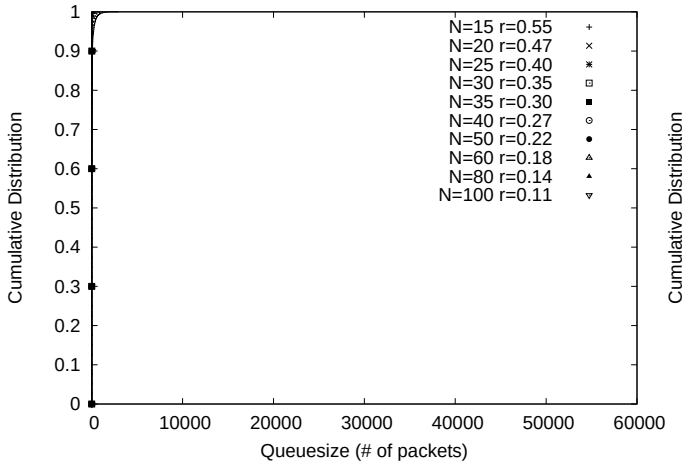


(d) n=50

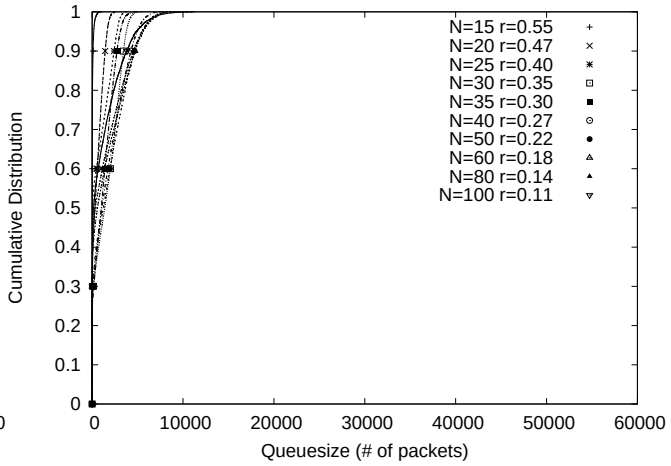


(e) n=100

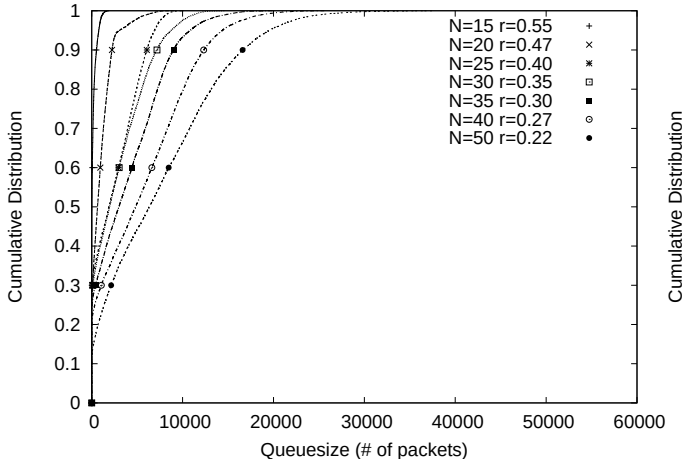
Figure 41: Effect of N on utilisation m = 1.05



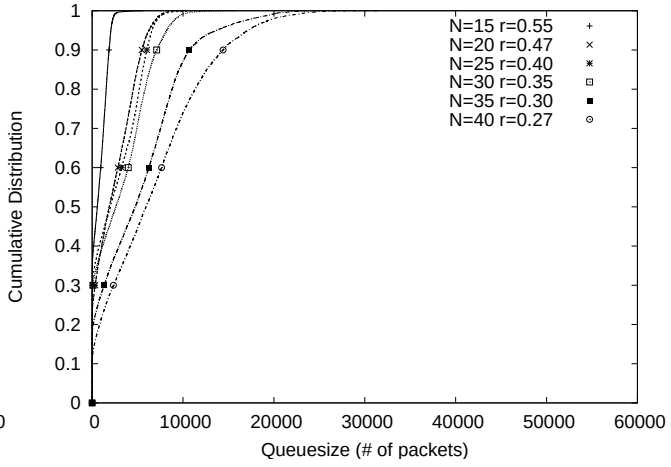
(a) n=2



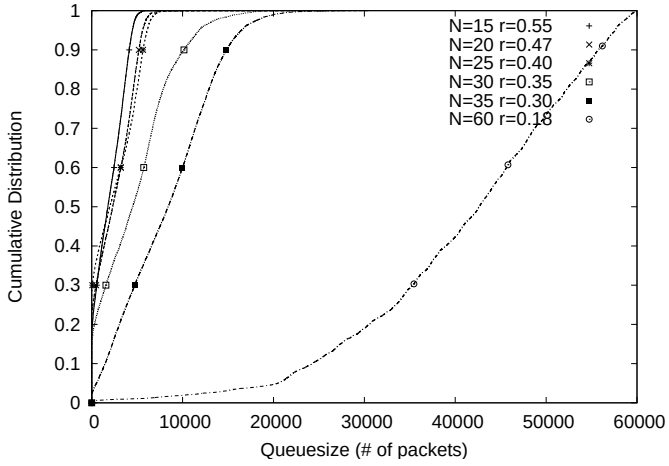
(b) n=4



(c) n=24

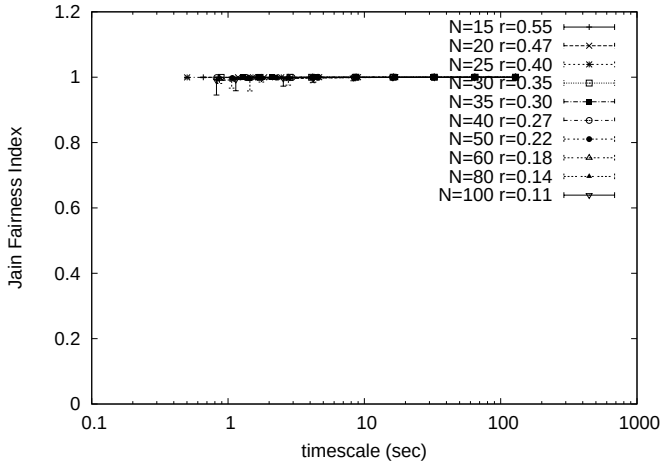


(d) n=50

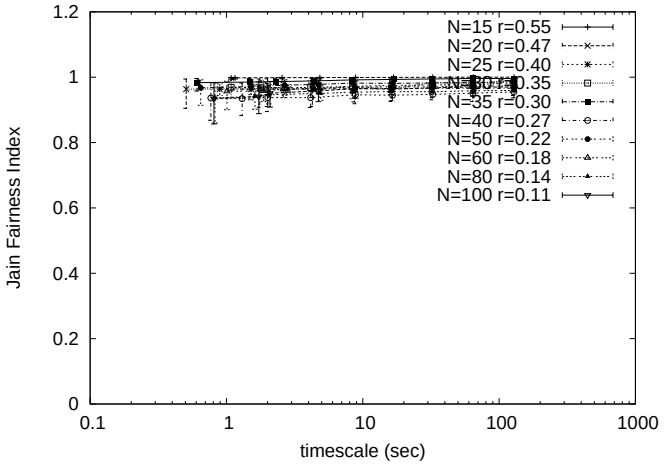


(e) n=100

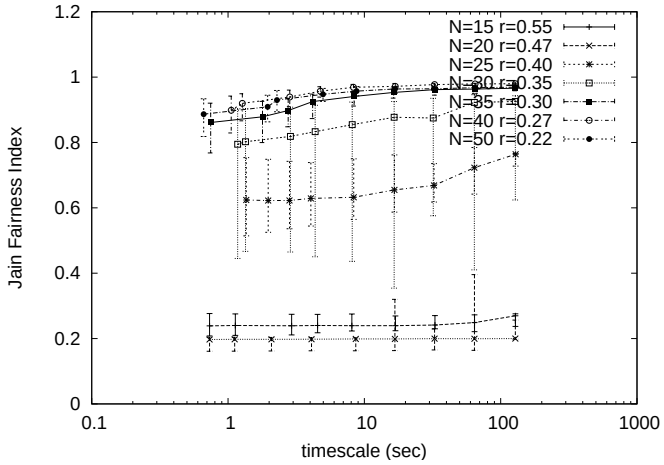
Figure 42: Effect of N on queuesizes with m = 1.11



