Modeling and Analysis to Estimate the End-System Performance Bottleneck for High-Speed Data Transfer

Amitabha Banerjee, Biswanath Mukherjee, and Dipak Ghosal
Department of Computer Science, University of California, Davis, CA 95616, USA
abanerjee@ucdavis.edu, mukherje@cs.ucdavis.edu, ghosal@cs.ucdavis.edu

***Abstract*— The bandwidth available in a high-speed backbone network today is much greater than the capacity at the edge. As an example, when the transmission line rate is 10 Gbps, the end computing machine (end-system) may not be able to keep up with the arriving data rate and is hence often a bottleneck for data transfer. A key deficiency in most transport protocols is the lack of an accurate estimate of this bottleneck rate. In the absence of the same, the sender often overshoots the receiver's capacity resulting in packet losses. When the sender receives feedback about the packet losses, it throttles the sending rate. Repeat occurrences of the same limits the performance of end-to-end data transfer.**

We propose a mechanism to determine the end-system bottleneck rate. We employ a Stochastic Reward Net (SRN) queuing network model which captures different system components such as the processor, disk, etc., as well as the current workload in terms of the number of CPU-bound and I/O-bound tasks. The sending rate which would yield the best performance for data transfer to the end-system is estimated from the above model. We define this rate as the *effective bottleneck rate*. Finally, we validate our analytical model on an experimental testbed and demonstrate that matching the sending rate with the *effective bottleneck rate* leads to improved performance of end-to-end data transfer.

I. Introduction

End-systems such as end-user workstations, scientific computational systems, cluster nodes, etc. are now equipped with 10 Gbps Network Interface Cards (NICs) [6]. Also occurring is a shift from the packet-switched paradigm to the circuit-switched paradigm for large-scale high-speed data transfers. Backbone optical circuit-switched networks which provide dedicated end-to-end circuit connectivity between a network operator's Points-of-Presence (PoP) have now been deployed. Such networks are described in the literature as lambda grids. The National LambdaRail (NLR) [3] and Department of Energy (DOE)'s UltraScienceNet [2] projects in the United States, and CANARIE's CA*net in Canada [1] are a few examples. The scientific community already uses such infrastructure to establish an end-to-end circuit between two end-systems for deterministic large-scale data transfers required by scientific experiments and applications.

With an end-to-end circuit, there is no congestion in the core network (lambda grid) because dedicated end-to-end bandwidth is available. In such a setting, the congestion moves to the edge of the network, namely the end-system, which may be unable to handle the high bit-rate connections available in the lambda grid. For example, high-priority and critical tasks may pre-empt the execution of the task which is responsible for receiving data. Compute-intensive tasks such as analysis and visualization of the received data may also be executing at the end-system, in which case the operating system (OS) has to schedule a computationally (CPU)-bound task (e.g., visualization), and an interrupt-driven task (receiving data) simultaneously. In such a case, packets may get dropped due to buffer overflow if interrupts are not serviced by the OS within the appropriate time. Such losses must be minimized to improve the data transfer time. This may be achieved by transmitting data at a rate which matches the receiving end-system's ability, so that the OS can handle the arriving packets with few or no packet losses.

Our contribution in this work is to demonstrate that end-system dynamics may often be modeled and predicted to a reasonable accuracy. We develop a SRN queuing network model to capture the system components such as the processor, disk, etc. We solve this model to estimate the end-system *effective bottleneck rate* as a function of the current workload (the number of executing tasks). We demonstrate that such an end-system's rate-matched sending rate maximizes the network utilization while minimizing the losses at the end-system and the file transfer time. Thus, the same may be used to achieve efficient flow control.

In order to validate our analytical model, we use an experimental setup in which two machines are connected using a 1-Gbps Ethernet link. By comparing the experimental results with the analytical model, we demonstrate that the *effective bottleneck rate* determined using our analytical model and chosen as the sending rate yields a file transfer time very close to the experimentally observed optimal value of the sending rate. This validates the importance of the analytic model in determining the end-system bottleneck.

