A Decentralized Scheduler for Distributed Video Streaming in a Server-less Video Streaming System[1]

C. Y. Chan and Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong
{cychan2, yblee}@ie.cuhk.edu.hk

Abstract

Recently, a server-less video-on-demand architecture has been proposed to eliminate costly dedicated video servers and yet is highly scalable and reliable. However, because of the potentially large number of user hosts streaming video data to a receiver for playback, the aggregate network traffic can become very bursty, leading to significant packet loss at the access routers. To tackle this problem, we propose a novel network-aware transmission scheduler called Least Schedulable First (LSF) to reduce the traffic burstiness. Simulation results show that LSF can reduce the congestion-induced packet loss from over 95% to 0.05% in a 500-host system at 0.95 system utilization. Moreover, LSF can adapt to variations in the underlying network, is inherently distributed, and does not require synchronization among hosts in the system.

1. Introduction

Peer-to-peer (P2P) and grid computing have shown great promises in building high-performance and yet low cost distributed computational systems. By distributing the workload to a large number of low-cost, off-the-shelf computing hosts such as PCs and workstations, one can eliminate the need for a costly centralized server and at the same time improve the system's scalability. Many of the current works on P2P and grid computing focus on computational problems [1-3] and on the design of middleware [4-6]. In this study, we focus on another application – video streaming systems, and in particular, investigate the problem of scheduling data transmission in such a distributed system.

Unlike conventional video streaming systems, a decentralized video streaming system does not require video server at all [7-10]. Instead, video data are distributed to user hosts and these user hosts cooperatively serve one another's streaming workload. This novel architecture, however, poses many new challenges not found in traditional video streaming systems.

In particular, with potentially large number of nodes streaming data to one another, the aggregate network traffic can become very bursty. This could lead to substantial congestion at the access network as well as the user nodes receiving the video data. A previous study [11] has revealed that the packet loss due to congestion can exceed 95% if one does not explicitly schedule the data transmissions to avoid network congestion.

In an earlier study [11] we have investigated two *network-neutral* schedulers, namely the staggered scheduler and the randomized scheduler, to tackle this problem. These network-neutral schedulers do not require knowledge of the underlying network but can still significantly reduce congestion-induced packet losses.

Intuitively if the underlying network's properties are known, we should be able to exploit the knowledge to reduce the congestion-induced packet loss even further. This is the focus of this paper, where we present a new *network-aware* transmission scheduler called Least Schedulable First (LSF) that exploits knowledge of the underlying network to outperform the previous two network-neutral schedulers.

2. Background

[1] This work was supported in part by the Hong Kong Special Administrative Region Research Grant Council under a Direct Grant, Grant CUHK4211/03E, and the Area-of-Excellence in Information Technology.

We review in this section the decentralized server-less video streaming architecture [9] studied in this paper and formulate the transmission scheduling problem. Interested readers are referred to the original studies [9-10] for more details on the server-less video streaming architecture.

2.1 Server-less Video Streaming Architecture

A server-less video streaming system comprises a pool of user hosts, or called nodes, connected together by a network such as a broadband network or the Internet. Each node has a system software that can stream a portion of each video title to, and playback video data received from, other nodes in the system. Unlike conventional video server, this system software serves a much lower aggregate bandwidth and thus can readily run on today's set-top boxes (STBs) and PCs.

In this server-less architecture, a video title is first divided into fixed-size data blocks and then distributed to all nodes in the cluster. To start a video streaming session, a receiver node will first locate the set of sender nodes carrying blocks of the desired video title, the placement of the data blocks and other parameters (format, bitrate, etc.) through a directory service, and then send a request to all sender nodes to initiate streaming the video blocks to the receiver node for playback. For larger systems with potentially tens of thousands of nodes, we can further divide the nodes into independent and autonomous clusters to keep the control traffic overheads to acceptable levels.

Let N be the number of nodes in the cluster and assume all video titles are constant-bit-rate (CBR) encoded at the same bitrate R_v. A sender node in a cluster may have to retrieve video data for up to N video streams, of which N-1 of them are transmitted while the remaining one played back locally. Note that as a video stream is served by N nodes concurrently, each node only needs to serve a bitrate of R_v/N for each video stream. With a round-based transmission scheduler, a sender node simply transmits a data block to each receiver node in each round. With multiple receiver nodes thus we need to determine the order of transmissions within a round. This is the transmission scheduling problem investigated in this work.