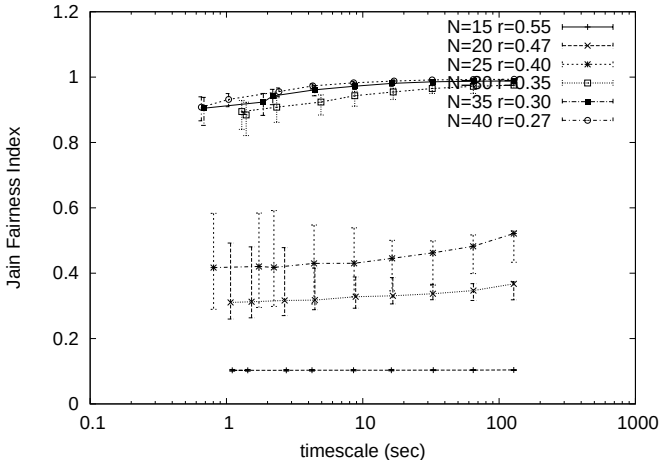
(a) n=2



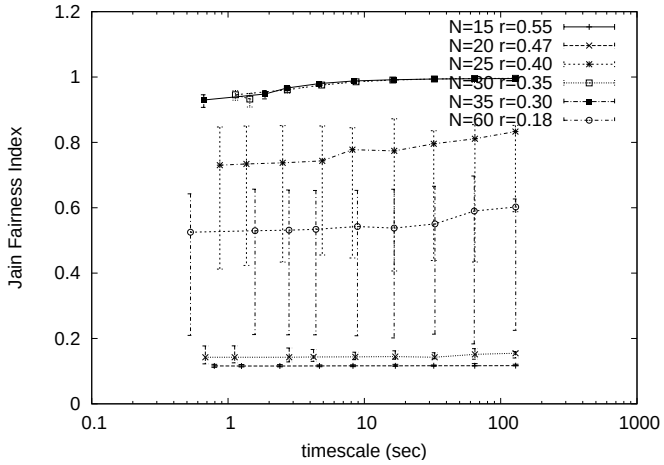
(b) n=4



(c) n=24

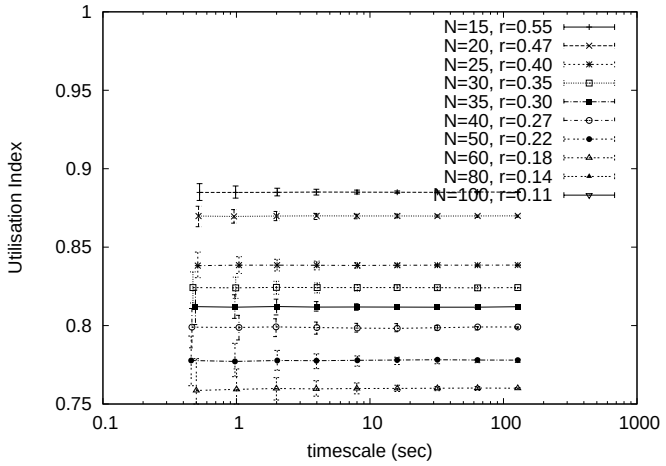


(d) n=50

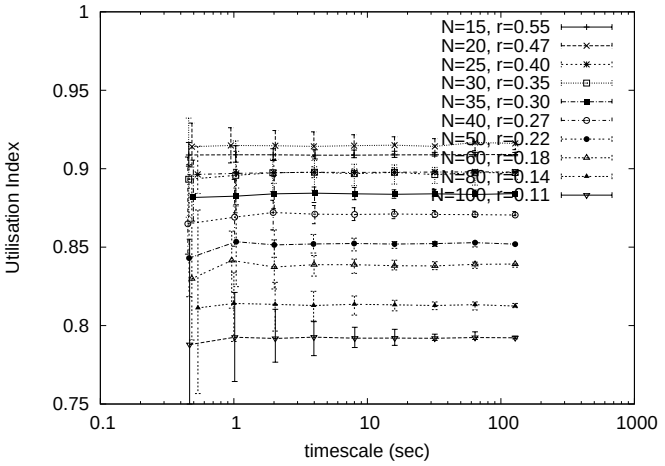


(e) n=100

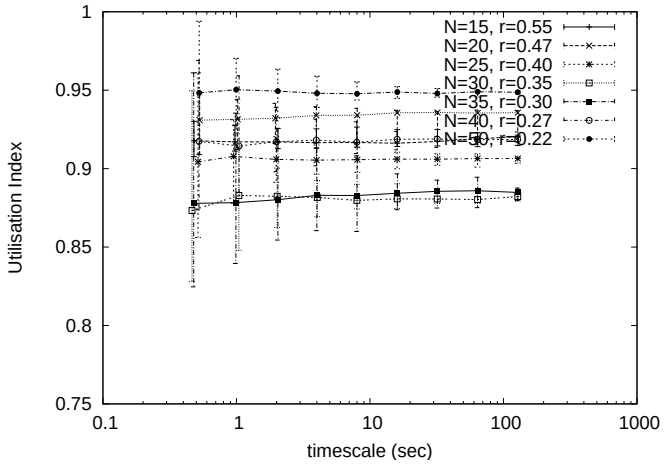
Figure 43: Effect of N on fairness with m = 1.11