This paper is organized as follows. Section II describes the proposed SRN-based queuing network model to characterize the end-system. In Section III, we illustrate how some parameters of SRN model such as processing rates may be determined. Section IV presents an evaluation of our proposed analytical model with experimental results on a 1 Gbps Ethernet testbed. We conclude this paper in Section V.

II. Modeling and Analysis of the End-system

We determine the *effective bottleneck* rate by using the following three steps:

A: *Service-Time Distribution Analysis*:

We model the end-system as a SRN queuing network model. We determine the service-time distribution of the processing of the interrupt service routine calls generated by the NIC, as a function of the end-system workload. It is very difficult to obtain a closed-form expression for this distribution because of the complexity of the queuing model representation of the end-system. An alternative is to use the *tagged customer* approach [11]. The interrupt-handling process is considered to be the *tagged customer*. Its service-time distribution conditioned on the state of the queueing network at the time of arrival of the tagged customer, is then computed.

B: *NIC Packet Loss Analysis*:
We model the NIC as a $M/M[N]/1/K$ bulk-service finite-capacity queue. The arrival rate is the sending data rate. The service corresponds to interrupt processing, and the service rate is determined from the service time distribution in the previous step. Moreover, each service occurs in a batch of fixed size N; this is to model interrupt coalescing, in which a group of packets is served by a single interrupt service routine. The queue has finite capacity K.

C: *Determination of Effective Bottleneck Rate*:
Given the packet loss probability, we determine the time to transfer a file of known size on a link of known propagation delay, as a function of the sending data rate. The *effective bottleneck rate* is the sending rate which yields the minimum transfer time.

We now describe each of the three above steps in greater detail.

A. Service-Time Distribution Analysis

Queuing networks have been successfully used to model the performance of computer systems, particularly for issues such as resource contention [12]. In order to obtain the *effective bottleneck rate*, we wish to determine the service-time distribution of interrupt servicing in the presence of other end-system tasks.

Computing the service-time distribution using the tagged customer approach is a two-step process. The first step involves determining the steady-state probability vector for the queuing network without the tagged customer, called *steady-state analysis*. The second step uses the above steady-state probability vector to compute the time-to-absorption distribution. This step is called *transient analysis*.

To construct a representative model of the end-system, we adopt a variation of the Stochastic Petri Net (SPN) called Stochastic Reward Net (SRN), proposed in [8]. Graphically, a Petri Net is a directed graph with two disjoint types of nodes: *places* and *transitions*. Directed input/output arcs connect from a place to a transition and from a transition to a place, respectively. A positive integer called *multiplicity* may be marked for each arc. Each place may have zero or more tokens to start with, which is known as *marking*. A *transition* is enabled when each of its input places has at least as many tokens as the multiplicity of the arc. When a transition occurs, a number of tokens equal to the *multiplicity* is delivered from the input place to the output place. In SPN, a transition time may be associated with each transition. Moreover, this transition time may be dependent on the current *marking* in the SPN.

A SRN is obtained from the SPN by associating reward rates with the markings. With the help of such a reward rate, the time to reach a particular marking may be determined. A SRN can be automatically converted into a Markov reward model, thus permitting the evaluation of not only performance and availability but also their combination. More details about the SRN are available in [12] and are omitted here for the sake of brevity. Commercial packages such as Stochastic Petri Net Package (SPNP) [5] may be used to solve the SRN.

In order to model the end-system as a SRN, we classify tasks executing at the end-system into different categories as follows:

1) *CPU-bound tasks:* These tasks are computationally intensive and have high processor utilization. Examples are simulation and visualization-based tasks. Since such tasks have long execution times and are not interactive, they are usually assigned a lower dynamic priority by the OS.
2) *Input/Output (I/O)-bound tasks:* These tasks are I/O intensive, and they spend significant time waiting for a peripheral device other than the NIC. Examples are a text editor or a task interacting with a disk-subsystem. Processor utilization is therefore significantly low. In order to provide better interactive experience to users and to improve utilization of peripheral devices, the OS assigns them a high dynamic task priority. Since network I/O is modeled as a tagged customer, it is not classified into this task category.