2.2 Network Congestion

In video streaming, video data are often assumed to be transmitted in a continuous, fluid-flow-like, data stream at a constant bit-rate to a receiver. However, in practice data are always transmitted in discrete packets (e.g. over RTP/UDP) and thus the data stream is inherently bursty in a small time scale. This burstiness is usually insignificant in traditional client-server video streaming systems because only a single video server will be transmitting video data to a client machine and thus the outgoing data packets will be evenly spaced out at constant time intervals.

By contrast, nodes in a server-less video streaming system all participate in transmitting video data packets to a receiver node for playback. Thus without proper coordination, these multiple streams of data packets could combine into very bursty traffic that ultimately can lead to network congestion and packet losses.

For example, in a previous work [11] we studied a straightforward transmission scheduler called the On Request Scheduler (ORS), which determines the transmission schedule simply based on the arrival time of the receiver's request during admission. In particular, the sender nodes will schedule data transmission to begin from the first available timeslot within a round. Not surprisingly, this simple scheduler resulted in significant network congestion and packet losses exceeding 95%.

The reason for the exceedingly poor performance is that despite variations in network delay, the arrival times of the receiver's requests at the sender nodes are still highly correlated. Thus once transmission begins, the sender nodes in the system will all transmit video data packets to the receiver node at around the same time, leading to serious network congestion.

Therefore the key to the solution is to decorrelate the packet transmission times. We investigated two such algorithms, namely the Randomized Scheduler (RS) and the Staggered Scheduler (SS), in a previous work [11]. The Randomized Scheduler decorrelates the data transmission times by randomizing the transmission order in each round. The Staggered Scheduler on the other hand explicitly spread out the transmission times of packets destined to the same receiver to smooth out the aggregate traffic. Simulation results showed that these two schedulers can both significantly reduce congestion-induced packet losses, e.g., from over 95% (ORS) to 9.3% (RS) and 0.16% (SS). Interested readers are referred to [11] for more details.

3. A Model for Transmission Scheduling

The two transmission schedulers presented in Section 2.2 are network-neutral, i.e., they do not make use of any knowledge of the underlying network properties (e.g. delay). The question then is whether we can reduce the packet loss even further by exploiting knowledge of the network.

To this end we need to address three challenges. First, we need to formulate the transmission scheduling problem *in terms of* the network model. Second, we need to find a way to estimate properties of the underlying network. Finally, armed with knowledge of the network, we need to devise a transmission scheduling algorithm to exploit the knowledge to further reduce congestion-induced packet loss. We address the first two challenges in this section and then present a network-aware transmission scheduling algorithm in Section 4.

3.1 A Matrix Representation

Despite the complexity of the transmission scheduling problem, we can devise a very concise mathematical model to capture all the essential features of the system. We first define three N-by-N matrices **S**, **D**, and **R**, where their row i column j element, denoted by $(i,j)^{th}$ element, represents respectively the schedule time, network delay, and arrival time of the packet transmission from node i to node j. Next we introduce a fourth matrix **C** with its $(i,j)^{th}$ element representing the clock difference that node i lags behind node j. Using these four matrices, we can then describe the system in the following equation:

$$\mathbf{S}+\mathbf{D}+\mathbf{C}=\mathbf{R} \quad (1)$$

where + is matrix operation. In other words, the arrival time is equal to the schedule time plus network delay and clock jitter.

Note that as transmission can occur only at the beginning of a timeslot, the elements in the matrix **S** must be integer multiples of the duration of a timeslot T_s. The elements in the matrices **D**, **C**, and **R** however, can take on any real number values. We employ three modifications to further simply this model.

First, we introduce an *apparent delay* matrix, denoted by **A**, which is defined as **A**=(**D**+**C**). Since a summation of two constants is another constant, this substitution removes redundancy in the representation:

$$\mathbf{S}+\mathbf{A}=\mathbf{R} \quad (2)$$

Second, we convert the matrices to integer matrices by quantizing the matrix elements with the timeslot duration T_s. In other words, we replace $s_{i,j}$, $a_{i,j}$, and $r_{i,j}$ by $\texttt{round}(s_{i,j}/T_s)$, $\texttt{round}(a_{i,j}/T_s)$, and $\texttt{round}(r_{i,j}/T_s)$ respectively. Thus with N timeslots in a round, the valid schedule time is $s_{i,j} \in \{0,1,\ldots,(N-1)\}$.