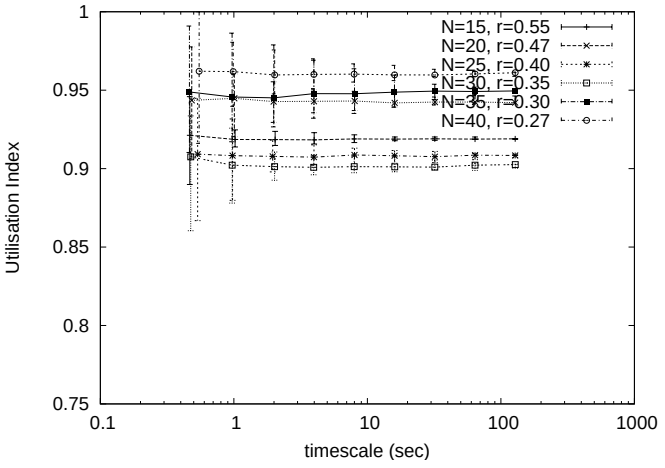
(a) n=2



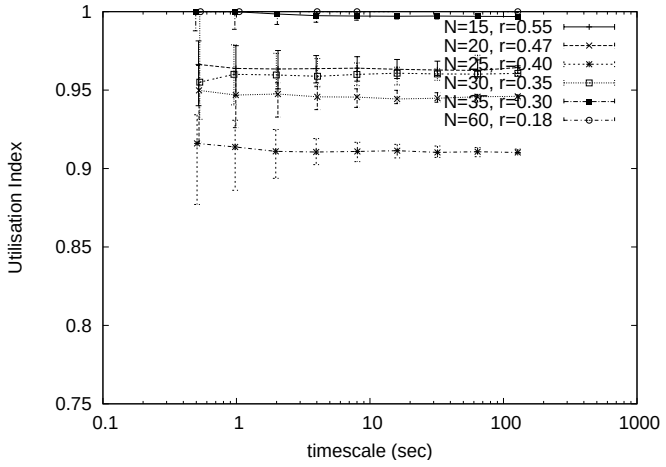
(b) n=4



(c) n=24

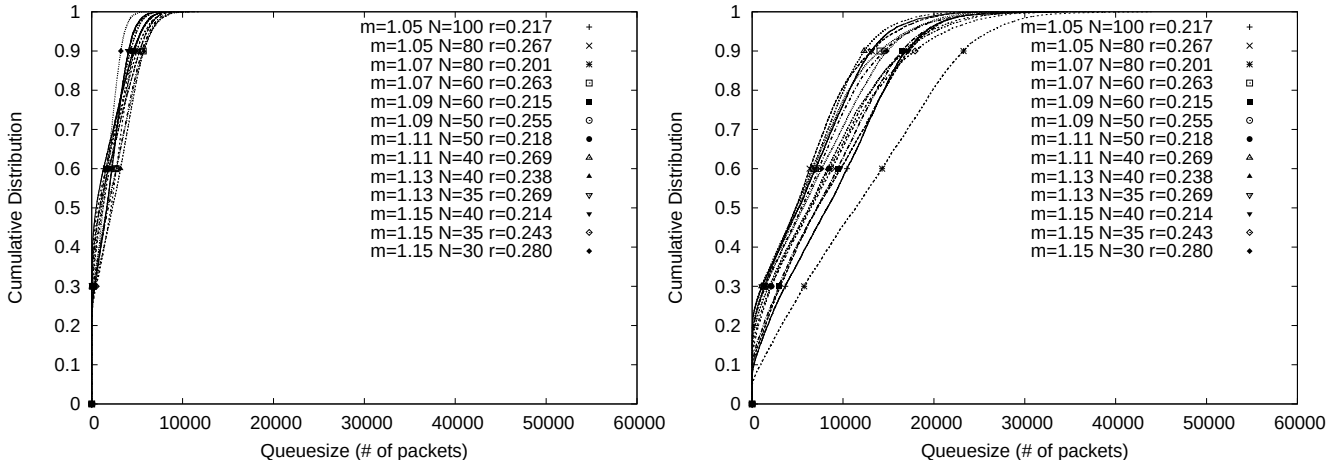


(d) n=50



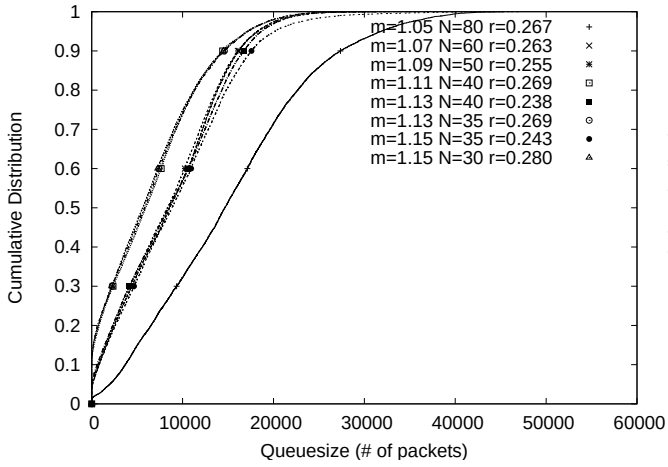
(e) n=100

Figure 44: Effect of N on utilisation with m = 1.11

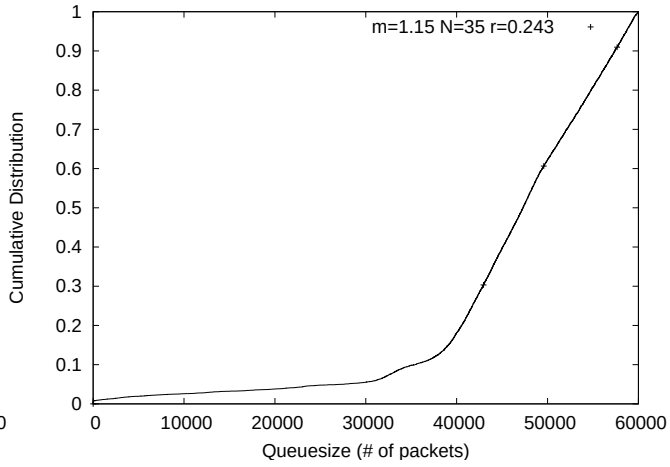


(a) n=4

(b) n=24

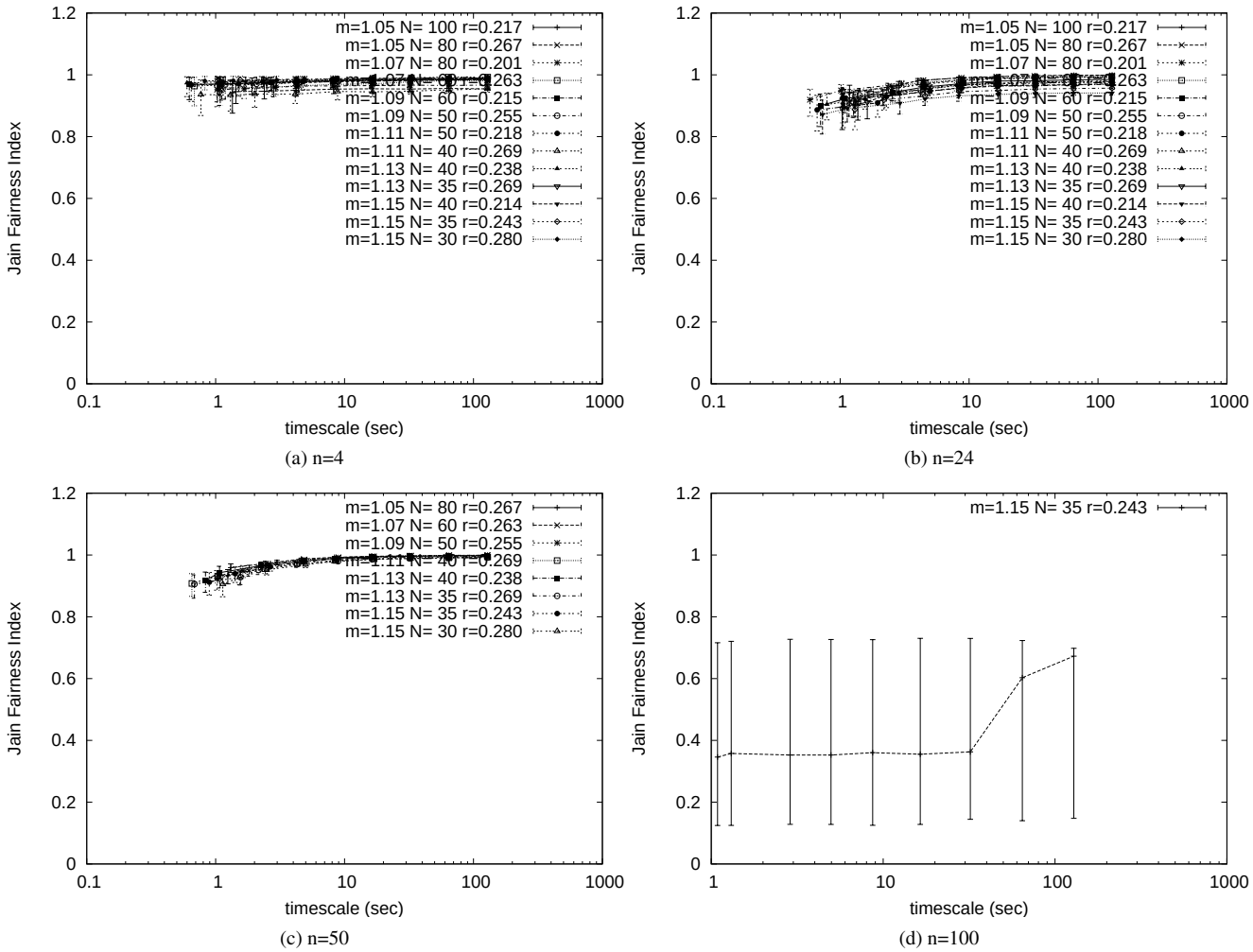


(c) n=50



(d) n=100

Figure 45: Effect of N on queuesizes with $r = 0.2$



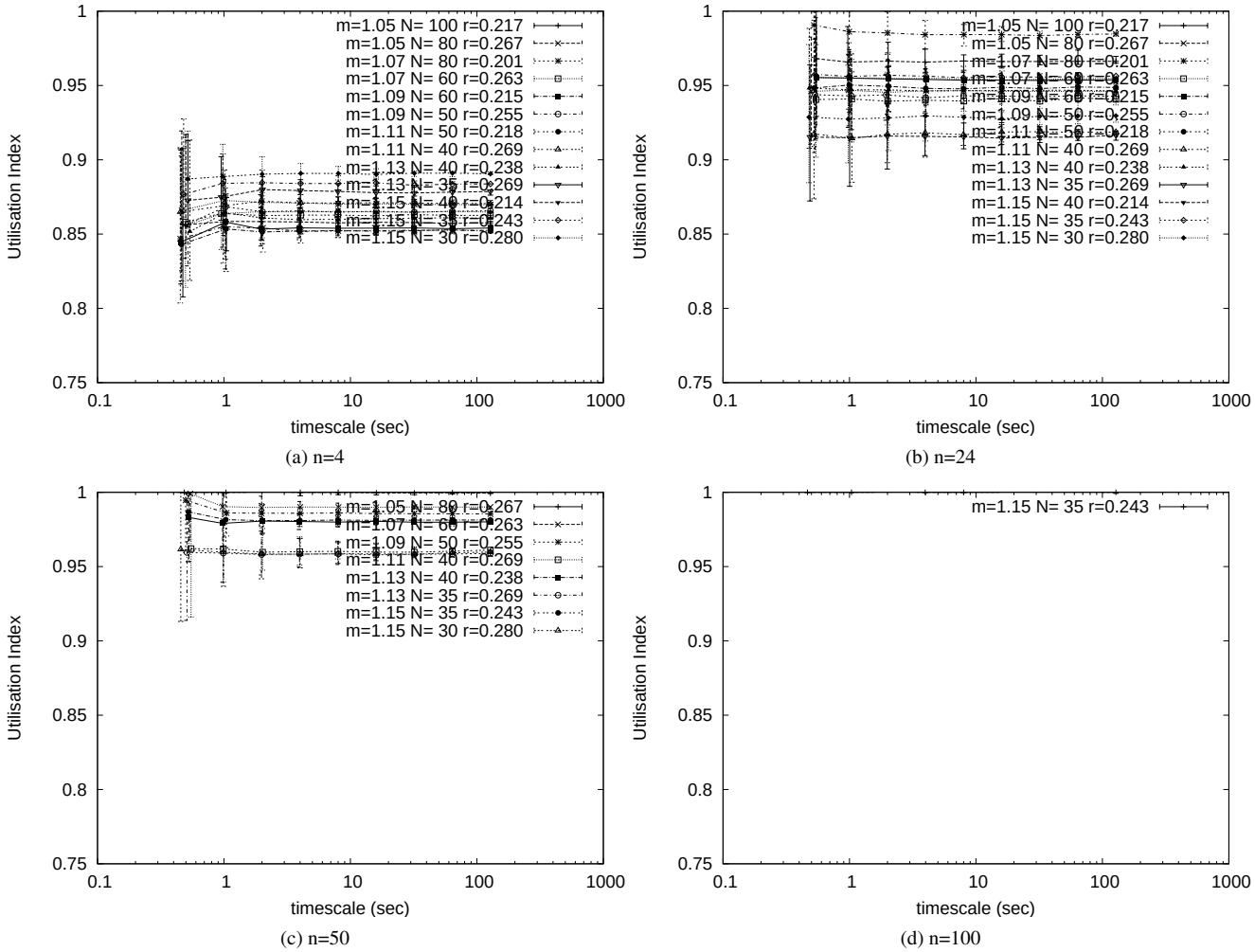
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 46: Effect of N on fairness with $r = 0.2$



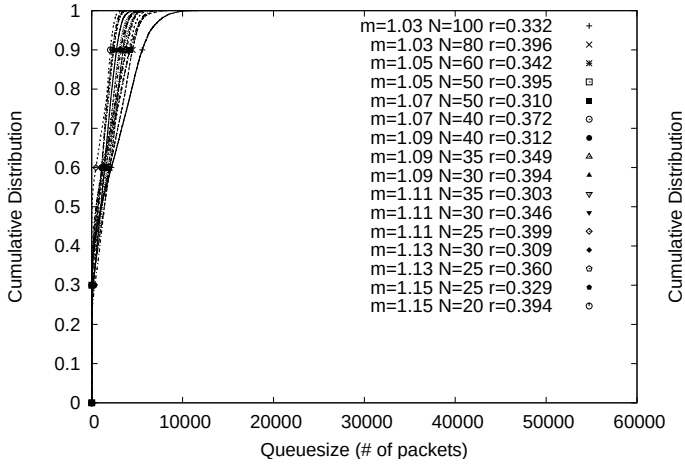
(a) n=4

(b) n=24

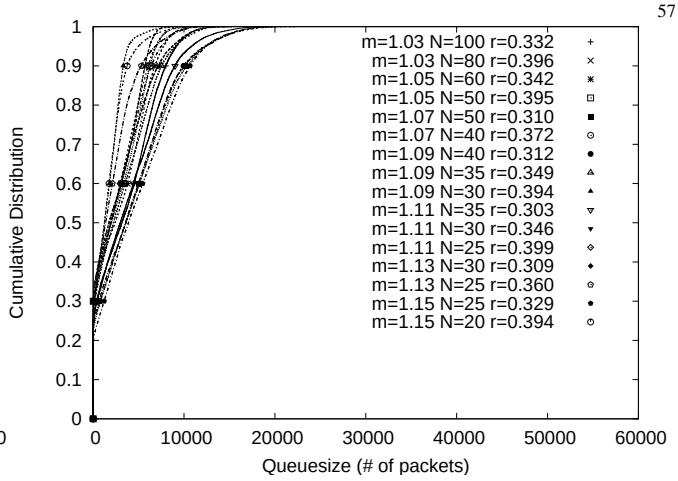
(c) n=50

(d) n=100

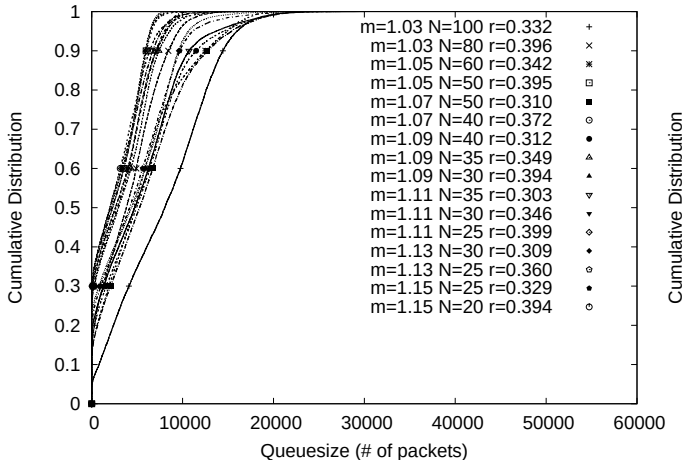
Figure 47: Effect of N on utilisation with $r = 0.2$



