Gradient-Descent Scheduler – A Network-Aware Transmission Scheduler for Server-less Video Streaming Systems

C. Y. Chan, Jack Y. B. Lee
Department of Information Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
Email: jacklee@computer.org

M. Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, N.T., Hong Kong
Email: hamdi@cs.ust.hk

***Abstract* - Recently, a server-less architecture has been proposed for building video streaming systems which does not need any costly dedicated video servers and yet is highly scalable and reliable. However, due to 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. This study tackles this problem by investigating a novel network-aware transmission scheduling algorithm called *Gradient-Descent Scheduler* (GDS) to reduce the traffic burstiness. Simulation results will demonstrate that GDS can reduce the congestion-induced packet loss from over 95% to 0.07% in a 500-host system. Moreover, GDS can automatically adapt to the underlying network and does not require hosts in the system to be synchronized. These are essential for practical design of server-less architectures and peer-to-peer systems.**

***Keywords* – Transmission scheduling, video streaming, server-less architecture.**

I. INTRODUCTION

Peer-to-peer (P2P) systems 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-shelve 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. Most of the current research on P2P and grid computing is focused on computational problems [1-3], and on the design of middleware [4-6]. In this work, we focus on another application of P2P architecture – video streaming systems, and in particular, investigate the problem of transmission scheduling in such a distributed system.

Existing video streaming systems are commonly built around the client-server architecture, where one or more dedicated video servers are used for storage and streaming of video data to video clients for playback. Recently, Lee and Leung [7] proposed a new server-less architecture for building video streaming systems that do not require dedicated video servers at all. In this server-less architecture, video data are distributed to user hosts and these user hosts cooperatively serve one another's streaming workload. Their early results have shown that such a decentralized architecture is both scalable [7] and reliable [8].

This work was funded in part by the RGC Earmarked Grant CERG4211/03E, Direct Grant, and the UGC Area of Excellence in Information Technology (AoE/E−01/99).

Nevertheless, there are still significant challenges in a server-less video streaming system. In particular, with potentially thousands of nodes streaming data to one another, the aggregate network traffic can become very bursty and this could lead to substantial congestion at the access network and the user nodes receiving the video data. Our simulation results revealed that packet loss due to congestion can exceed 95% if one does not explicitly schedule the data transmissions to avoid network congestion [9].

In a previous work [9] we also investigated the network congestion problem in a server-less video streaming system. We studied two transmission scheduling algorithms, namely staggered scheduler and randomized scheduler, both not making use of any knowledge of the network such as link delay. These *network-neutral* schedulers are relatively simple to implement and yet still manage to significantly reduce congestion-induced packet losses. In this work, we investigate a *network-aware* transmission scheduling algorithm called *Gradient-Descent Scheduler* (GDS) to reduce the traffic burstiness even further. Simulation results show that GDS can reduce the congestion-induced packet loss from over 95% to 0.07% in a 500-host system, can automatically adapt to the underlying network, and does not require hosts in the system to be synchronized.

II. BACKGROUND

We review in this section the server-less architecture [7] and then formulate the transmission scheduling problem.

A. Server-less VoD Architecture

A server-less video-on-demand (VoD) system comprises a pool of interconnected user hosts, or called nodes in this paper as shown in Fig. 1. Inside each node is a system software that can stream a portion of each video title as well as playback video received from other nodes in the system. A video title is first divided into fixed-size blocks and then equally distributed to all nodes in the cluster. This node-level striping scheme avoids data replication while at the same time share the storage and streaming requirement equally among all nodes in the cluster.

To initiate 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 the directory service. These sender nodes will then be notified to start streaming the video blocks to the receiver node for playback.