We consider a closed queuing network model, in which the number of tasks of each task category is available and constant. The SRN model for steady-state and transient analysis is shown in Fig. 1. Circles are used to denote places P, and rectangular boxes are used to denote transitions T. The processor (CPU) is assumed to follow the Processor Sharing (PS) service discipline and is modeled as a place. In PS, the processor allocates equal share to each executing task if a long time interval were to be considered. The scheduling of most OSs is approximately close to PS.

The service rate of a processor in PS is dependent on the number of tasks. This is modeled using marking-dependent input/output arcs. For example, the transition from the processor to the disk of the I/O task is dependent on the number of tasks queued at the processor. The number of CPU-bound tasks is known and is denoted by $\#C$. The number of I/O tasks instantaneously queued at the processor is denoted by $\#i$. Therefore, the transition rate for an I/O task from processor to disk is defined by:

$$T_{CPU->Disk} = \frac{\#i * \mu_{P\_I/O}}{\#i + \#C} \tag{1}$$

This is represented as an output arc in Fig. 1. The naming notations for the service times of different tasks are shown in Table I.

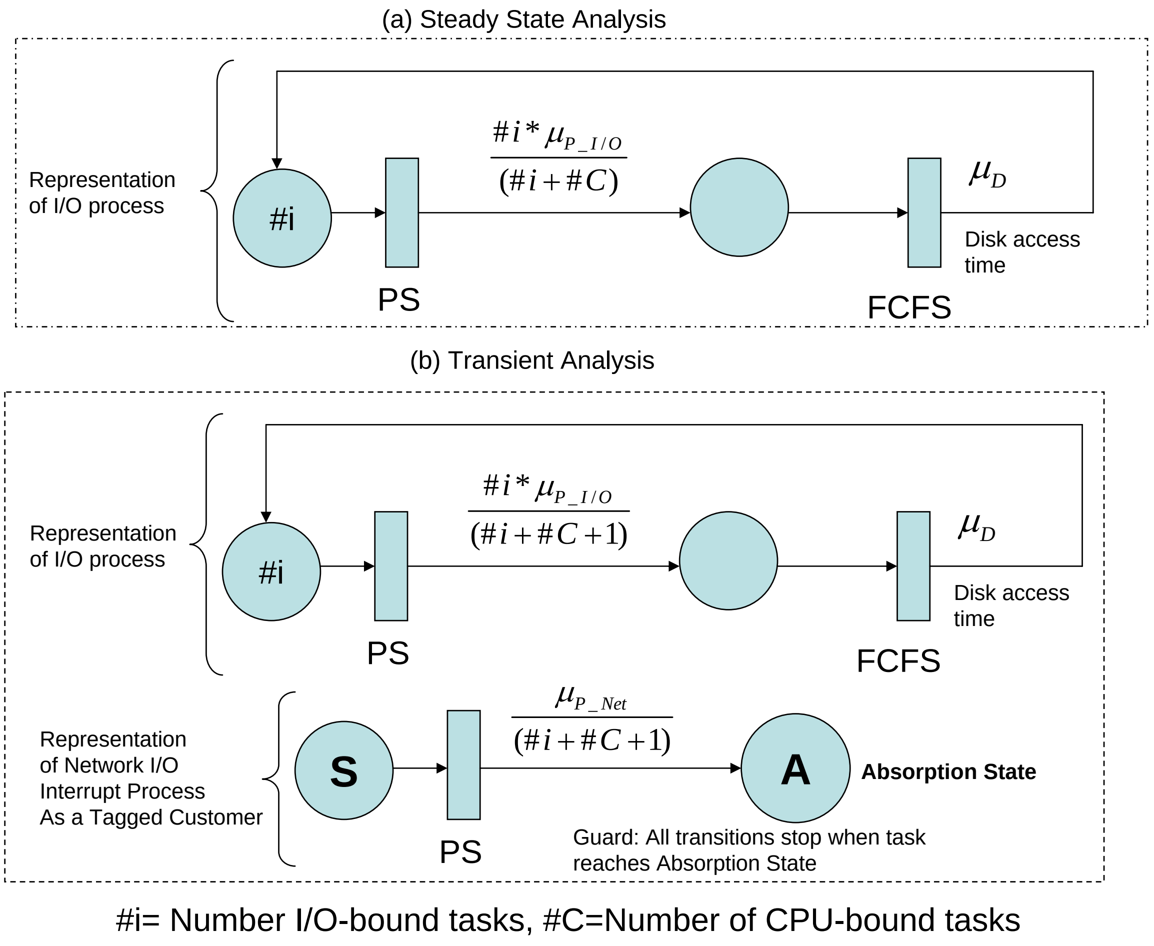

(a) Steady State Analysis
Representation of I/O process
#i
$\frac{\#i * \mu_{P\_I/O}}{(\#i + \#C)}$
μ_D
Disk access time
PS
FCFS
(b) Transient Analysis
Representation of I/O process
#i
$\frac{\#i * \mu_{P\_I/O}}{(\#i + \#C + 1)}$
μ_D
Disk access time
PS
FCFS
Representation of Network I/O Interrupt Process As a Tagged Customer
S
$\frac{\mu_{P\_Net}}{(\#i + \#C + 1)}$
A
Absorption State
PS
Guard: All transitions stop when task reaches Absorption State
#i= Number I/O-bound tasks, #C=Number of CPU-bound tasks

Fig. 1. SRN model to determine the *effective bottleneck rate*.

An I/O task interacts alternately with a peripheral device (e.g., disk, printer, etc.), which we broadly label as a disk and the processor. It is expected that the processor service time will be much less than the disk processing time for an I/O task (i.e., $\mu_{P\_I/O} >> \mu_D$). The disk is modeled as First-Come-First-Serve (FCFS) service since it serves requests in a sequential fashion. Since the CPU-bound task consumes a constant CPU workload, the fraction of the processor available to other tasks in a PS discipline is reduced by the corresponding factor.