Third, we observe that in case the sum of network delay and clock jitter is large, the packet arrival times for a particular receiver may span over multiple rounds. This can be easily compensated by starting the transmission in different rounds in different sender nodes to offset the delay variations. This can be modeled by applying (mod N) to (2):

$$\mathbf{S}+\mathbf{A} \equiv \mathbf{R} \pmod{N} \quad (3)$$

With this technique we can always keep the arrival time to within a round's duration, i.e., $r_{i,j} \in \{0,1,\ldots,(N-1)\}$.

Fig. 2 illustrates this model using a system with three nodes, i.e., N=3. Consider the $(1,2)^{th}$ elements of the matrices. From the schedule matrix **S**, node 1 has scheduled the transmission to node 2 at timeslot 2 according to its own clock. From the apparent delay matrix **A**, the sum of network delay and clock difference between node 1 and node 2 is 2 timeslots. Thus packets transmitted from node 1 will arrive at node 2 at timeslot (2+2) mod 3 = 1, i.e., timeslot 1.

Using this matrix representation, we can formally define the constraints and the goal of the transmission scheduling problem. Specifically, assuming that each node can send a packet in each timeslot in each round, then the transmission schedule defined by the matrix **S** must not have repeating elements in any of the row. For example, with N=3, a row containing elements of values (0,2,1) is a valid schedule representing the schedule of transmission to node 0 in timeslot 0, to node 1 in timeslot 2, to node 2 in timeslot 3, and so on. This type of matrix is also known as row-latin matrix [12]. By contrast, the schedule (0,2,2) is invalid because transmissions to both node 1 and node 2 are scheduled in the same timeslot number 2.

On the other hand, we want the arrival time matrix **R** to have non-repeating elements in any of the columns, also known as column-latin matrix [12]. As each column represents the arrival time of packets transmitted from the N sender nodes, repeating elements represent overlapping arrival times and hence could induce congestion/packet loss.

Therefore our goal in the transmission scheduling problem is, given **D** and **C**, to find a transmission schedule **S** that is row-latin such that the arrival time matrix **R** is column-latin. We note that although related matrix problems, latin squares in particular, have been studied extensively in the literatures [12-16], no known solution exists for the specific problem in (3).

3.2 Network Delay and Clock Jitter Estimation

The previous discussions assume that the network delay matrix **D** and the clock jitter matrix **C** are known. Obviously we cannot assume *a priori* knowledge of these properties in a distributed system running on the Internet. Thus in this section we address the second problem, namely to obtain estimates of the matrices **D** and **C** at run time.

For network delay estimation, a well-known technique is to use echo messages. A node i will send an echo packet to another node j, which then immediately replies node i with a reply packet. The time from sending the echo packet to receiving the reply is the round-trip time (RTT) and the one-way delay can then be estimated from RTT/2.

However, this echo technique implicitly assumes that the network path between the two nodes is symmetric, i.e., the network delay is the same for both directions of the path. However, previous studies [17-18] have shown that in general network paths in the Internet are asymmetric, thus reducing the accuracy of this network delay estimation method.

One way around this problem is to measure the one-way network delay directly, i.e., by comparing the transmission time and the arrival time of a packet. However, if the clocks in the sender node and the receiver node are not precisely synchronized, then we simply cannot obtain the one-way delay by subtracting the transmission time, measured by the sender node's clock, from the arrival time, this time measured by the receiver node's clock. While in principle we can implement and deploy distributed clock-synchronization protocols [19-20] to reduce the clock jitter to improve estimation accuracy, this nonetheless will create an additional hurdle to deploying such a decentralized system.

Surprisingly, although it is not possible to measure the one-way network delay without node synchronization, we discover that we can measure the *sum* of one-way network delay and clock jitter in a single step – Jitter-Adjusted Delay Estimation (JADE).