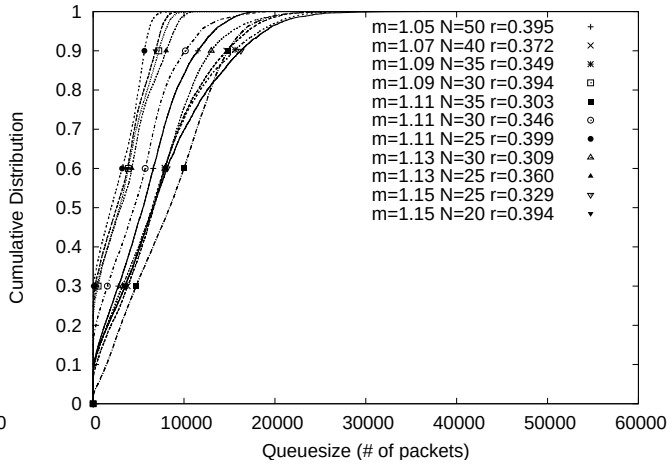
(a) n=4



(b) n=24

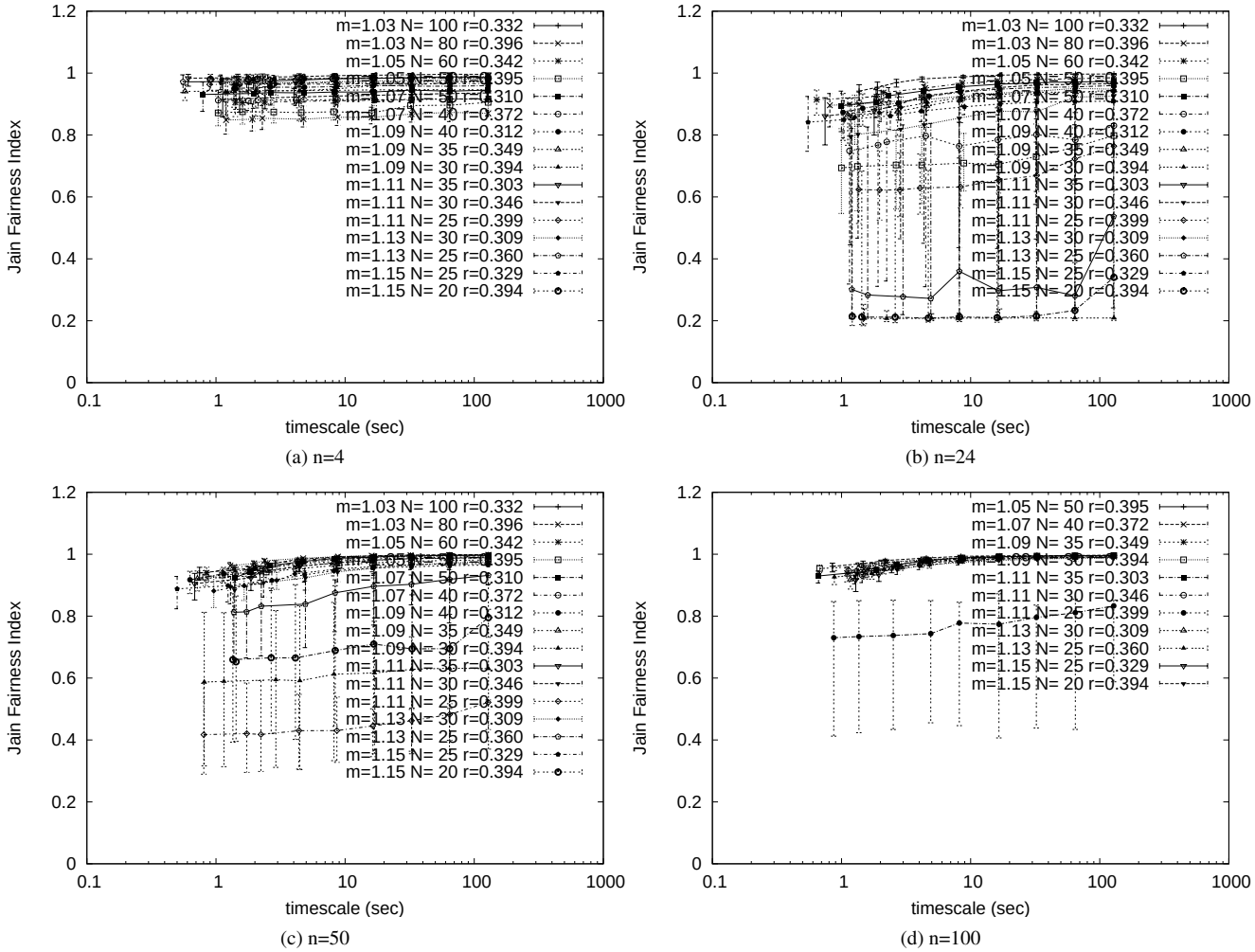


(c) n=50



(d) n=100

Figure 48: Effect of N on queuesizes with $r = 0.3$



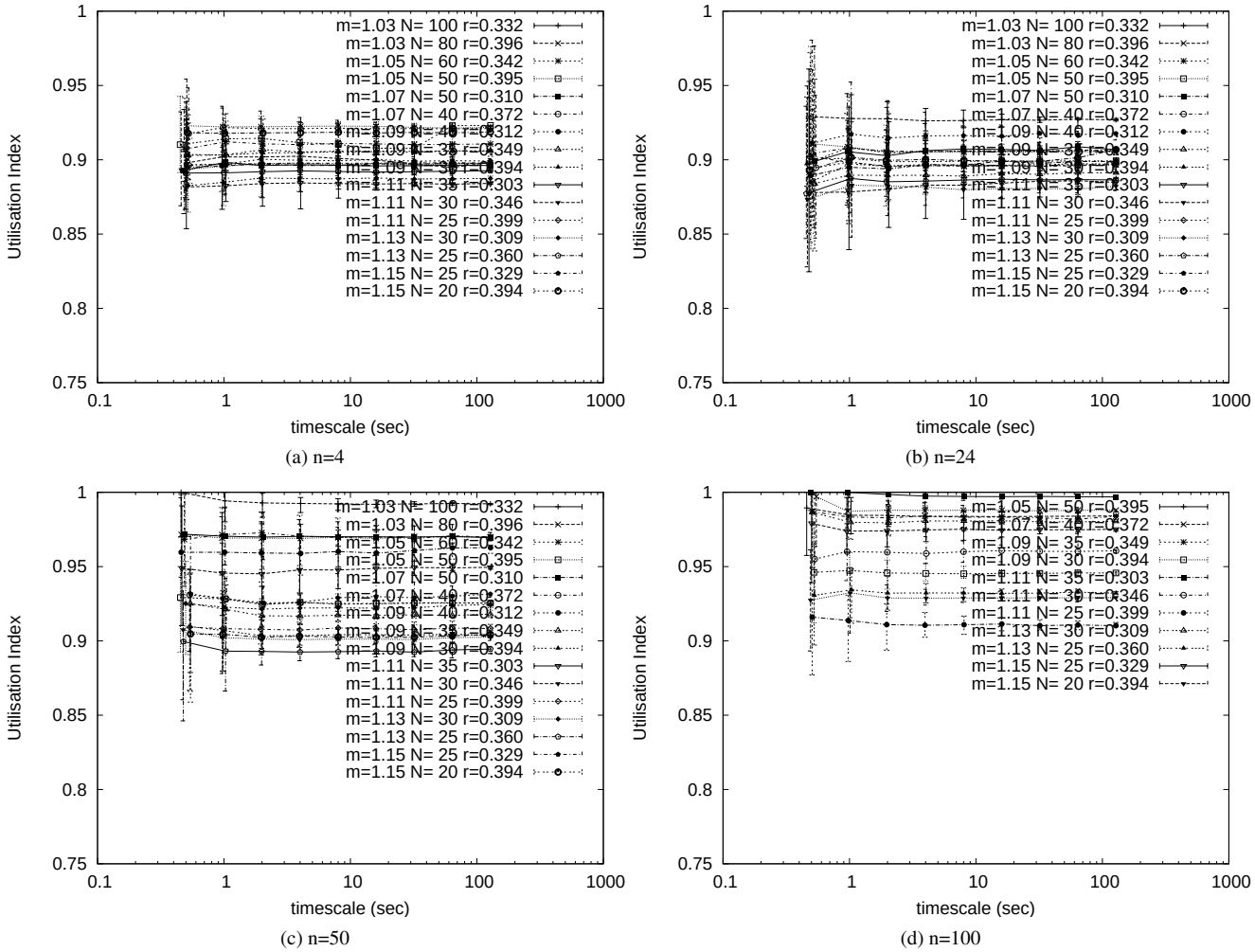
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 49: Effect of N on fairness with $r = 0.3$



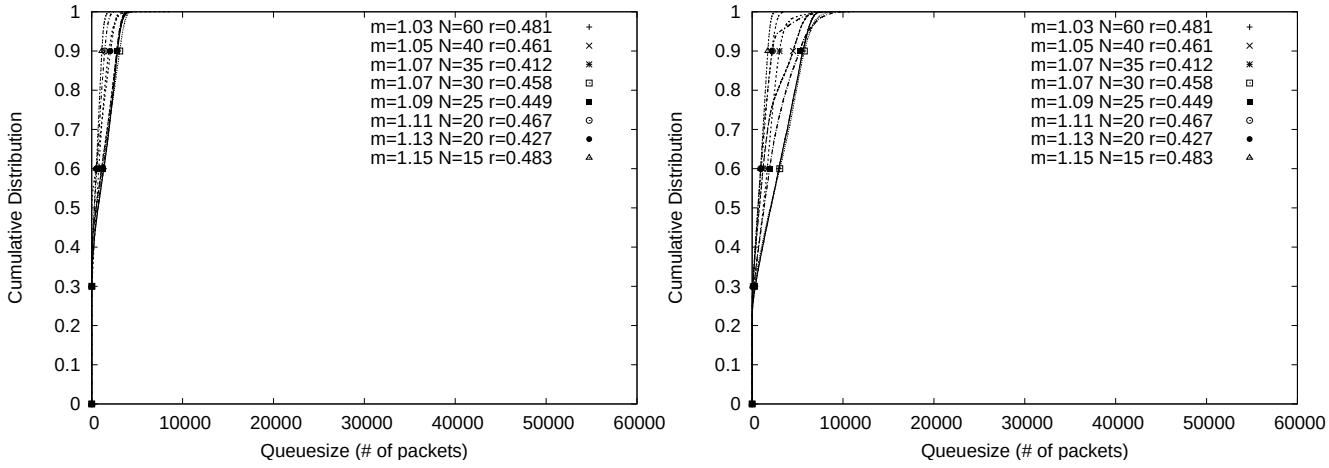
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 50: Effect of N on utilisation with $r = 0.3$