For steady-state analysis, the SRN model in Fig. 1(a) is analyzed to determine the steady-state probabilities of different markings in the SRN (corresponding to the number of tasks at each place P). Thereafter, for transient analysis, network I/O is represented as an interrupt process. Service of this task corresponds to the CPU handling the interrupt and transferring the packets from the NIC buffer to the application with processing along the network protocol stack. This task is represented as a tagged customer.

Fig. 1(b) shows the SRN model for transient analysis. The response-time distribution for the tagged customer to reach the absorption state (A) from arrival state (S) is determined by associating a reward rate of 0 when there is no marking in the absorption state (A), and a reward rate of 1 otherwise. The above time distribution may then be determined as the expected value of the reward after a given time. The initial probabilities of different states determined from Fig. 1(a) may be easily incorporated by adding a vanishing initial marking that transfers to all the possible steady states with the specified probabilities. In this way, we determine the response-time distribution for an interrupt service routine.

TABLE I
NAMING NOTATIONS FOR SRN MODEL

Symbol	Notation
$\mu_{P\_I/O}$	Expected processing rate for I/O Task
μ_{P_Net}	Expected processing rate for Network I/O Interrupt
μ_D	Expected disk processing rate

B. NIC Loss Analysis

The NIC is modeled as a $M/M[N]/1/K$ bulk-service finite-capacity queuing model. NIC cards transfer data to main memory using Direct Memory Access (DMA). However, the portion of memory reserved for DMA by the device driver is limited. For example, the device driver for the Broadcom BCM5700 1-Gbps Ethernet NIC that we employed in our experimental studies reserved 96 KBytes of memory. This is modeled by the finite capacity K in our queuing representation. NICs also have the feature of interrupt coalescing, in which an interrupt is generated only when a certain number of packets arrive. When the interrupt is processed, all these packets are served as a bunch. For example, the BCM5700 generates one interrupt for every 6 arriving packets. The same is modeled using a bulk-service queue, where service occurs in a batch of size N. Packet arrival at the NIC is modeled as a memoryless process with mean rate equal to the sending rate. The service corresponds to the OS processing the interrupt, which is akin to the tagged customer in the SRN model.

