Self-Diagnostic Peer-Assisted Video Streaming through a Learning Framework

Di Niu
Department of Electrical and Computer Engineering
University of Toronto
dniu@eecg.toronto.edu

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
bli@eecg.toronto.edu

Shuqiao Zhao
Multimedia Development Group
UUSee, Inc.
shuqiao.zhao@gmail.com

ABSTRACT

Quality control and resource optimization are challenging problems in peer-assisted video streaming systems, due to their large scales and unreliable peer behavior. Such systems are also prone to performance degradation in the event of drastic demand changes, such as flash crowds and large-scale simultaneous peer departures. In this paper, we demonstrate the deficiency of state-of-the-art video streaming systems by analyzing real-world traces from UUSee, a popular commercial P2P media streaming system based in China, during the 2008 Beijing Olympics. We show how simple machine learning techniques combined with periodic collection of statistics can be used for automated monitoring and diagnosis of peer-assisted video streaming systems. With such a framework, it is possible to estimate performance given certain resource usage patterns, making resource utilization more efficient. It also enables the prediction of large-scale performance degradation due to irregular demand patterns. The effectiveness of our proposed framework is validated with extensive trace-driven evaluations.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques; Modeling Techniques; C.2.3 [**Network Operations**]: Network Monitoring; Network Management

General Terms

Algorithms, Measurement, Performance, Reliability

Keywords

Video Streaming, Peer-to-Peer, Learning, Measurement, Diagnosis, Prediction, Resource Allocation, Troubleshooting

1. INTRODUCTION

It has been widely recognized that taking advantage of peer assistance is important to bandwidth-intensive applications, and live multimedia streaming in particular. By leveraging upload capacities of peers to upload to one another, the burden on server bandwidth is substantially alleviated. To achieve scalability and robustness in the presence of peer dynamics, it has been proposed that peers exchange availability information and explicit requests for content in a mesh topology [13]. Due to its simplicity and resilience to peer dynamics, such a design has been adopted in most real-world commercial peer-assisted live streaming systems.

A successful peer-assisted video streaming system involves a large number of media channels — on the order of tens of thousands — and attracts millions of users at peak times. Due to the unprecedented scale and the unpredictable behavior of end users using the system, it is a fundamental challenge to offer any kind of quality assurance to the users, even with a large collection of dedicated streaming servers [6]. This often causes unexpected performance bottlenecks and issues, which would best be avoided in media streaming systems that have stringent quality requirements.

In this paper, by analyzing the operational traces collected from UUSee Inc. — a major commercial P2P media streaming service in China — during the 2008 Beijing Olympics, we seek to identify problems and deficiencies that exist in real-world peer-assisted live streaming systems. We find that current systems lack appropriate mechanisms to utilize resources efficiently, which leads to unnecessarily high server costs. For example, the server bandwidth used to serve a channel sometimes exceeds the amount needed to maintain smooth playback by a substantial margin, while at the same time some other channels are suffering from performance issues. The UUSee system is also particularly prone to severe performance degradation in the presence of flash crowds or large-scale peer departures, which have occurred frequently in popular media channels. In contrast to a centralized solution that uses only dedicated servers in the cloud, one major obstacle that prevents peer-assisted media streaming from being reliable is that peer bandwidth contributions are much less reliable and much more dynamic than dedicated servers. Such bandwidth contributions are good to have as they reduce operating costs of servers, but are difficult to predict.

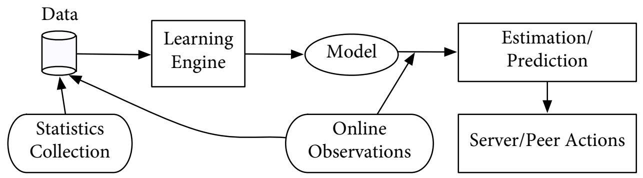

Figure 1: The concept of self-diagnostic peer-assisted streaming.

To provide a highly reliable streaming service, the health of peer-assisted systems should be carefully monitored, and both peers and dedicated servers should make their decisions based on such online network measurements. In this paper, we propose a new monitoring and learning framework, that enables the learning of internal system dependencies with the aid of periodic collection of online statistics.

The gist of our new framework is illustrated in Fig. 1. Once operating rules in the system have been determined by learning, they can subsequently be used to estimate performance under a certain resource configuration, to direct server and peer algorithms, and to forecast performance anomalies, based on online observations. The operating model can be initially learned through the collected trace data, and be refined online by more recently collected traces and fast online observations.