0-7803-8939-5/05/$20.00 (C) 2005 IEEE

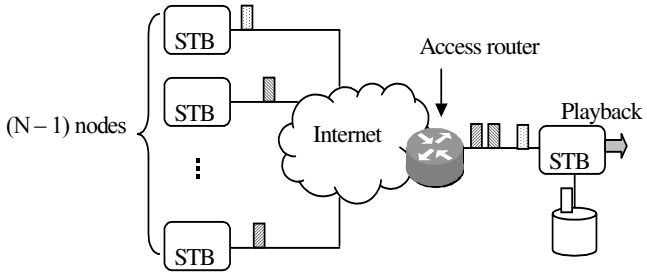

Fig. 1. A *N*-node server-less video-on-demand system.

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 are transmitted while the remaining one is played back locally. Note that since 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 one block of video data to each receiver node in each round. The ordering of transmissions for blocks destined to different nodes is the transmission scheduling problem we investigate in this research.

B. Network Congestion

In an ideal system, video data are transmitted in a continuous stream at a constant bit-rate to a receiver node. However, in practice data are always transmitted in discrete packets and thus the data stream is inherently bursty. This problem is 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 data packets will be transmitted at constant time intervals. By contrast, in a server-less VoD system, video data are distributed across all nodes and as a result, all nodes in the system participate in transmitting video data packets to a node for playback. If these data transmissions are not properly coordinated, a large number of packets could arrive at the receiver node's access network in a short time interval, thereby leading to network congestion and consequently packet loss.

For example, consider the straightforward transmission scheduler - On Request Scheduler (ORS) [9], which determines the transmission schedule based on the initial request arrival time. Specifically, a node transmits video data in fixed-duration rounds, with each round is further sub-divided into N timeslots. The node can transmit one Q-byte data packet in each timeslot of length T_s. Thus, the length of a round can be computed from $T_r=NT_s=NQ/R_v$.

When a node initiates a new video streaming session, it will send a request to all nodes in the system. A node upon receiving this request will reserve an available timeslot in a first-come-first-serve manner to begin transmitting video data for this video session. While this scheduler can smooth out the traffic leaving the sender, the combined traffic from multiple senders at the receiver becomes very bursty. In a simulation of a 500-node system with Q=8KB and R_v=4Mbps, ORS can result in over 95% packet losses due to congestion in the access network.

The fundamental problem here is due to the very large number of nodes in the system and the fact that data transmissions are packetized. With the ORS algorithm, a new video session will likely be assigned to timeslots that are temporally close together. Thus once transmission begins, all nodes in the system will transmit video data packets to the receiver node in a short time interval, and then all cease transmission for T_r seconds before transmitting the next round of packets. While the average aggregate transmission rate is still equal to the video bit-rate, the aggregate traffic is clearly very bursty and thus leads to buffer overflows and packet drops at the access network router connecting to the receiver node.

III. Network-Neutral Transmission Schedulers

One approach to tackle the congestion problem is to spread out the arrivals of packets from different senders. We briefly review in the following three *network-neutral schedulers* – schedulers that do not make use of any knowledge of the network.

A. Staggered Scheduler

The staggered scheduler spreads out the packet arrival times by explicitly staggering (or offsetting) the timeslots in different senders assigned to the same receiver. For example, data packets transmitted from node i to node j will always be transmitted in timeslot $(j–i–1) \bmod N$. Assuming the nodes are clock-synchronized, then transmissions from different nodes to the same receiver node will be separated by at least T_s seconds, thus eliminating the traffic burstiness problem in ORS. Simulation results show that SS can reduce the packet loss to as low as 0.18% compared to ORS's 95% packet loss.

Nevertheless, the need for clock-synchronization has two implications. First, as clocks in different nodes cannot be precisely synchronized in practice, the performance of the algorithm will depend on the clock synchronization accuracy. Second, depending on the application, the assumption that all nodes in the system are clock-synchronized may not even be feasible.

B. Randomized Scheduler

Alternatively, we can also reduce the likelihood of burst arrivals by randomizing the timeslot assignments on a round-by-round basis – randomized scheduler. It is easy to see that RS does not require clock synchronization among nodes in the system and hence is easier to deploy. Each node simply generates its own random schedule on a round-by-round basis. Simulation results show that RS can achieve consistent performance that is independent of network delay variations and level of clock synchronization. However, the packet loss rate generated by RS (9.3% under the same setting) is significantly higher than SS.