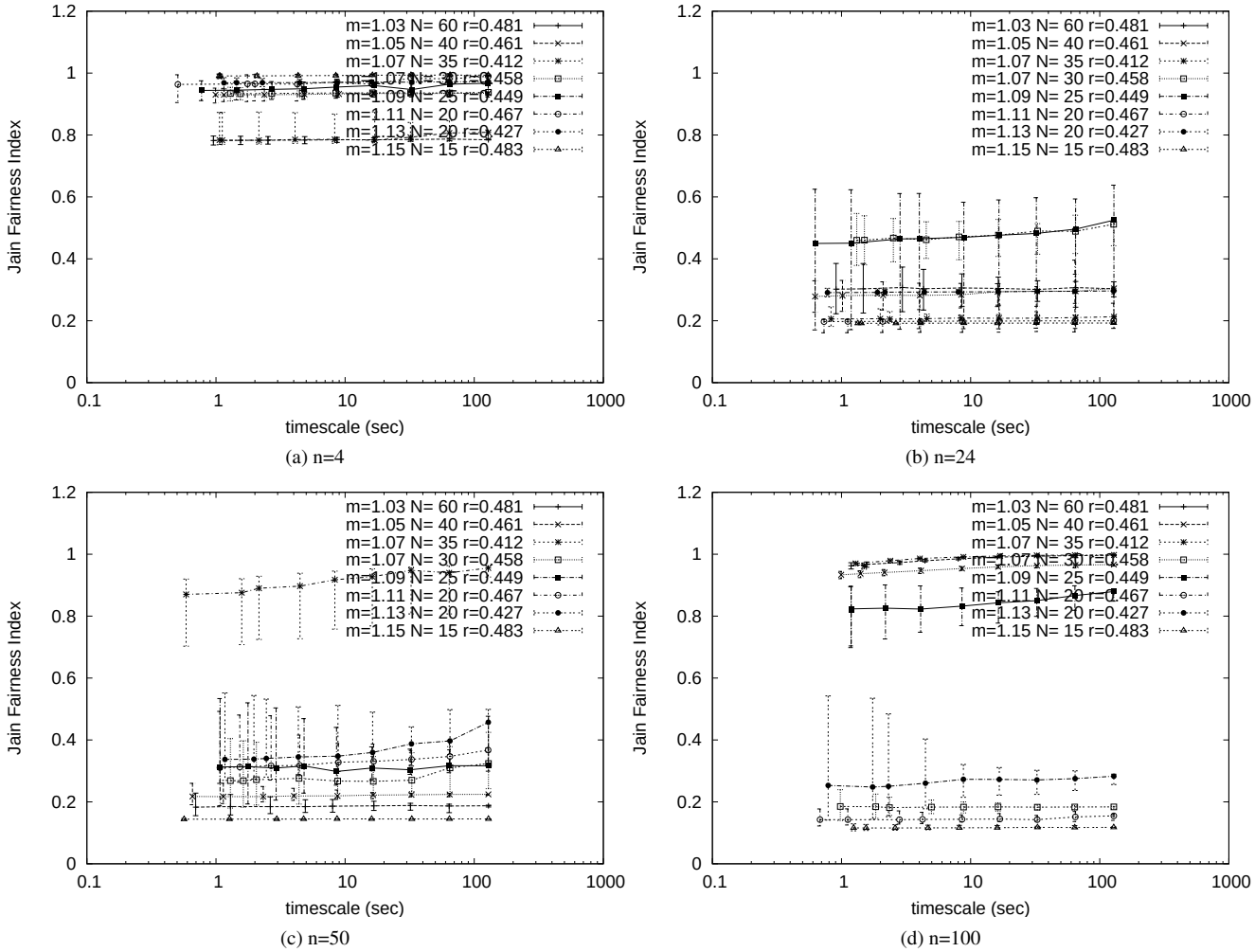
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 51: Effect of N on queuesizes with $r = 0.4$



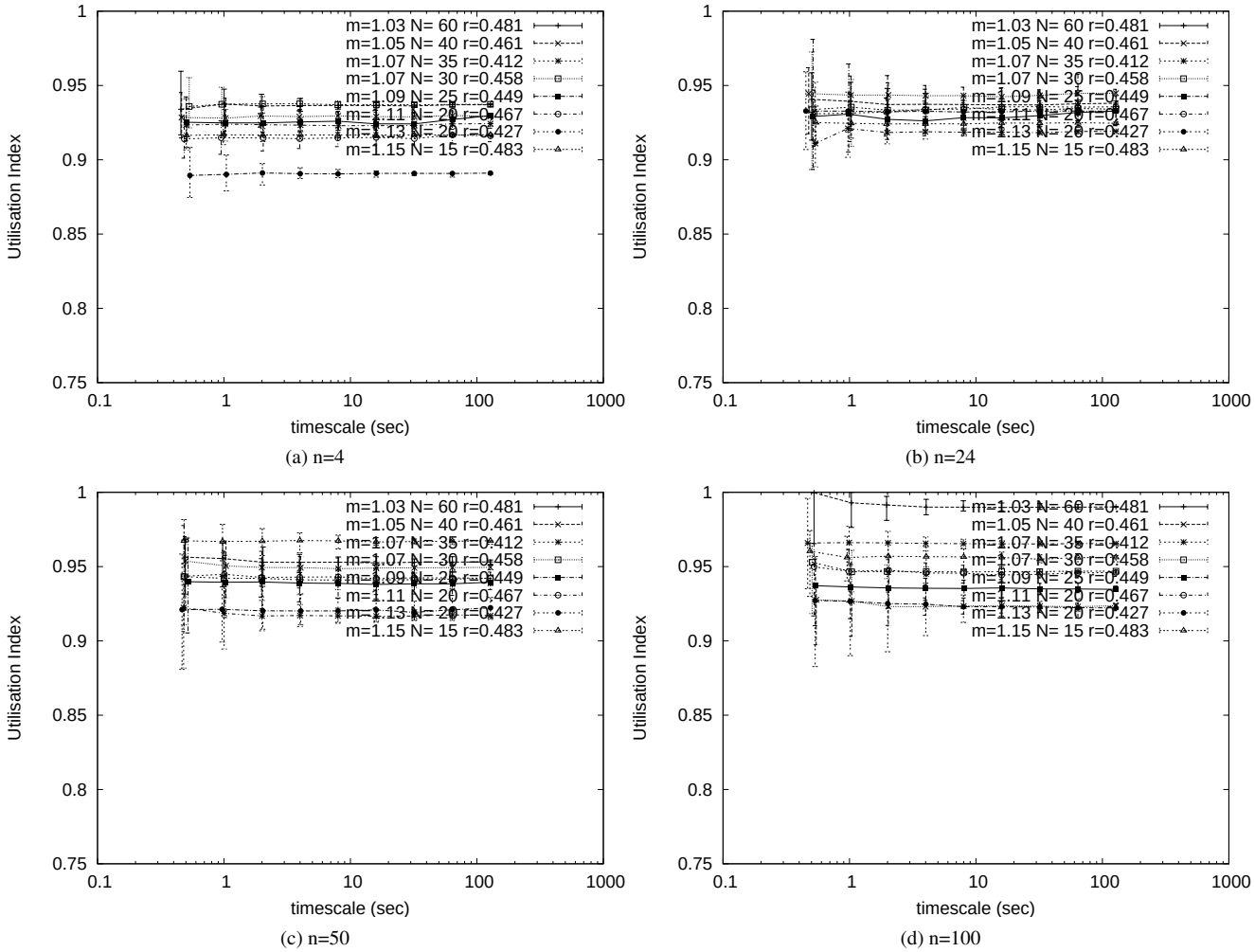
(a) n=4

(b) n=24

(c) n=50

(d) n=100

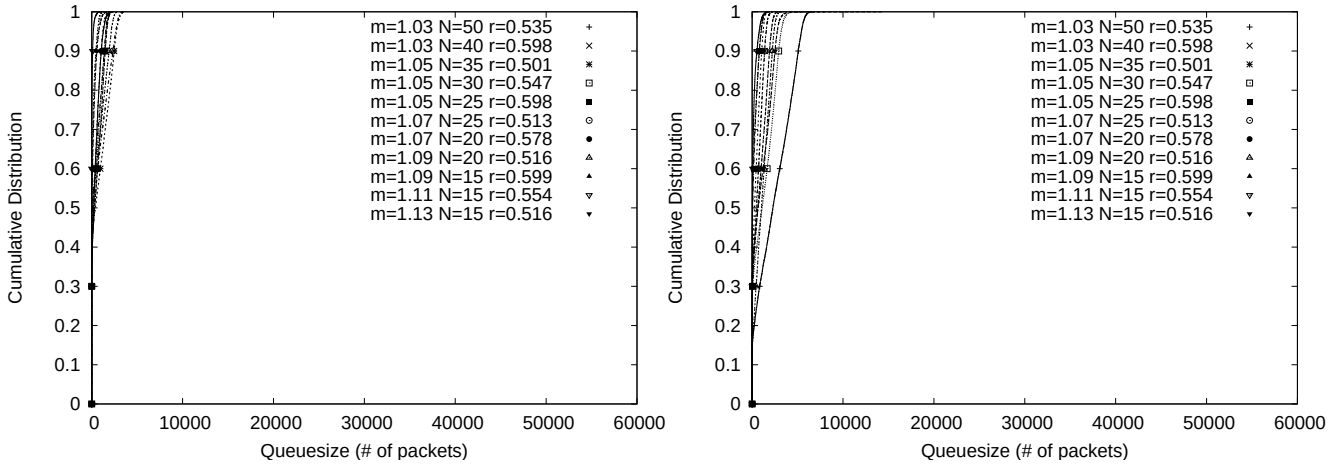
Figure 52: Effect of N on fairness with $r = 0.4$