As a first step towards a self-diagnostic peer-assisted streaming architecture, we demonstrate the use of our monitoring and learning framework in two important applications: (1) making decisions on how much server bandwidth is to be provisioned in each media channel, when multiple channels co-exist; and (2) predicting large-scale performance degradation due to flash crowds or peer departures. Note that the solutions to these problems in current systems heavily depend on heuristics and human expertise. Our proposed algorithms are designed to target actual problems that have been observed from the UUSee system, and their effectiveness is also evaluated based on real-world traces.

The remainder of the paper is organized as follows. Sec. 2 describes our learning framework in general. We then analyze the existing performance issues in UUSee streaming system in Sec. 3, and shed lights on how such a learning framework could improve system performance and reliability. Sec. 4 focuses on explaining how the network behavior under its normal states may be learned. In Sec. 5, we demonstrate the application of our learning framework in the context of provisioning server bandwidth across different channels in the UUSee system. In Sec. 6, we describe and evaluate a learning algorithm based on neural networks, with an objective of predicting performance anomalies caused by flash crowds or peer departures. Sec. 7 reviews previous work related to this paper. In Sec. 8, we draw conclusions and outline directions for future research.

2. DEPENDENCY CHARACTERIZATION THROUGH LEARNING

Although current-generation peer-assisted streaming systems can provide satisfactory streaming quality to users in general cases, they are not without deficiencies. These systems usually incorporate fine-tuned partner selection algorithms [7] and block request mechanisms [12] to maximize the upload contribution from peers. However, much less effort is invested in optimizing how server resources should be used. As server resource usage is directly linked to the monetary cost incurred to a streaming service provider, it should be meticulously managed to save cost, yet with an optimized user experience. In fact, it has been recognized in [9, 10] that carefully provisioning server capacities across channels that coexist in a peer-assisted system can bring salient benefits with regard to the aggregate system performance. In reality, extreme peer dynamics such as flash crowds and peer departures have also posed additional challenges to peer-assisted streaming systems. In these cases, the system performance often becomes unstable, degrading the quality of media streams presented to users.

Despite their constant effort in improving streaming protocols at individual peers, current systems still suffer from the above problems, due to the lack of a fundamental understanding of the run-time relationships among demand, server bandwidth supply, peer bandwidth supply and performance. Unlike streaming services using a client-server architecture, peer contributions in a peer-assisted system is highly dynamic and unpredictable. Relying on peer bandwidth contributions may cause performance issues, yet underutilizing peer contributions may incur unnecessary server costs. To provide optimized and highly reliable peer-assisted streaming services, we need a mechanism that can automatically learn and diagnose the network behavior.

Table 1: Frequently used notations.

N_t^c	the number of peers in channel c at time t
S_t^c	the total server bandwidth allocated to channel c at time t
$\overline{s}_t^c$	the average server sending rate per peer in channel c at time t
$\overline{u}_t^c$	the average peer sending rate in channel c at time t
$\overline{r}_t^c$	the average peer receiving rate in channel c at time t
$\overline{q}_t^c$	the ratio of peers with smooth playback in channel c at time t
$\overline{P}_t^c$	the average number of partners per peer in channel c at time t

In a complex peer-assisted streaming system, the time series of different quantities are interdependent. Notations for frequently used quantities are listed in Table 1. Fig. 2 illustrates our model for representing such multi-level dependencies, which can be classified into three modules, capturing application specific peer behavior, server behavior, as well as network behavior that links the former two. For the purpose of this paper, the server behavior module can be omitted. There are both causal relationships represented by directed edges, and non-causal correlations denoted by undirected edges.

The core of this learning framework is the network behavior module. It tries to learn a network response $p(\overline{\boldsymbol{r}}_t^c|\overline{s}_t^c)$ for each channel c, which yields a distribution of the average peer receiving rate $\overline{\boldsymbol{r}}_t^c$ given the server bandwidth supply per peer $\overline{s}_t^c$. Since

$$\overline{\boldsymbol{r}}_t^c = \overline{s}_t^c + \overline{\boldsymbol{u}}_t^c, \qquad (1)$$

we can obtain $p(\overline{\boldsymbol{u}}_t^c|\overline{s}_t^c)$ first, and then derive $p(\overline{\boldsymbol{r}}_t^c|\overline{s}_t^c)$. Note that the response $p(\overline{\boldsymbol{r}}_t^c|\overline{s}_t^c)$ is non-linear in nature and can exhibit very complex behavior under different network conditions and demand patterns.