C. Staggered on Request Scheduler

Both of the previous two schedulers are sender-driven. Alternatively, we can also schedule the data transmissions by

the receiver – staggered on request scheduler (SORS). The principle of SORS is to let the receiver schedule the transmission timeslots to a staggered schedule. Specifically, suppose a receiver node initiates a new video session in timeslot k, then it will send out a request to node i at timeslot $k+i$. When a sender node receives this request, it will reserve an available timeslot in a first-come-first-serve manner as in ORS. In this way, the staggered scheduling is implicitly performed by the receiver node and the sender nodes can operate without synchronization. Simulation results show that SORS can reduce the packet loss to only 2.9%.

IV. Network-Aware Model

The three network-neutral transmission schedulers presented in Section III can already reduce the congestion-induced packet loss substantially. The question then, is whether we can reduce the packet loss even further by exploiting some knowledge of the underlying network.

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

A. A Matrix Representation

Despite the seeming complexity of the transmission scheduling problem, we can devise a very concise mathematical model to describe all the essential features of the system. We first define three N-by-N matrices **S**, **D** and **R**, where the $(i, j)^{\text{th}}$ element of **S**, **D** and **R** 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 $(i, j)^{\text{th}}$ element representing the clock difference between node i and j. Using these four matrices, we can then describe the system using the following equation:

$$\mathbf{S}+\mathbf{D}-\mathbf{C}\equiv\mathbf{R} \pmod{N} \qquad (1)$$

where the +, –, and mod are matrix operations.

To interpret the model, consider a particular element, say (i, j) in the equation. The receiving schedule $r_{i,j}$ is simply computed from the transmission schedule $s_{i,j}$ plus the network delay $d_{i,j}$ and the clock jitter $c_{i,j}$ between the sender node i and the receiver node j.

In principle, elements in the matrix **S** must be integer multiples of the length of a timeslot T_s in the round-based scheduler, while elements in the matrices **D**, **C**, and **R** can take on any real number values. We employ two modifications to further simply this model.

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

Second, 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. We can compensate for large delay variations by starting the transmission in different rounds in different sender nodes to offset the delay variations. With this technique we can always keep the arrival time to within a round's duration, i.e., $r_{i,j}\in\{0,1,\ldots,(N-1)\}$.

In this quantized model, we can formally define the constraint and 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 rows. Consider an example with N=5, a row containing elements of values {0, 2, 3, 1, 4} 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 [10]. By contrast, the schedule {0, 2, 2, 1, 4} is invalid because transmissions to both node 1 and node 2 are scheduled in the same timeslot number 2.

On the other hand, the arrival time matrix **R** must not have repeating elements in any of the columns, also known as column-latin matrix [10]. 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 literature [10-14], to the best of our knowledge the specific problem in (1) is new and no known solution exists.

B. 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 how 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. Furthermore, previous studies [15-16] have shown that in general network paths in the Internet are asymmetric, thus reducing the accuracy of this network delay estimation method.

0-7803-8939-5/05/$20.00 (C) 2005 IEEE

One way to get 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, this method suffers from another problem – clock jitter between different nodes. In particular, 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 [16-18] to reduce the clock jitter so as to improve estimation accuracy, this nonetheless will create an additional hurdle to deploying such a decentralized system.