(a) n=4 (b) n=24

(c) n=50 (d) n=100

Figure 53: Effect of N on utilisation with $r = 0.4$



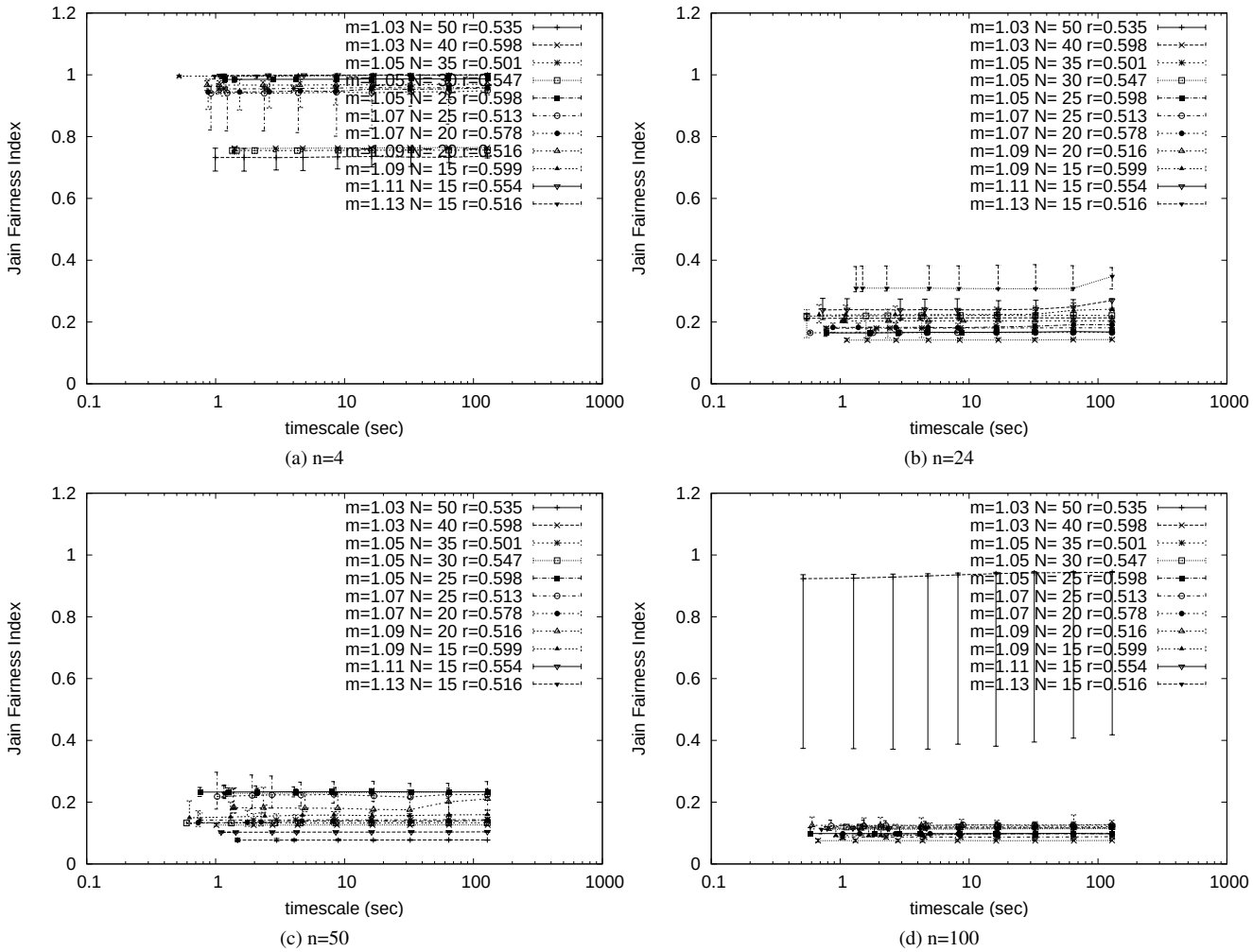
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 54: Effect of N on queuesizes with $r = 0.5$



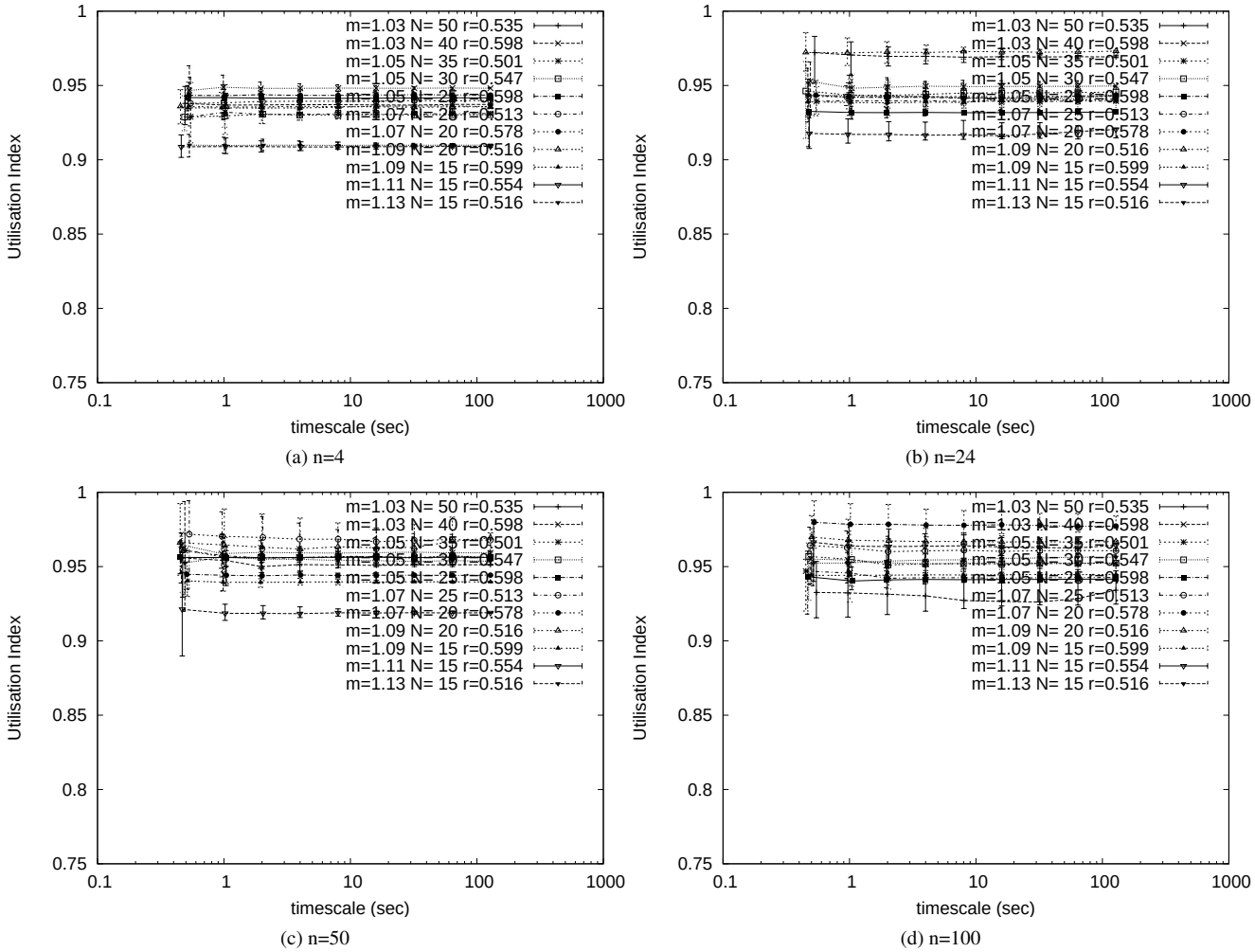
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 55: Effect of N on fairness with $r = 0.5$



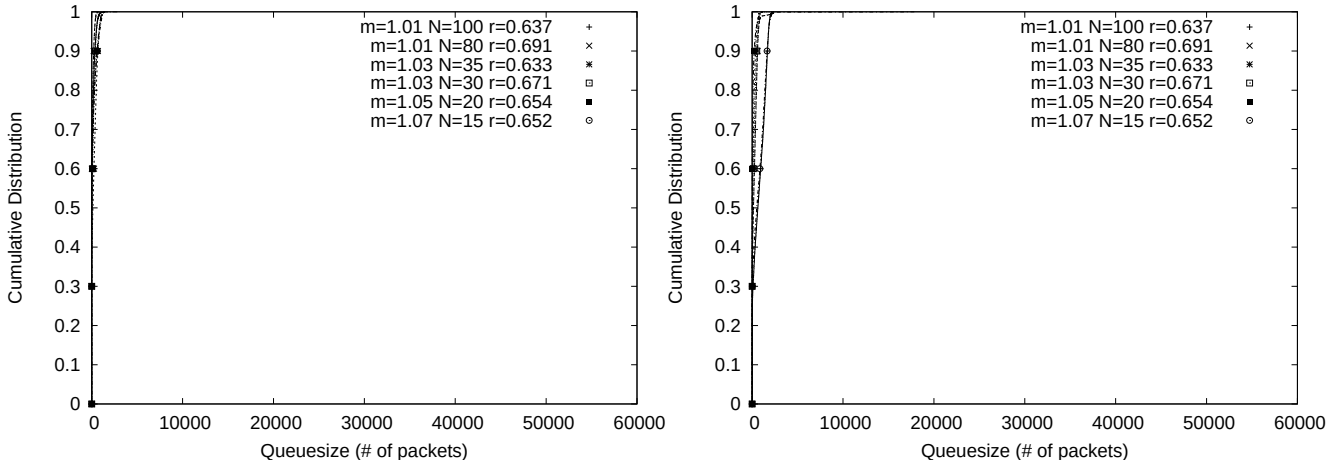
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 56: Effect of N on utilisation with $r = 0.5$