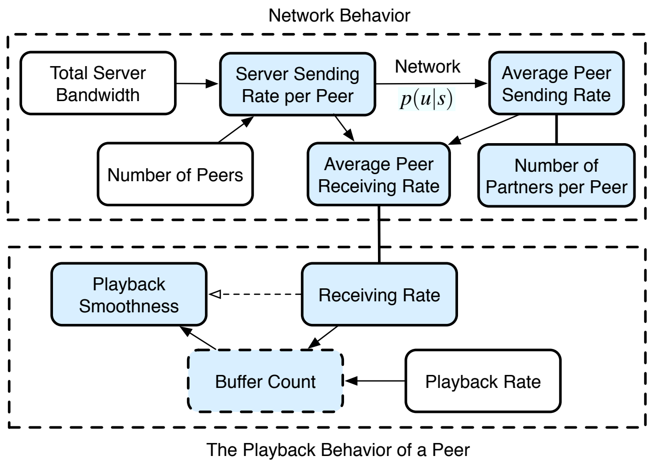
Network Behavior
Total Server Bandwidth
Server Sending Rate per Peer
Network
$p(u|s)$
Average Peer Sending Rate
Number of Peers
Average Peer Receiving Rate
Number of Partners per Peer
Playback Smoothness
Receiving Rate
Buffer Count
Playback Rate
The Playback Behavior of a Peer

Figure 2: The dependencies in a streaming channel. White boxes denote easily observable parameters. Shadowed boxes are quantities to be inferred. A directed edge denotes a causal relationship. An undirected edge denotes a non-causal correlation.

In reality, some of the easily obtainable quantities such as the peer population, the server bandwidth allocated and the number of partners per peer can be periodically monitored, forming the observation time series. The observed time series can then be used to predict performance related quantities such as peer receiving rates and the playback quality, based on learned dependencies. The learned model can also direct server and peer algorithms to improve performance and save cost. In the following section, we analyze real-world traces from UUSee Inc. to see how our learning framework can help the system to diagnose itself and make better decisions in a wide range of settings, in the context of live streaming. However,

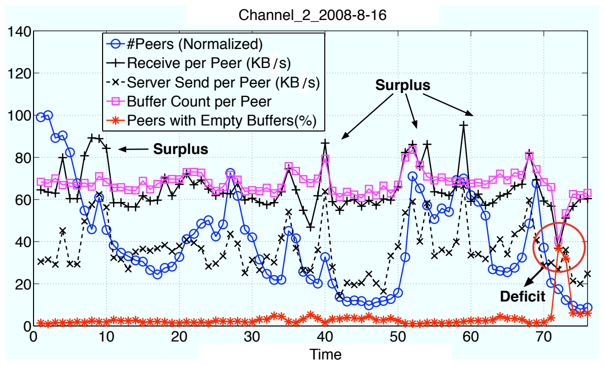

Figure 3: Traces from a live channel on Aug 16, 2008. The per peer quantities are averaged over the statistics from all peers in the channel at the same time.

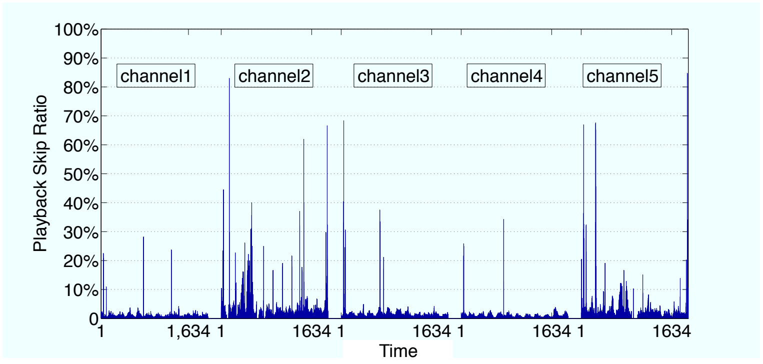

Figure 4: The playback skip ratio during the period Aug 8-19, 2008 for 5 popular channels. The period includes 1634 timestamps for each channel.

such a framework is not limited to live streaming, and can be easily generalized to other peer-assisted content distribution systems such as on-demand streaming and file sharing.