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

First, we define a new apparent delay matrix, denoted by **A**, which are computed from **A**=(**D**−**C**). Then we can rewrite the system model in (1) in terms of A:

$$\mathbf{S}+\mathbf{A}=\mathbf{R} \pmod N \tag{2}$$

Next, we consider the individual elements in **A**, denoted by $a_{i,j}$. We can express them in terms of $d_{i,j}$ and $c_{i,j}$:

$$a_{i,j} = d_{i,j} - c_{i,j} \tag{3}$$

In estimating the apparent delay $a_{i,j}$, we need to at least send out a control message between any two nodes. Suppose a control message is sent from node i to node j at physical time (i.e., the absolute time according to a given time reference) $p_{i,j}$ and it reaches the destination at time $q_{i,j}=p_{i,j}+d_{i,j}$. Let δ_i be the clock difference between node i and the physical time. Thus $c_{i,j}$ can be computed from $c_{i,j}= \delta_i-\delta_j$. Substituting $p_{i,j}$, $q_{i,j}$, δ_i and δ_j into (3) we can obtain

$$\begin{aligned} a_{i,j} &= (q_{i,j} - p_{i,j}) - (\delta_i - \delta_j) \\ &= (q_{i,j} + \delta_j) - (p_{i,j} + \delta_i) \end{aligned} \tag{4}$$

Note that $(q_{i,j}+\delta_j)$ is simply the packet reception time *as measured by* node j's clock, and $(p_{i,j}+\delta_i)$ 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 (4) without the need for any clock-jitter adjustment.

V. Gradient-Descent Scheduler

With the system model being 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 a few cpu clocks for N=3, 124 milliseconds for N=4, but 2.7 hours for N=5.

On the other hand, the problem in general may not even have a solution at all. Thus instead of finding only the schedule matrix **S** that results in column-latin matrix **R**, we generalize the goal to finding the schedule matrix **S** that reduces the number of colliding arrival times in the arrival time matrix **R**.

In the following, we present a Gradient-Descent Scheduler that employs a probabilistic local search algorithm [19-21] to find **S** that minimizes the number of collisions in **R**.

A. Performance Metric

The key performance metric in evaluating a schedule matrix **S** is the number of collisions in the arrival time matrix **R**. To define the metric precisely, we consider how the computations are performed in a particular node, say, node j. When more than one packet arrives at node j at the same timeslot, they will collide and may cause congestion if the router buffer is full. In the matrix representation, collision occurs if the same integer appears more than once in the same column j of the matrix **R**.

Let *count* be a function that returns the number of elements in a finite set. Therefore, the number of the integer k appearing in column j, denoted by $\sigma_{j,k}$, can be obtained from

$$\sigma_{j,k} = count\{r_{i,j} = k, \forall i \in \{0,1,...,N-1\}\} \tag{5}$$

Since there is no collision if $\sigma_{j,k}\leq 1$, we compute the number of collisions at node j and timeslot k from

$$b_{j,k} = \max(\sigma_{j,k} - 1, 0) \tag{6}$$

and form an N-by-N collision matrix **B** with $b_{j,k}$ as the matrix elements. Finally, we can obtain the total number of collisions, denoted by Ω, from summing all the collisions:

$$\Omega = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} b_{j,k} \tag{7}$$

B. Optimization Algorithm

Fig. 2 lists the pseudo-code of GDS. First, the schedule matrix **S** is initialized with the staggered schedule generated by the Staggered Scheduler because of its good performance. As SS only converges to the randomized case when the network variation is large, the staggered pattern should provide a better, or at worst similar, initial condition than the randomized instances. Then, using the initialized matrix **S** and the matrix **A**, the collision matrix **B** and the total number of collisions Ω are computed in the function *compute_collisions*.

After the initialization process, GDS starts searching the local solution space through a swapping heuristic as defined in lines 14-16 in Fig. 2. The basic idea is to swap two scheduled timeslots of a sender to reduce the number of collisions. The algorithm first performs a local search to find out a pair of scheduled timeslots such that either one is collided. Next, it swaps the pair and if Ω is reduced, the swapping will be committed. Otherwise, the swapping will be rolled back. This process loops for every pair of scheduled timeslots of each sender and repeats until no swapping action is committed (i.e., cannot reduce Ω any further).

To speed up the algorithm, the collision matrix **B** and the integer Ω are updated in the function *update_collisions* after

each swapping. When a scheduled timeslot $s_{i,j}$ is about to change such that the arrival timeslot changed from x to y, only two elements of b, i.e. $b_{j,x}$ and $b_{j,y}$, are affected. Thus, a swap will require changes to only four elements in **B**. These changes in the matrix **B** can be directly added to Ω, which eliminates the need to recompute (7).