The $M/M[N]/1/K$ queue does not have a closed-form expression for packet loss. However, it may be easily solved by constructing the SPN shown in Fig. 2. An output arc of multiplicity N is used to denote bulk service. An inhibitor arc of multiplicity K is used to prevent arriving packets to be added to the buffer when the buffer is full. λ is the mean packet arrival rate, and μ is the mean interrupt servicing rate, which may be determined from the response-time distribution of the interrupt-handling process from the SRN model, as illustrated above.

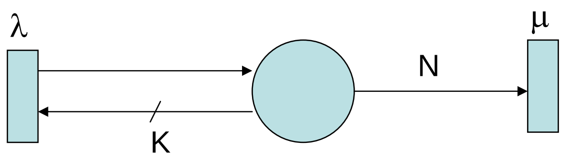


Fig. 2. SPN to represent the $M/M[N]/1/K$ queue.

C. Determination of Effective Bottleneck Rate

We consider the Reliable Blast UDP (RBUDP) [10] as an example of a rate-based UDP transport protocol. Given the loss probability determined in the above step, and neglecting the impact of any other losses, we may estimate the time to transfer a file of known size as follows:[10].

$$\begin{aligned} T &= [T_{prop} + \frac{S_{total}}{B_{send}}] \\ &\quad +[(N_{resend} * T_{prop}) + \sum_{i=1}^{\lfloor N_{resend} \rfloor} \frac{l * S_{send_{i-1}}}{B_{send}}] \\ &\quad +[(N_{resend}+1) * (\frac{S_{total}}{8 * S_{pkt} * B_{send}} + T_{prop}))] \end{aligned} \tag{2}$$

where:

- T is the estimated transfer time,
- T_{prop} is the one-way propagation delay,
- S_{total} is the total of the file size and the overhead,
- B_{send} is the sending rate,
- N_{resend} is the number of times to resend, which may be calculated from the loss rate as:

$$N_{resend} = log_l(S_{pkt}/S_{total}) \tag{3}$$

- l is the loss probability,
- $S_{send_{i-1}}$ is the payload to be sent in the i^th iteration,
- S_{pkt} is the packet size, S_{pkt}=1468 bytes considering that the MTU of an Ethernet frame is 1500 bytes, and the UDP-over-IP overhead is $(8+24=32)$ bytes.

The transfer time is estimated as the sum of transmission and propagation delays involved to transfer the file. This consists of three terms. The first term represents the time to transfer the entire file. The second term represents the cumulative time to send the packets corresponding to the error sequence numbers in multiple iterations of sending. The third term represents the time to transmit the lost-packet sequence numbers from the receiver to the sender.

The end-system *effective bottleneck rate* is defined as the sending rate which yields the minimum file transfer time, i.e., B_{send} which yields the minimum value of T.

We note that, although we have described a simple model for an uniprocessor system and a single network I/O task, it may be easily extended to Symmetrical Multi Processor (SMP) systems and multiple network I/O tasks. Additional processors may be represented as places in the SRN model, and the transitions may be appropriately modeled to achieve load balancing among the processors. Similarly, multiple network I/O tasks may be modeled similar to I/O tasks, while one of them is considered as a tagged customer for end-system bottleneck analysis.

III. Determining the Model Parameters

In order to employ the queuing network model described above to determine the *effective bottleneck rate,* we must determine the service-time distributions for processing of the I/O-bound and the interrupt service routine calls, and the current workload at the end-system (i.e., the number of CPU-bound and I/O-bound tasks currently executing). The service-time distributions quantify the hardware aspects such as processing rates. These are static parameters which must be determined only once. The workload quantifies the dynamic conditions at the end-system.

In order to determine the service-time distributions, we leverage MAGNET (Monitoring Apparatus for General kerNel-Event Tracing)[9]. MAGNET allows us to timestamp each event with the CPU cycle counter which is the highest-resolution clock available on most machines. We examine the end-system with the following representative task workloads corresponding to each task category:

1) *Input/Output (I/O)-Bound Task*: We choose a task which reads a randomly generated line number from a very large file stored in a disk. Let the mean time interval between successive disk accesses, as measured by MAGNET, be $t_{inter-disk}$ and the mean time interval for a disk access be t_{disk}. We determine the service times as:

$$\mu_{P\_I/O} = \frac{1}{t_{inter-disk}} \qquad \mu_D = \frac{1}{t_{disk}}$$

2) *Interrupt-Handling Task*: We choose a task which is receiving data through the NIC. Similar to the above method, $\mu_{P_{Net}}$ was determined from the context-switch intervals to serve the interrupt generated by the NIC.