Consider the individual elements in A, denoted by $a_{i,j}$. We can express it in terms of $d_{i,j}$ and $c_{i,j}$:

$$a_{i,j} = d_{i,j} + c_{i,j} \tag{4}$$

To estimate the apparent delay $a_{i,j}$, node i sends a message to node j at time $x_{i,j}$ according to an arbitrary time reference. After traversing the network link with a delay of $d_{i,j}$ the message will reach node j at time $y_{i,j}=x_{i,j}+d_{i,j}$ according to the same time reference. Let δ_i be the clock difference that node i lags behind the time reference. Thus we can compute $c_{i,j}$ from $c_{i,j}= \delta_i-\delta_j$. Substituting $x_{i,j}$, $y_{i,j}$, δ_i and δ_j into (4) we can then obtain

$$\begin{aligned} a_{i,j} &= (y_{i,j} - x_{i,j}) + (\delta_i - \delta_j) \\ &= (y_{i,j} - \delta_j) - (x_{i,j} - \delta_i) \end{aligned} \tag{5}$$

Note that $(y_{i,j}-\delta_j)$ is simply the packet reception time *as measured by* node j's clock, and $(x_{i,j}-\delta_i)$ is simply the packet transmission time *as measured by* node i's clock. Now both entities can be measured independently by the sender node i and the receiver node j. Thus we can compute $a_{i,j}$ directly from (5) without the need for node or clock synchronization.

4. Least-Schedulable-First Scheduler

With the system model formulated and the network parameters estimated, our goal then is to find a row-latin schedule matrix **S** such that the resultant arrival time matrix **R** is column-latin. The trivial method is to enumerate all permutations of **S** until we find a solution. However, given that a row-latin schedule matrix **S** can have $(N!)^N$ permutations, this brute force approach is clearly not practical. For example, enumerating **S** takes only a few CPU cycles for N=3, 124 milliseconds for N=4, but 2.7 hours for N=5 using a Pentium-4 class machine.

On the other hand, it can be shown that the problem in general may not even have a solution. Thus instead of finding only the schedule matrix **S** that results in column-latin matrix **R**, we relax the goal to finding the schedule matrix **S** to reduce the number of colliding arrival times in the arrival time matrix **R**.

In the following, we present a Least Schedulable First (LSF) scheduler that greedily selects a schedule to minimize the number of collisions for a new receiver. We present the admission and scheduling algorithm in Section 4.1 and analyze its performance in Section 4.2.

4.1 Admission and Scheduling

Suppose there are v active video streams running in a cluster of N nodes. Without loss of generality, let node j be the particular node that initiate the video session at this moment. When node j sends the requests to the other N-1 nodes, the other nodes will send back replies carrying the list of the N-v idle timeslots, denoted by L_i, and the request reception time $(p_{i,j}-\delta_i)$ for calculating the apparent delay $a_{i,j}$.

Using these information the receiver node can then compute the *schedulability* of the arrival timeslots, represented by an N-by-N Boolean matrix $\mathbf{\Theta}$ with its $(i,k)^{\text{th}}$ element denoted by $\theta_{i,k}$. Specifically, we set $\theta_{i,k}$=1 if the arrival timeslot k is schedulable by node i, i.e.,

$$k - a_{i,j} \in L_i \tag{6}$$

Otherwise we set $\theta_{i,k}$=0.

Physically, the number of 1's in a column, say column k, in the matrix $\mathbf{\Theta}$ indicates the number of nodes that can schedule their packets to arrive at the arrival timeslot k. We define the schedulability of arrival timeslot k as

$$\eta_k = \sum_{i=0}^{N-1} \theta_{i,k} \tag{7}$$

The receiver node then schedules the arrival timeslots in order of increasing schedulability. Thus the next arrival timeslot to be scheduled, denoted by κ, is determined from

$$\left\{\kappa : 0 < \eta_\kappa \le \eta_k, \forall k \in \{0,1,...,N-1\}\right\} \tag{8}$$

The intuition behind this algorithm is that there is a one-to-one mapping between sender and arrival timeslot. Thus for each arrival timeslot scheduled there will be one fewer sender for scheduling the remaining timeslots. Hence arrival timeslots with lower schedulability are more likely to become unschedulable when all suitable senders are assigned to other arrival timeslots. An unscheduled timeslot will have no packet arrival, which implies the packet will arrive at another scheduled timeslot, resulting in a collision. Thus the receiver node will always schedule the least schedulable arrival timeslot first to reduce the likelihood of collisions.

This is illustrated in Fig. 3 with node 0 and node 1 already streaming video and node 2 is requesting for a new video session. Using the JADE algorithm the apparent delay to node 2, i.e. $a_{i,2}$ for i=0,1,2, are first estimated. Next, we compute the schedulability matrix **Θ**. Consider sender node 0 as an example, the transmission timeslot 1 is available and the arrival timeslot $1+a_{0,2}=1+1=2$ will be schedulable by node 0.