Similar to other local search algorithms, GDS does not guarantee finding the global optimal solution. To obtain better solutions, we can repeat the whole searching process with different initial matrix **S**. There are also other methods, such as k-change heuristic [20] and simulated annealing [21], to tackle this problem. It is beyond the scope of this study to address this optimization issue. Nevertheless, our experiments show that reasonably good results can be obtained from even a single iteration using the staggered schedule as the initial schedule matrix S.

Finally, we note that the GDS algorithm has a theoretical time complexity of $O(N^3)$ due to the three levels of loops. For example, the algorithm requires 0.421, 11.7 and 70 seconds of computation time for N=100, 300 and 500 respectively. Thus further optimization may be needed for systems with a huge number of nodes.

VI. Performance Evaluation

In this section, we evaluate and compare GDS with other scheduling algorithms using simulation. The simulator is developed using CNCL and it simulates a network with 500 nodes. To generate a realistic network topology, we implement the extended BA (EBA) 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 to have separate buffers for each connected node. These buffers are used to queue up incoming data packets for transmission to the connected node in case of bursty traffic. When the buffer is full, then subsequent arriving packets for the node will be discarded and thus resulting in packet loss.

End-to-end delay of network links is separated into propagation delay in the link and queueing delay at the router. While the propagation delay is primary determined by physical distance, queueing delay at a router depends on the utilization of the outgoing links and the queue size. We model the propagation delay and queueing delay as normally-distributed and exponentially-distributed random variables respectively [24]. The link delay data used in the GDS algorithm are obtained from (simulated) measurement using the JADE algorithm.

To model the clock synchronization protocol, we assume that the clock jitter of a node, defined as the deviation from the mean time of all hosts, to be normally-distributed with zero mean. We can then control the amount of clock jitter by choosing different variances for the distribution.

```
01. s_{i,j} is the scheduled timeslot for packet
    transmission from node i to node j
02. Ω is the total number of collisions
03. for(int i=0 to N-1){
04.   for(int j=0 to N-1){
05.     s_{i,j} = j-i-1;
06.   }
07. }
08. compute_collisions()
09. do{
10.   swapped = 0;
11.   for(int i=0 to N-1){
12.     for(int j=0 to N-1){
13.       for(int k=j+1 to N-1){
14.         if(either s_{i,j} or s_{i,k} is collided){
15.           if(swap(s_{i,j}, s_{i,k}) reduces Ω){
16.             perform_swap(s_{i,j}, s_{i,k});
17.             update_collisions();
18.             swapped ++;
19.           }
20.         }
21.       }
22.     }
23.   }
24. while (swapped > 0)
```

Fig. 2. Pseudo-code for the Gradient-Decent Scheduler

TABLE 1
DEFAULT SYSTEM PARAMETERS

Parameters	Values
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

To model the dynamic activities of the system, we allow nodes to initiate videos in a stochastic process. Specifically, when a node initiates a video title, its stream will last for a video length, denoted by t_{video}. When the video stops, the node will be idle for some time, which is an exponentially random variable with mean t_{idle}. 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 I summarizes the default values of various system parameters. We investigate in the following sections the effect of four system parameters, namely cluster size, router buffer size, queueing delay, clock jitter on the performance of the five scheduling algorithms in terms of packet loss rate. Each set of results is obtained from the average results of 10 randomly generated network topologies.