3. ANALYZING REAL-WORLD TRACES

We now seek to uncover performance issues that exist in the current UUSee live streaming system, and shed lights on how these problems can be tackled based on our learning framework. UUSee is one of the leading P2P multimedia solution providers in China, featuring online broadcasting rights to the 2008 Beijing Olympics. It simultaneously broadcasts over 800 live streaming channels, mostly encoded to video streams with a bit rate of around 500 kbps, to millions of users distributed across over 40 countries in the world.

Similar to most commercial P2P streaming protocols, UUSee's live streaming protocol allows each peer to download media blocks that will be played in the immediate future from servers or from other peers, and to cache them in its playback buffer. A media block will be removed once it is played. Once a new peer joins a channel in UUSee, an initial set of partners (up to 50) is supplied by one of its tracking servers. Each peer requests media blocks from its partners based on the periodically exchanged buffer maps. UUSee incorporates a number of algorithms in optimizing peer selection, so that a peer with better quality and higher uploading bandwidth will serve more partners, and a peer that experiences low streaming quality can identify better partners to request from.

To inspect the run-time behavior of UUSee P2P streaming, we have implemented detailed measurement and reporting capabilities within each UUSee client software. Each peer collects a set of its vital statistics, encapsulates them into "heartbeat" reports, and sends them to dedicated logging servers every 10 minutes via UDP. The statistics include its IP address, the channel it is watching, its buffer availability map, its buffer count (the number of blocks in its current buffer), as well as a list of all its partners, with their corresponding IP addresses, and current sending/receiving rate to/from each partner.

Our study is based on the measurements collected from 5 popular UUSee live channels in a 12-day period during 2008 Summer Olympics, from 14:51:58, Friday, August 8, 2008 (GMT+8) to 23:43:56, Tuesday, August 19, 2008 (GMT+8). We believe this set of traces have captured the characteristics of UUSee in a variety of typical scenarios, including large flash crowds gathered to watch popular Olympic games.

3.1 Inefficient Server Resource Usage

The first apparent problem of the current UUSee streaming system is that it is not allocating server bandwidth to channels with any awareness of performance. This leads to an inefficient and unfair allocation of server capacity across channels, incurs high server cost, and undermines the overall system performance. Analyzing UUSee traces, we find P2P streaming systems can operate in 3 different states, namely *deficit*, *normal*, and *surplus* states. In the deficit state, the total bandwidth supply is insufficient to satisfy the demand, leading to network-wise performance issues. In the surplus state, the total demand is satisfied, however, with resources excessively allocated. The normal state leads to a rough balance between supply and demand, and is the healthy state for the system to operate in.

For example, Fig. 3 shows the average statistics over a period of half a day in a typical live channel on Aug 16, 2008. Intuitively, to guarantee smooth playback, each peer only needs to receive media blocks at a rate that is slightly higher than the required playback rate (around 500 kbps). However, the figure shows that at certain times, the average peer receiving rate reaches 90 KB/s (720 kbps) or even 100 KB/s (800 kbps), which exceeds the playback rate by a substantial margin. At each point of the surplus receiving rate, we have also observed an excessively high server sending rate per peer. This implies that the existence of surplus states is due to a surplus allocation of server bandwidth when peers already have good upload abilities themselves. Nevertheless, when peer upload abilities degrade, the allocated server bandwidth does not increase to compensate the deficit bandwidth supply, leading to network-wise performance degradation in the channel. An example of such deficit states can also be observed in Fig. 3.

To optimize performance and save server cost, it is desirable to run the system in its normal state. To prevent surplus states and save server cost, we can let the system learn the network response $p(\bar{\mathbf{r}}_t^c|\bar{s}_t^c)$ in its normal state for each channel c. We can then find an appropriate rate cap on the server bandwidth allocated to channel c, by increasing $\bar{s}_t^c$ until the resulted $\bar{r}_t^c$ exceeds the playback rate by a certain margin. Furthermore, to improve the system overall performance when multiple channels co-exist, the limited amount of server capacity should be optimally allocated to different channels so as to optimize the aggregate receiving rate in all channels. In Sec. 4, we will propose a learning mechanism that allows the system to learn the network response in its normal state, given the collected noisy data with all possible states. We will address the problem of learning-based server provisioning in Sec. 5. We show that with such performance-aware server capacity provisioning, the server cost can be greatly reduced and the overall performance of all channels can be enhanced.