To determine the next arrival timeslot to schedule, we then look for the arrival timeslot with the lowest and yet non-zero schedulability, i.e. arrival timeslot 2 in Fig. 3. In this case there is only one sender schedulable and so we assign node 0 to schedule the transmission to node 2 at timeslot $2-a_{0,2}=2-1=1$. The matrix **Θ** is then updated by setting all elements in row 0 (i.e., sender 0 is no longer available) and column 2 to zero (i.e., the arrival timeslot 2 has been scheduled).

This process then repeats until all the elements in the matrix **Θ** become zero. If all sender nodes have been scheduled then this result in a transmission schedule with no collision. Otherwise, we need to re-initialize the rows in the matrix **Θ** for the remaining sender nodes by restoring their original elements in the matrix **Θ**, i.e. (0,1,0) for node 2. The scheduling process is then repeated until all sender nodes are scheduled.

4.2 Performance Analysis

In the LSF algorithm the likelihood of scheduling collision increases when the number of schedulable timeslots decreases, i.e., when the system utilization increases. Thus a key factor to the algorithm's performance is the system utilization. To investigate this factor we derive in the following the relation between system utilization and the availability of timeslots.

Let v be the number of active video sessions in the system. Then there will be v distinct transmission timeslots in each node that are occupied and the corresponding v arrival timeslots are not schedulable. We assume that the v non-schedulable arrival timeslots are all randomly distributed in each row of **Θ**.

Consider a row of the matrix **Θ**, the probability that $w(\le v)$ particular elements are zero is given by

$$\frac{\mathrm{C}_{v-w}^{N-w}}{\mathrm{C}_{v}^{N}} \tag{9}$$

Thus, the probability that w particular columns are zero, denoted by p_w, is equal to

$$\left(\frac{\mathrm{C}_{v-w}^{N-w}}{\mathrm{C}_{v}^{N}}\right)^N \tag{10}$$

On the other hand, we can obtain the same probability in (10) by conditioning on the number of zero-columns in the matrix. Specifically, if there are exactly u, $w \le u \le v$, zero-columns, then the probability that the w particular columns are non-zero is:

$$\frac{\mathrm{C}_{u-w}^{N-w}}{\mathrm{C}_{u}^{N}} = \frac{\mathrm{C}_{w}^{u}}{\mathrm{C}_{w}^{N}} \tag{11}$$

By total probability theorem [21], we can get p_w by:

$$\sum_{u=w}^{v} \frac{\mathrm{C}_{w}^{u}}{\mathrm{C}_{w}^{N}} \Pr[u] \tag{12}$$

where $\Pr[u]$ denotes the probability that the matrix **Θ** has exactly u columns. Thus equating (10) and (12), we can obtain

$$\left(\frac{\mathrm{C}_{v-w}^{N-w}}{\mathrm{C}_{v}^{N}}\right)^N = \sum_{u=w}^{v} \frac{\mathrm{C}_{w}^{u}}{\mathrm{C}_{w}^{N}} \Pr[u] \tag{13}$$

Finally, and surprisingly, by putting w=1 in (13) we can obtain the formula for the expected value of u:

$$E[u] = \sum_{u=1}^{v} u \Pr[u] = \left(\frac{\mathrm{C}_{v-1}^{N-1}}{\mathrm{C}_{v}^{N}}\right)^N \mathrm{C}_{1}^{N} = \frac{v^N}{N^{N-1}} \tag{14}$$

Note that the expected value of u represents the expected number of non-schedulable arrival time slots. The existence of non-schedulable time slots implies there will be scheduling collisions in other time slots. Thus, (14) quantifies the extent of scheduling collision and relates that to the system utilization (i.e., v/N).

For example, we can apply (14) to find the operating point at which $E[u]$=1:

Table 1 - Default system parameters

Parameters	Values
Cluster size	500
Video block size	8KB
Video bitrate	4Mbps
Access network bandwidth	$1.1R_v$
Router buffer size (per node)	32KB
Mean propagation delay	0.005s
Variance of propagation delay	10^{-6}
Mean router queueing delay	0.005s
Variance of clock jitter	10^{-6}
Video length	7200s
System Utilization	0.95

$$v = N^{\frac{N-1}{N}} \Rightarrow \frac{v}{N} = N^{-\frac{1}{N}} \quad (15)$$

In other words, if the system utilization exceeds (15) the expected number of collisions will increase beyond 1. Thus by keeping the system utilization below this operating point (e.g. by rejecting new requests) we can maintain a low level of collision for existing video streams. We further illustrate this observation in Section 5.3.