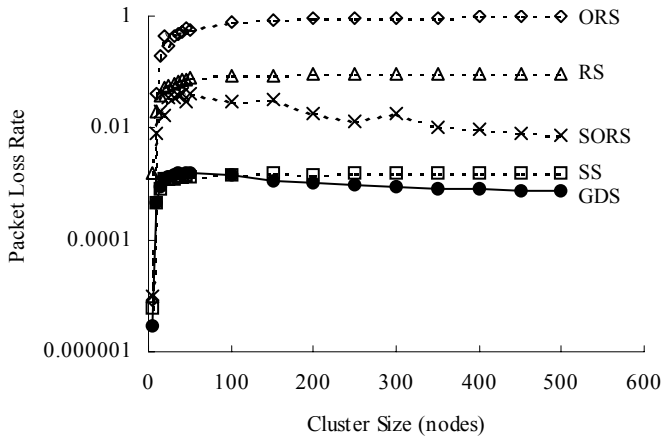

Fig. 3. Packet loss rate versus cluster size.

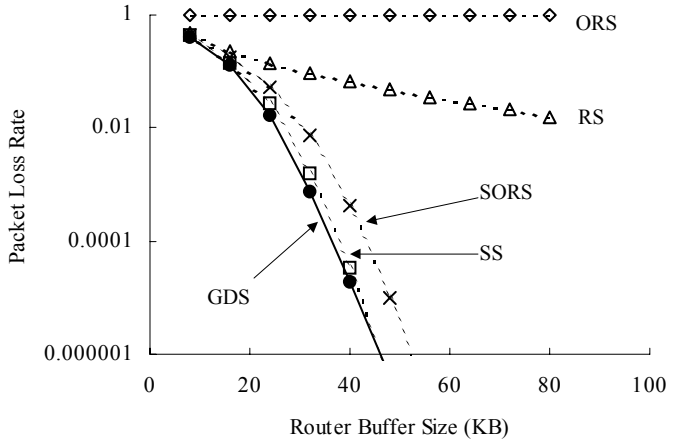

Fig. 4. Packet loss rate versus router buffer size.

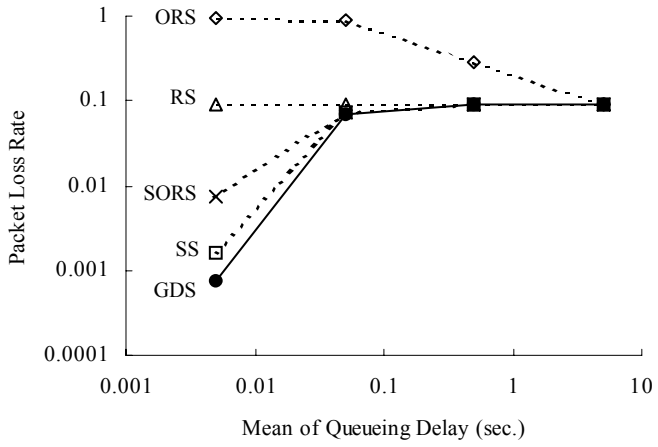

Fig. 5. Packet loss rate versus mean queueing delay of a single router.

A. Sensitivity to Cluster Size

Fig. 3 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% for ORS. This confirms that the traffic burstiness problem is unique to a server-less VoD system where the number of nodes is typically large.

Second, comparing the five algorithms, ORS performs extremely poorly with loss rates as high as 95%, which is clearly not acceptable in practice. RS and SORS perform significantly better, with the loss rates approaching 9.3% and 2.9% respectively when the cluster size is increased to 500. SS performs best among the four network-neutral algorithms, with 0.18% packet loss regardless of the cluster size when the nodes are clock synchronized. By exploiting knowledge of the network, the GDS algorithm performs best with a loss rate of 0.07% for a cluster size of 500 nodes.

B. Sensitivity to Router Buffer Size

To investigate the effect of the buffer size at the access router on the packet loss rate, we plot in Fig. 4 the packet loss rate against router buffer sizes ranging from 8KB to 80KB. With a video packet size of Q=8KB this corresponds to the buffer space for one to ten packets. As expected, the loss rates for all five algorithms decrease with increases in the router buffer size. The reduction in loss rate however, decreases more rapidly for SS, SORS, and GDS. By contrast, ORS and RS exhibit substantial packet loss even for buffer size as large as 80KB. Thus one cannot rely on simply increasing router buffer size to solve the congestion problem.