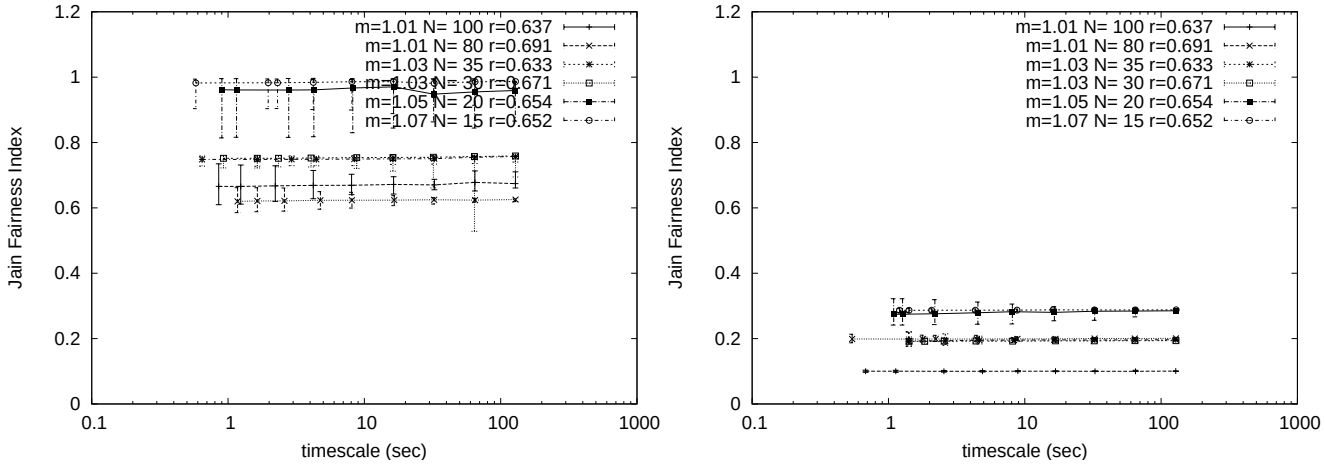
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 57: Effect of N on queuesizes with $r = 0.6$



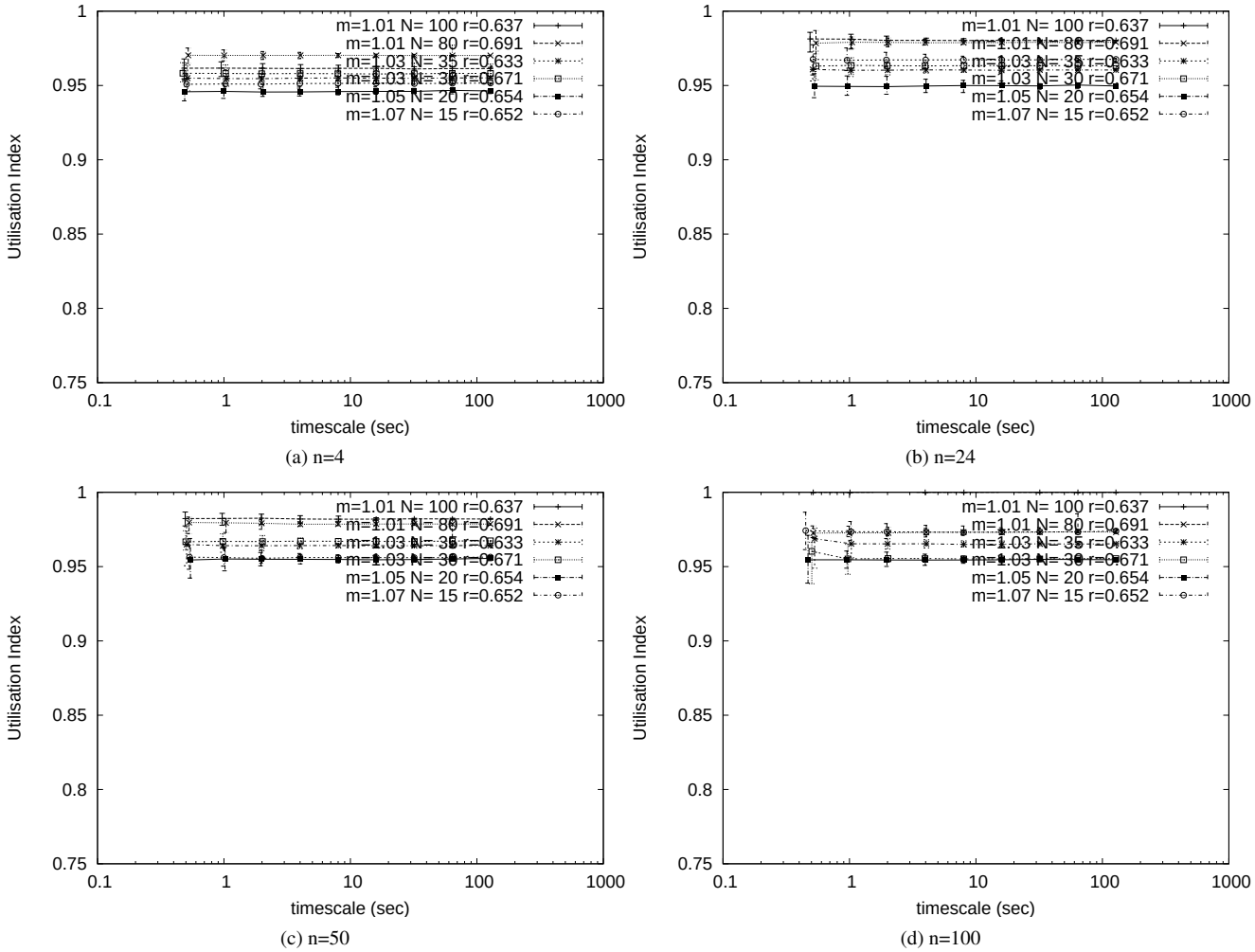
(a) n=4

(b) n=24

(c) n=50

(d) n=100

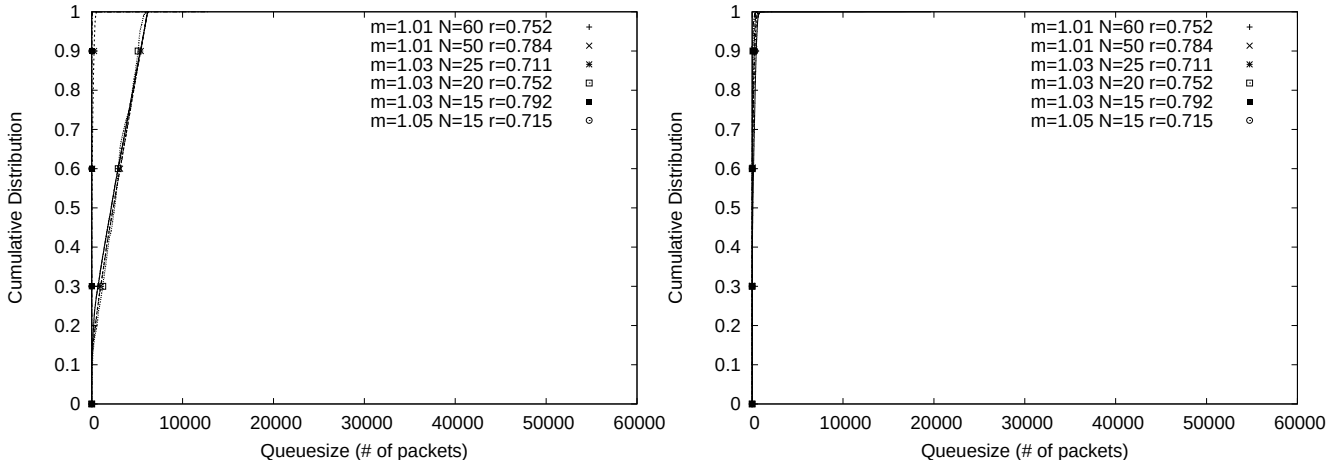
Figure 58: Effect of N on fairness with $r = 0.6$



(a) n=4 (b) n=24

(c) n=50 (d) n=100

Figure 59: Effect of N on utilisation with $r = 0.6$



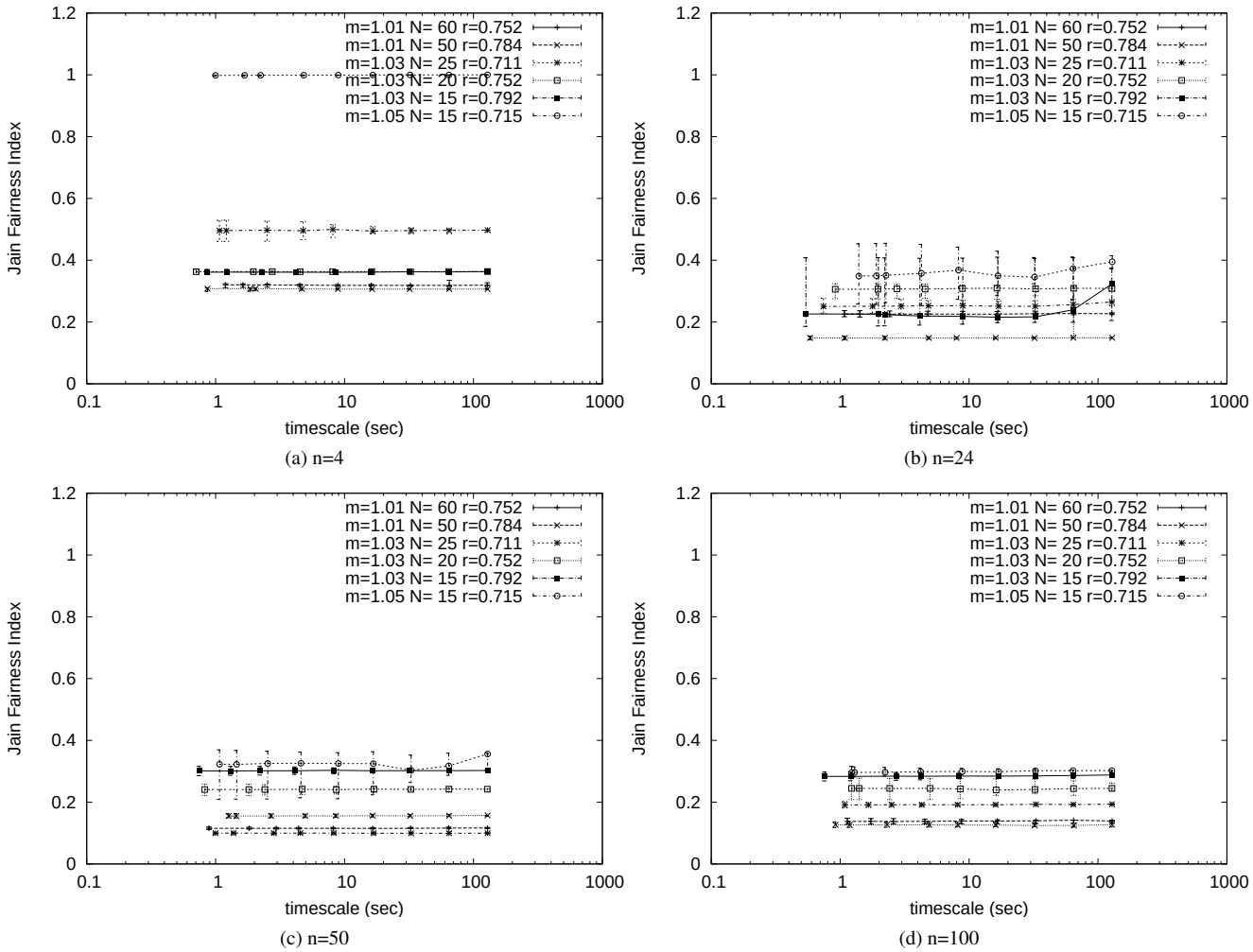
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 60: Effect of N on queuesizes with $r = 0.7$