5. Performance Evaluation

In this section, we evaluate and compare LSF with other scheduling algorithms studied in this paper using simulation. The simulator simulates a network with 500 nodes. To generate a realistic network topology, we implement the extended BA model proposed by Barabási et al. [22] as the topology generator, using parameters measured by Govindan et al. [23].

To model access routers in the network, we assume an access router have separate buffers for each connected node. These buffers are used to queue up incoming data packets for transmission to the connected nodes. When the buffer is full, then subsequent arriving packets for the node will be discarded and thus resulting in packet loss.

To model the network links, we separate the end-to-end delay into two parts, namely, propagation delay in the link and queueing delay at the router. While the propagation delay is primary determined by the link's physical distance, queueing delay at a router depends on the utilization of the outgoing links. The propagation delay for a link is a constant drawn from a normally distributed random variable and the queueing delay at a router is modeled by an exponentially-distributed random variable [21]. To model clock differences among nodes, we assume that the clock jitter of a node, defined as the deviation from the mean time of all hosts, is normally-distributed with zero mean. We can then control the amount of clock jitter by choosing different variances for the distribution.

To model the dynamic activities of the system, we allow nodes to initiate videos in a stochastic fashion. Specifically, when a node initiates a video title, its stream will last for t_{video} seconds. When the video stops, the node will be idle for an exponentially-distributed random duration with mean t_{idle} seconds. Thus by adjusting the two parameters t_{video} and t_{idle} we can control the system utilization, $\rho = t_{video}/(t_{video}+t_{idle})$.

Table 1 summarizes the default values of various system parameters. We investigate in the following sections the effect of three system parameters, namely cluster size, network delay, and system utilization on the performance of the scheduling algorithms in terms of packet loss rate. Each set of results is obtained from the average results of 10 randomly generated network topologies.

5.1 Sensitivity to Cluster Size

Fig. 4 plots the packet loss rate versus cluster size ranging from 5 to 500 nodes. There are two observations. First, the loss rates of all schedulers decrease rapidly at smaller cluster size and become negligible for very small clusters. For example, for a 10-node cluster the loss rate is only 6.6% even for the ORS algorithm. This confirms that the traffic collision problem is unique to a server-less video streaming system where the number of nodes is typically large.

Second, comparing the three algorithms, ORS performs extremely poorly with loss rates as high as 95%, which is clearly not acceptable in practice. RS performs significantly better with loss rates approaching 9.3% when the cluster size is increased to 500. By exploiting knowledge of the network and making efficient use of idle timeslots, the proposed LSF scheduler performs best with a loss rate of only 0.05% for a cluster size of 500 nodes.

5.2 Sensitivity to Network Delay Variations

Given that the transmission schedule computed from LSF is deterministic, random variations in the network delay will degrade its performance. To investigate this effect, we plot in Fig. 5 the packet loss rate against the mean network delay from 20-500ms. We vary the mean network delay by varying the mean queueing delay of the routers in the network. Thus increasing the mean network delay will also increase the delay variations.

There are two interesting observations from this result. First, the performance of the RS algorithm is

independent of the underlying network delay. This is because packet transmission times under RS are already randomized, and thus adding further random delay to the packet transmission times has no effect on the resultant traffic burstiness.

Second, the performance of LSF converges to that of RS when the mean network delay is very large (e.g. 500ms). This is because the large variations in the network delay effectively randomize the packet arrival times at the access router. However, according to a recent study [24] the Internet have far lower delay and delay variations, with mean delay in the range of 20-40ms. Thus the LSF scheduler will likely perform significantly better than the other algorithms in practice.

5.3 Sensitivity to System Utilization

As mentioned in Section 4.2, system utilization is a key element in determining the performance of LSF. Therefore, we plot in Fig. 6 the packet loss rate versus system utilization, ρ, ranging from 0.8 to 0.99 for different cluster sizes under LSF.

We again have two observations from this result. First, LSF performs consistently for almost the whole range of system utilization, but only deteriorates drastically at extremely high system utilization. For instance, for N=300, the packet loss rates for ρ=0.8, 0.9 and 0.99 are 0.040%, 0.046% and 0.70% respectively. Since the instantaneous utilization of a video system is usually moderate, LSF can be deployed for low packet loss rate. To guarantee the performance of LSF, one can limit the system utilization by simply blocking requests as suggested in Section 4.2.