The above representative workloads are used to measure the processor service-time distributions on an unloaded machine. In the presence of other tasks running on the machine, the service time distributions do not differ by much, except for some very high service time values (outliers), when the OS is possibly handling some alternate task which may be starving.

As a consequence, the loss rate may be slightly higher than that reported by the analytical model. Some of the effects of the outliers may be handled by providing dynamic feedback from the end-system, using RAPID [7].

We note that the above representative I/O workload is only an approximation of the execution plan of a general I/O-bound task. Given the variety of I/O-bound tasks, it is extremely difficult to characterize the CPU processing and peripheral device service rates for all of them. Hence, the representative workload is used as an example for the processing parameters for a general I/O-bound task in our analytical model. Although the above may be an approximation, its impact is minimal because the CPU is usually the bottleneck resource, and I/O-bound tasks have low CPU utilizations.

The end-system workload may be determined from OS parameters. For example, Linux and Free BSD OS maintain a *proc* directory which records different kernel execution parameters. A system command such as *top* may be used to parse the proc file systems and present a real-time view of the workload. Using *top*, we may determine the total number of tasks. Tasks may be either in running state or in sleeping state. CPU-bound tasks and I/O-bound tasks may be classified based on the average sleep-time values. The OS maintains the average sleep time of every task by adjusting its value, each time the task wakes out of sleep, or gives up processing voluntarily or involuntarily. With these steps, we obtain the parameters to evaluate the SRN model in Fig. 1.

The parameters of the $M/M[N]/1/K$ queue to model the NIC buffer may be determined from the device driver parameters. For the Broadcom 5700 1-Gbps Ethernet Adapters used in our experimental studies, the device driver allocated memory of 96 KBytes which would correspond to $K = 64$ maximum-sized Ethernet frames (MTU=1500 bytes). The interrupt coalescing factor N was 6 Ethernet frames.

In the next section, we compare how the results from the analytical model compare with that in an experimental setting.

IV. Experimental Results and Discussions

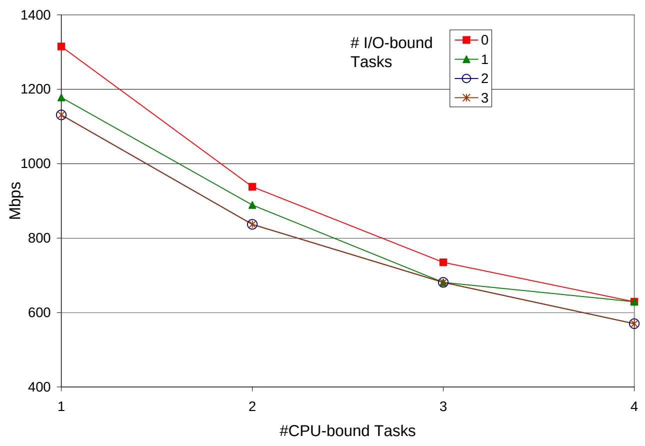


Fig. 3. *Effective bottleneck rate* versus number of tasks.

We connected two Dell SX280 machines (with Intel Pentium 3.2 GHz processors and 1 GByte RAM) back-to-back using a 1-Gbps Ethernet link. Using the MAGNET version 2.0 [9] toolkit, we determined the processing parameters with the representative workload as described in Section III. The measured parameters are shown in Table II.