C. Sensitivity to Delay Fluctuation

On the other hand, delay fluctuations in the network can also affect performance of the schedulers. To study this effect, we vary the mean queueing delay of a single router from 0.005 to 5 seconds and plot the corresponding packet loss rate in Fig. 5. We note that queueing delay is modeled by an exponential random variable, where the variance is equal to the square of the mean. Thus the result also shows the effect of delay variations.

There are two interesting observations from this result. First, performance of the RS algorithm is not affected by changes in the mean queueing 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.

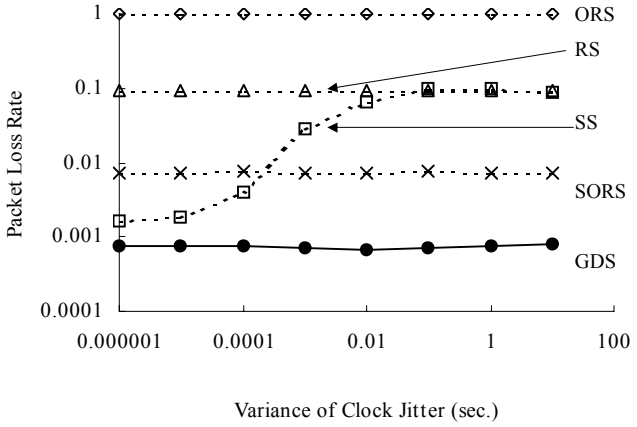

Fig. 6. Packet loss rate versus the variance of clock jitter.

Second, the performances of all five algorithms converge when the mean queueing delay is increased to 5 seconds. This is because when the mean queueing delay approaches the length of a service round (i.e. T_r=8.192 seconds), the random queueing delay then effectively randomizes the arrival times of the packets at the access router and hence performances of all algorithms converge to the performance of the RS algorithm. Nevertheless the average delay in the current Internet is significantly shorter than 5 seconds and thus the presented SORS, SS, and GDS algorithms can still be applied to reduce the congestion-induced packet losses.

D. Sensitivity to Clock Synchronization

Finally, we study in Fig. 6 the effect of clock jitter on the algorithms' performance. As expected, only the SS algorithm is affected by the magnitude of the clock jitter between nodes in the system. When the clock jitter is increased to beyond a variance of 0.0001, performance of the SS algorithm quickly deteriorates. This result clearly shows that although SS can perform well in a clock-synchronized system, the network-neutral algorithm SORS and the network-aware algorithm GDS are far more reliable in practice as accurate clock-synchronization is difficult, if not impossible, to achieve in a decentralized system running over the Internet.

VI. Conclusion and Future Works

In this work, we investigated the transmission scheduling problem in a server-less video streaming system. Specifically, we first addressed the problem of clock-synchronization in the existing staggered scheduling algorithm by presenting a new staggered-on-request scheduler that does not require clock synchronization at all and yet can still achieve robust performance across a wide range of network parameters. Next, by formulating the transmission scheduling problem as a matrix mathematical model, we 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 Gradient Descent Scheduler that exploits knowledge of the network properties to further reduce the congestion-induced packet loss to negligible levels. Nevertheless, the current GDS algorithm assumes that the network properties are stationary. The next step is to investigate dynamic algorithms to allow GDS to automatically adapt to the changing network conditions as well as system configurations.

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] 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.

[8] 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.

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

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

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

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

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

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

[15] 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.

[16] 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.

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

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

[19] C. H. Papadimitriou, and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, New York: Dover, 1998.

[20] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation*, New York: Springer, 1999.

[21] S. Russell, and P. Norvig, *Artificial Intelligence*, New Jersey: Prentice-Hall, 1995.

[22] R. Albert, and A.-L. Barabási, "Topology of Evolving Networks: Local Events and Universality," *Physical review letters*, vol.85, 2000, p.5234.

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

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

0-7803-8939-5/05/$20.00 (C) 2005 IEEE