Second, the critical points for which LSF starts its sharp deterioration increase with cluster size. Since critical points cannot be easily defined, we show in the same plot the bounds of system utilization obtained from (15), i.e. ρ=0.955, 0.981 and 0.988 for N=100, 300 and 500 respectively. These results show that a larger system can support higher utilization. Besides, the plot has also indicated the strong relation between the congestion-induced packet losses and the expected number of collisions – cases with the same expected number of collisions will have similar packet loss rate for any cluster size, N.

6. Conclusions and Future Work

In this study, we investigated the transmission scheduling problem in a server-less video streaming system. Specifically, we formulated the transmission scheduling problem as a matrix mathematical model and discovered that it is possible to perform one-way network delay estimation with clock jitter accounted for in a single step. This discovery led to the development of the Least Schedulable First (LSF) scheduler that exploits knowledge of the network properties to reduce the congestion-induced packet loss to negligible levels. The LSF algorithm is inherently distributed and hence does not require synchronization among hosts in the system.

Our results have shown that the LSF algorithm will likely perform very well in the Internet, where delay and delay variations are modest. With the rapid growth in wireless and even ad-hoc networks, medium-term variations in the network delay may increase substantially. Further study is thus warranted to investigate adaptive scheduling algorithms for these highly-dynamic mobile and wireless networks.

7. References

[1] A. Oram, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly Press, USA, 2001.

[2] C. Padgett, and K. Kreutz-Delgado, "A grid algorithm for autonomous star identification," *IEEE Transactions on Aerospace and Electronic Systems*, Vol.33(1), Jan. 1997, pp.202-213

[3] SETI@home. http://setiathome.ssl.berkeley.edu/.

[4] M. Baker, R. Buyya, and D. Laforenza, "The Grid: International Efforts in Global Computing," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, Rome, Italy, 31 July, 2000.

[5] Condor Project. http://www.cs.wisc.edu/condor/.

[6] The Globus Project. http://www.globus.org/.

[7] M. Hefeeda, A. Habib, D. Xu, and B. Bhargava, "CollectCast: A Tomography-Based Network Service for Peer-to-Peer Streaming," *ACM SIGCOMM'03*, Karlsruhe, Germany, August 2003.

[8] T. P. Nguyen, and A. Zakhor. "Distributed Video Streaming over the Internet," *Multimedia Computing and Networking (MMCN)*, January 2002.

[9] Jack Y. B. Lee, and W. T. Leung, "Study of a Server-less Architecture for Video-on-Demand Applications," *Proc. IEEE International Conference on Multimedia and Expo.*, Lausanne, Switzerland, 26-29 Aug 2002.

[10] Jack Y. B. Lee, and W. T. Leung, "Design and Analysis of a Fault-Tolerant Mechanism for a Server-Less Video-On-Demand System," *Proc. 2002 International Conference on Parallel and Distributed Systems*, Taiwan, 17-20 Dec, 2002.

[11] C. Y. Chan, and Jack Y. B. Lee, "On Transmission Scheduling in a Server-less Architecture," *Proc. International Conference on Parallel and Distributed Computing*, Klagenfurt, Austria, August 26-29, 2003.

[12] C. F. Laywine, and G. L. Mullen, *Discrete Mathematics Using Latin Squares*, New York: Wiley, 1998.

[13] J. Denes, and A. D. Keedwell, *Latin Squares: New Developments in the Theory and Applications*, New York: North Holland, 1991.

[14] D. Donovan, "The Completion of Partial Latin Squares," *Australasian Journal of Combinatorics*, 22, 2000, 247-264.

[15] G. G. Chappel, "A Matroid Generalization Of A Result On Row-Latin Rectangles," *Mathematics Subject Classification*, 1991.

[16] B. D. McKay, and I. M. Wanless, "Most Latin squares have many subsquares," *J. Combinatorial Theory (A)*, vol.86, 1999, pp.323-347.

[17] K. Claffy, H.-W. Braun, and G. Polyzos. "Measurement considerations for assessing unidirectional latencies," *Internetworking: Research and Experience*, vol.4(3), September 1993, pp. 121-132.

[18] A. Wolman, G. Voelker, and C. A. Thekkath, "Latency Analysis of TCP on an ATM Network," *Proceedings of the USENIX Winter '94 Technical Conference*, San Francisco, CA, Jan. 1994, pp.167-179.

[19] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transaction on Communications*, vol.39(10), Oct. 1991, pp.1482-1493.