TABLE II
Measured values of processing parameters using MAGNET

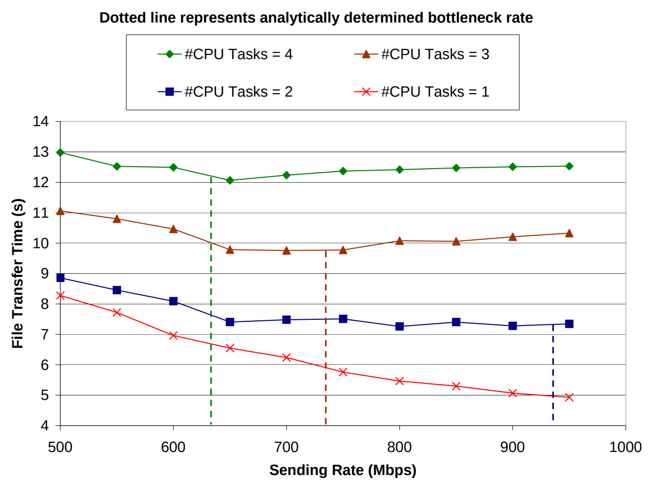
Symbol	Expected Rate per ms
$\mu_{P\_I/O}$	4.6
μ_{P_D}	1.01
μ_{P_Net}	41.6

We apply these parameters to solve the analytical model described in Section II. We determined the *effective bottleneck rate* as a function of the number of CPU-bound tasks and the number of I/O-bound tasks executing in parallel. The number of CPU-bound tasks executing was varied from 1 to 4. The number of I/O-bound tasks executing was varied from 0 to 3. An example of a system running 2 CPU-bound and 2 I/O-bound jobs in parallel is a system running two parallel visualization applications, the data from which is periodically updated in a file. We limit our study to 4 CPU-bound and 3 I/O-bound tasks because most systems are not expected to have a higher task load. Besides, the bottleneck rate does not decrease much beyond the above task load.

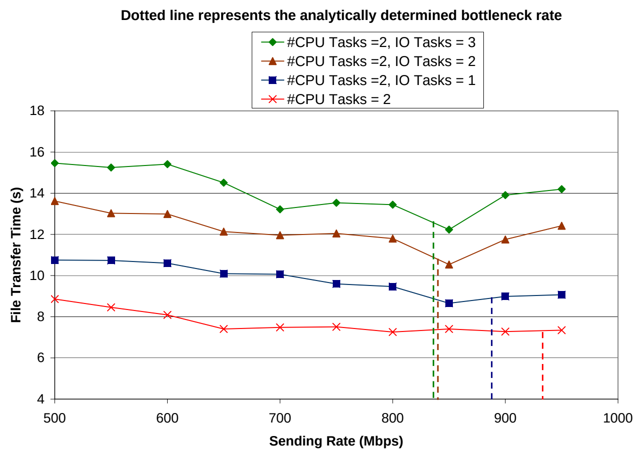
The *effective bottleneck rates* are shown in Fig. 3. As expected, the *effective bottleneck rate* decreases with higher workload. We observe that the slope tapers down with increasing workload. The drop in the bottleneck rate is sharper with increasing number of CPU-bound tasks than increasing I/O-bound tasks. This is because the I/O-bound tasks do not compete much for the CPU.

Having determined the *effective bottleneck rate* using the analytical model, our aim was to verify how it compares with an experimental study. We used the RBUDP protocol in the QUANTA 1.0 package [4] as an example of a rate-based UDP transport protocol. In order to emulate propagation delay, we delay the start of each UDP burst from the sender by the propagation delay, and similarly delay the feedback from the receiver. The file size was 500 MB and the propagation delay was 100 ms, the same as those values studied in the analytical model. We emulated a CPU-bound task by using an *infinite for-loop* which uses CPU cycles constantly, and an I/O-bound task using a process that reads a randomly-generated line from a file. The above I/O-bound task is the same as the representative workload used to determine Table II. The sending rate for the RBUDP protocol was varied from 500 Mbps to 950 Mbps in increments of 50 Mbps, note that 950 Mbps was the highest rate at which the sender could blast UDP packets on a line operating at a maximum transmission rate of 1 Gbps. For each sending rate, the experiment of transferring the payload was carried out 20 times, and the mean value of the file-transfer time was recorded.

The file-transfer time for different sending rates and for different CPU-bound loads with no I/O-bound task executing is shown in Fig. 4(a), and for different I/O-bound task loads with 2 CPU-bound tasks is shown in Fig. 4(b). As the sending rate is increased from 500 Mbps, the file-transfer time decreases. However, beyond a certain rate, the packet losses become quite



(a) Varying number of CPU-bound tasks.