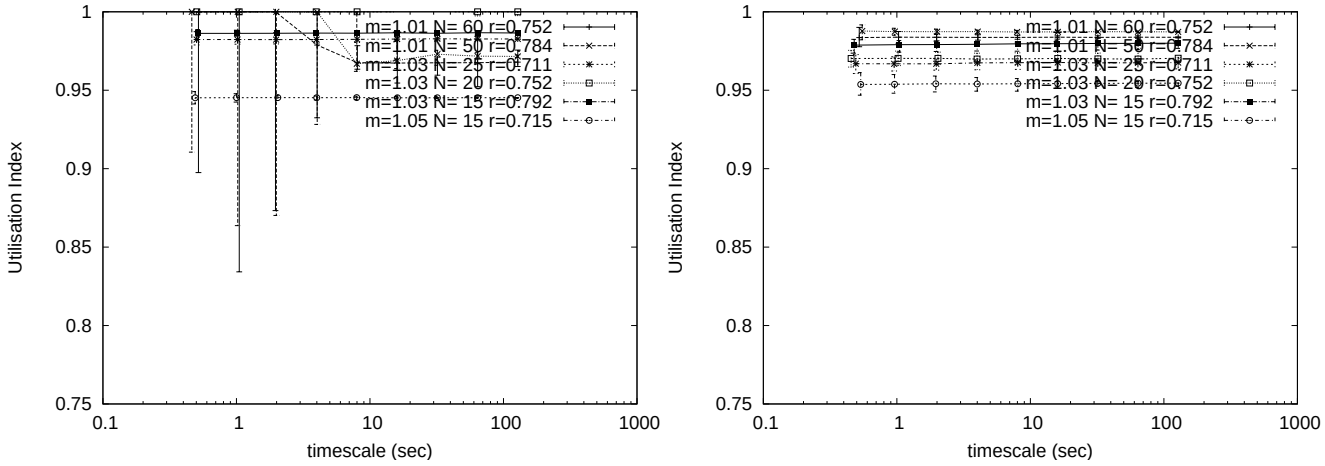
(a) n=4

(b) n=24

(c) n=50

(d) n=100

Figure 61: Effect of N on fairness with $r = 0.7$



(a) n=4

(b) n=24

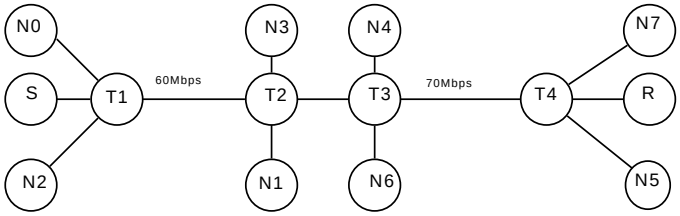
(c) n=50

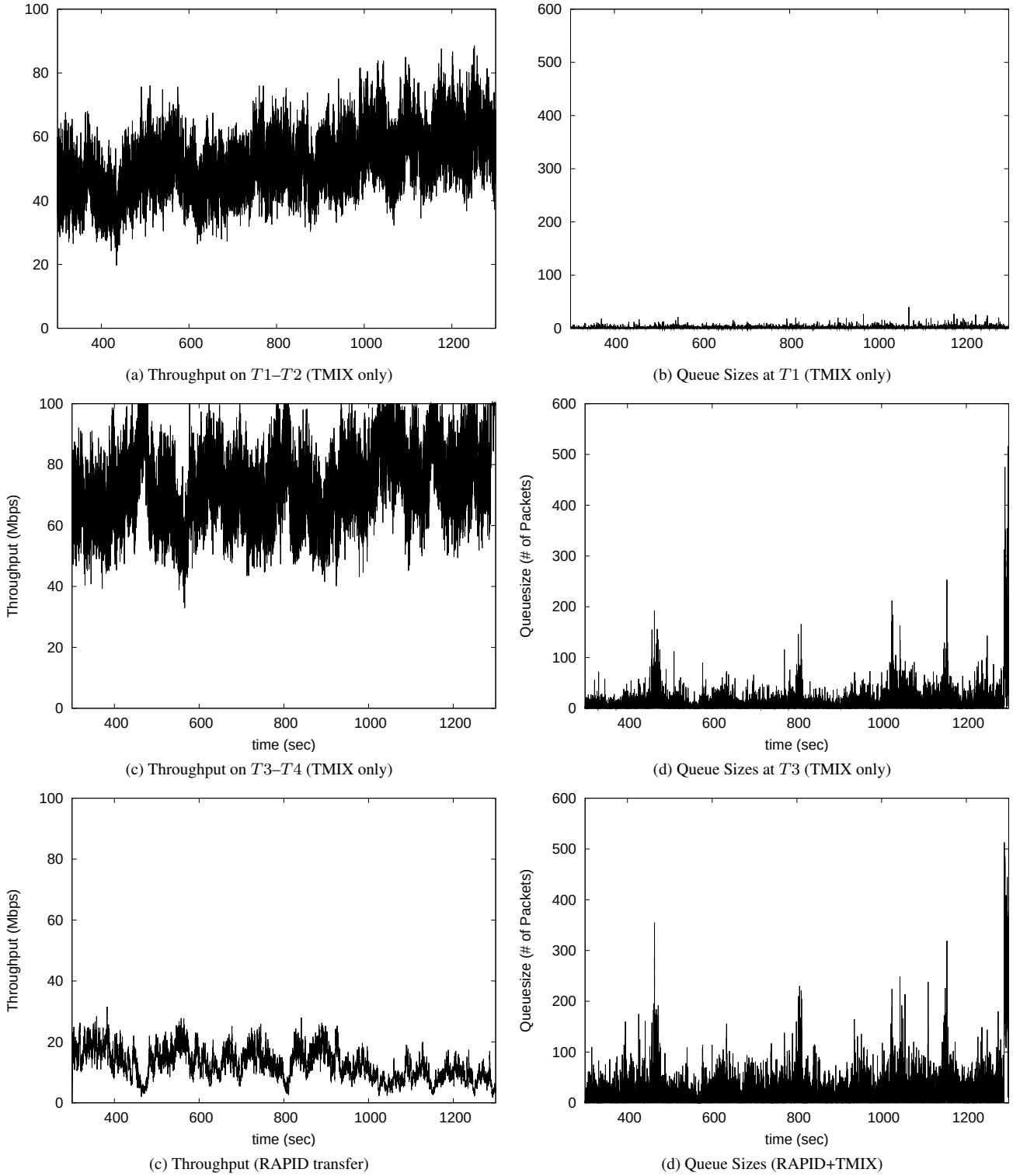
(d) n=100

Figure 62: Effect of N on Utilisation with $r = 0.7$

C Multihop

This section gives details about the performance of RAPID in multihop settings. The topology for this simulation is shown in fig C. $T1$, $T2$, $T3$, $T4$ are the TMIX Delay Boxes. $N0$ and $N1$ are the two initiator nodes for the link $T1$–$T2$. $N4$ and $N5$ are the two initiator nodes for the link $T3$–$T4$. $N2$, $N3$ and $N6$, $N7$ are the corresponding acceptor nodes. We use TMIX to generate TCP traffic at average offered loads of 60Mbps and 70Mbps for 30 minutes and drive it through links $T1$–$T2$ and $T3$–$T4$ respectively. A rapid connection runs between nodes S and R. So, the bottle neck link is $T3$–$T4$. Fig 63(a) plots the throughput time series on the link $T1$–$T2$ without rapid transfer. Fig 63(c) plots the throughput time series on the link $T3$–$T4$ without rapid transfer. Fig 63(e) plots the throughput of the rapid transfer with time. We can observe that the available bandwidth on each of the links $T1$–$T2$ and $T3$–$T4$ is highly variable. But, on an average the second link $T3$-$T4$ is the bottle neck link and rapid adjusts its throughput to get the best of the available bandwidth with minimum queueing overhead. Figs 63(b), 63(d) and 63(f) plot the time series of queues. We can see that rapid connection do not cause much increase in queueing.





(a) Throughput on $T1$–$T2$ (TMIX only)

(b) Queue Sizes at $T1$ (TMIX only)

(c) Throughput on $T3$–$T4$ (TMIX only)

(d) Queue Sizes at $T3$ (TMIX only)

(c) Throughput (RAPID transfer)

(d) Queue Sizes (RAPID+TMIX)

Figure 63: Statistics of TMIX Multihop Transfers