[20] Simple Network Time Protocol. http://www.faqs.org/rfcs/rfc2030.html.

[21] D. Gross, and C. M. Harris, *Fundamentals of Queueing Theory*, 3rd ed. New York: Wiley, 1998.

[22] R. Albert, and A.-L. Barabási, "Topology of Evolving Networks: Local Events and Universality," *Physical Review Letters*, vol.85, 2000, pp.5234-5237.

[23] R. Govindan, and H. Tangmunarunkit, "Heuristics for Internet Map Discovery," *IEEE Infocom 2000*, Tel Aviv, Israel, Mar. 2000, pp.1371-1380.

[24] G. Hooghiemstra and P. Van Mieghem, 2001, "Delay Distributions on Fixed Internet Paths," *Delft University of Technology*, Report20011020.

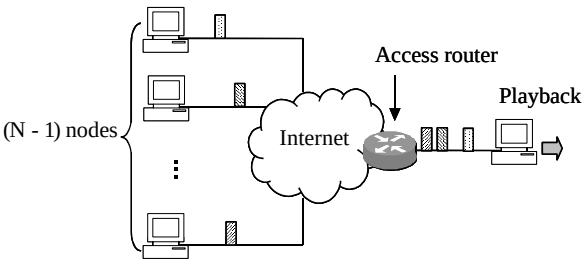

Fig. 1 A *N*-node server-less video streaming system

S			+	A			≡	R			(mod 3)
0	2	1		0	1	1		0	0	2	
0	1	(2)		1	0	(2)		1	1	(1)	
2	0	1		0	2	0		2	2	1	

Fig. 2 The transmission scheduling model for a 3-node system (The $(1,2)^{th}$ elements are circled and used as an example in Section 3.1)

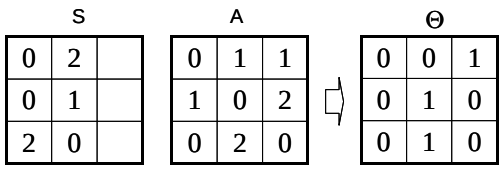

Fig. 3 Illustration of the LSF algorithm in computing the schedulability matrix for receiver node 2

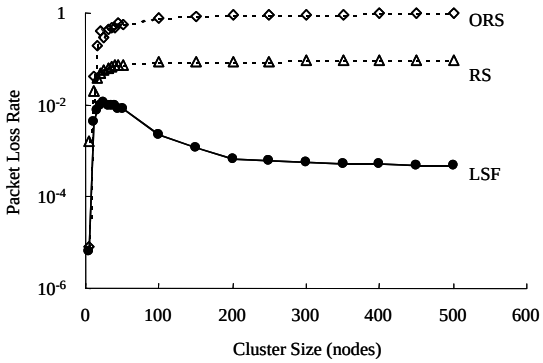

Fig. 4 Packet loss rate versus cluster size

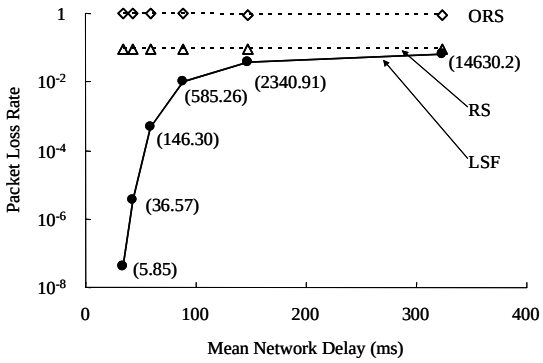

Fig. 5 Packet loss rate versus mean network delay (numbers in brackets are the variance of network delay)

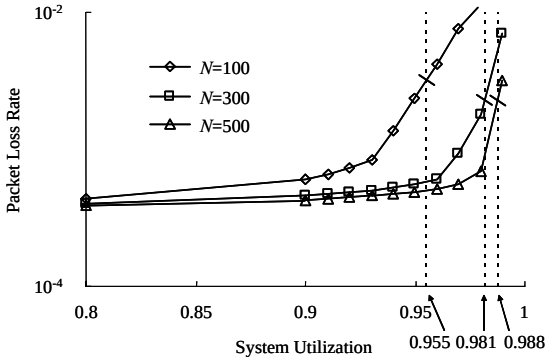

Fig. 6 Packet loss rate versus system utilization for different cluster sizes under LSF