(b) Varying number of I/O-bound tasks.

Fig. 4. Experimental results for different workloads (dotted vertical line represents the analytically-evaluated bottleneck rate).

high as the receiving end-system is not able to handle packets at such a high rate. Beyond this rate, the file-transfer time increases. The objective of our analytical study was to attempt to evaluate this rate which will lead to an optimal file-transfer time. The dotted vertical lines represent the *effective bottleneck rate* determined by the analytical model. We observe that the measure of the *effective bottleneck rate* provided by the analytic model yields a file-transfer time which is very close to the rate at which the minimum value of file-transfer time that is achieved. This illustrates the ability of the analytical model to determine the *effective bottleneck*. For all cases in which the number of CPU-bound tasks is greater than one, the *effective bottleneck rate* is lesser than the line rate of 1 Gbps, indicating that the receiving end-system was indeed the bottleneck. In a system setup of a line rate of 10 Gbps, the receiving end-system would be a bottleneck irrespective of the workload. We therefore expect our analytical model to be a very useful means of evaluating the system bottleneck at high line rates.

V. Conclusion

In this work, we consider the challenge of the end-system being the bottleneck for high-speed data transfer. We propose a mechanism to evaluate the best rate at which network performance may be extracted from such an end-system, given the number of executing tasks (the workload), and other factors such as the file size and the propagation delay on the network path. The *effective bottleneck rate* is estimated by: (i) determining the expected service time of an interrupt service routine call from a Stochastic Reward Net (SRN) queuing model, (ii) applying the above service rate for loss analysis of a $M/M[N]/1/K$ queuing model representation of a NIC buffer, and (iii) determining the sending rate which yields the minimum transfer time for a file on a specified path, given the above queuing loss.

Comparing with experiments on a 1-Gbps Ethernet testbed, we observe that the *effective bottleneck rate* determined analytically compares well with the data rate which achieves the best file-transfer time.

References

[1] CANARIE at http://www.canarie.ca/about/index.html
[2] DoE UltraScienceNet at http://www.csm.ornl.gov/ultranet/
[3] National LambdaRail at http://www.nlr.net
[4] QUANTA 1.0 package developed by EVL available at http://www.evl.uic.edu
[5] Stochastic Petri Net Package (SPNP), available at http://www.ee.duke.edu/k̃st/
[6] "Time for TOE: The benefits of 10 Gbps TCP Offload," *Chelsio Communications White Paper*, May 2005.
[7] A. Banerjee, W-c. Feng, B. Mukherjee, and D. Ghosal, "RAPID: An End-System Aware Protocol for Intelligent Data Transfer over Lambda Grids," *Proc. IEEE/ACM IPDPS 2006*, Rhode Island, Greece, 2006.
[8] G. Ciardo, A. Blakemore, P. Chimento, J. Muppala, and K. Trivedi, "Automated generation and analysis of Markov reward models using Stochastic Reward Nets," in C. Meyer and R.J. Plemmons, editors, *Linear Algebra, Markov Chains, and Queuing Models, IMA Volumes in Mathematics and its Applications*, vol. 48, Springer-Verlag, Heidelberg, Germany, 1992.
[9] M. Gardner, W. Feng, M. Broxton, A. Engelhart, and G. Hurwitz, "MAGNET: A Tool for Debugging, Analysis and Adaptation in Computing Systems," *Proc., CCGrid 2003*, Tokyo, Japan, May 2003.
[10] E. He, J. Leigh, O.Yu, and T. DeFanti, "Reliable Blast UDP : Predictable High Performance Bulk Data Transfer," *Proc., IEEE Cluster Computing*, Chicago, Illinois, 2002.
[11] J. Muppala, K. Trivedi, V. Mainkar, and V. Kulkarni, "Numerical computation of response time distributions using stochastic reward nets," *Annals of Operations Research*, no. 48, pp. 155-184, 1994.
[12] K. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, Second Edition, Wiley, 2002.