3.2 Performance Anomalies

Another problem of the current UUSee system is that, although the system normally performs well, there exist performance anoma-

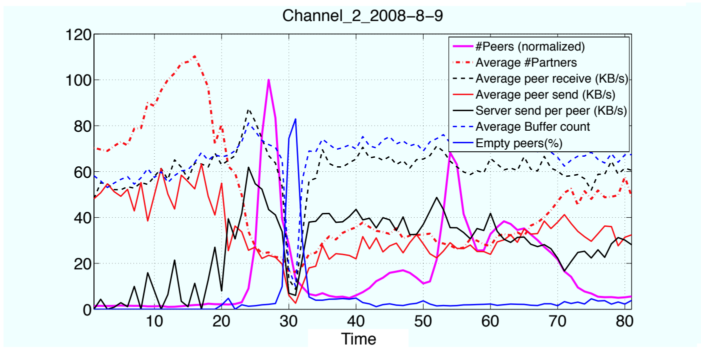

Figure 5: Traces from a live channel on Aug 9, 2008. An anomaly happens around time 30 after the flash crowd. The playback skip ratio equals to the percentage of empty peers in the channel.

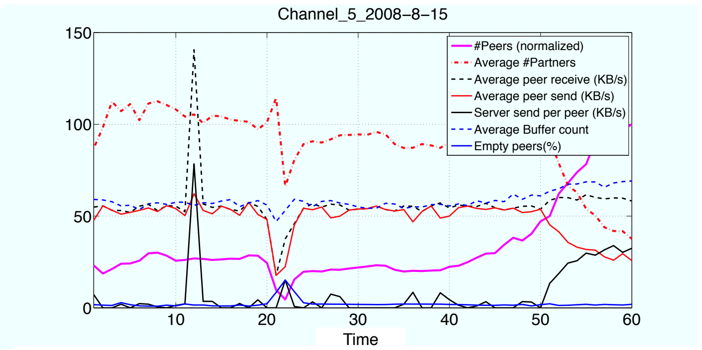

Figure 6: Traces from a live channel on Aug 15, 2008. An anomaly happens around time 22 due to peer departures. The playback skip ratio equals to the percentage of empty peers in the channel.

lies, at which points a channel will suffer from substantial performance degradation. We define a *performance anomaly* as a point in time when the channel has an extremely low average peer receiving rate, and an abnormally high playback skip ratio. We observe that such anomalies usually happen in those popular channels where peers are highly dynamic and flash crowds happen frequently.

Table 2: Anomaly ratios in the 12-day period.

Channels	1	2	3	4	5
Peak Population	33,431	29,926	88,810	56,979	26,630
Lowest Population	231	3	1,006	1,069	19
Average Daily Peak	17,248	7,250	37,624	29,392	9,868
Average Daily Lowest	748	46	2,333	1,490	90
Anomaly Ratio	0.43%	6.73%	0.61%	0.37%	2.63%

We define the *playback skip ratio* in a channel at a particular time as the percentage of peers in the channel that can not play the video smoothly at this time. Since the UUSee live streaming protocol ensures that blocks closest to their playback deadlines are downloaded first, a peer can playback smoothly as long as there is at least one media block in its buffer. Thus, the playback skip ratio equals to the percentage of peers with empty buffers in the channel. The playback skip ratio sampled at each timestamp (with sampling interval being 10 minutes in the traces) forms a coarse-grained indicator of the channel quality. We further define the *anomaly ratio* of a channel as the percentage of timestamps in the 12-day period that exhibit a playback skip ratio greater than 10%.

The playback skip ratios for the 5 representative channels are plotted in Fig. 4. Although most of the time the playback skip ratios are below 5%, anomalies do exist. During anomalies, the playback skip ratio can increase up to over 80%. The anomaly ratios of these 5 channels together with their population statistics are listed in Table 2. We can see more anomalies happen in channels 2 and 5 than in the other 3 channels. Table 2 actually reveals a correlation between the anomaly ratio and the population evolution in the channel. Although all 5 channels attract large peak and daily peak populations, channels 2 and 5 have much smaller lowest and daily lowest populations. This leads to the conjecture that, it is not the popularity of the channel, but the drastic change of the number of online peers between large and small values, that renders the system unstable.

In particular, we have identified two types of anomalies. The first type happens after a flash crowd, and is illustrated in Fig. 5. A serious network-wide performance degradation happens in the channel between time 28 and 33, after the actual flash crowd around time 25-30. The second type of anomalies are caused by the simultaneous departures of a large number of peers, as illustrated in Fig. 6. Around time 20-24, a large number of peers leaves the channel and triggers an anomaly. Another example of this type of anomalies has been shown in Fig. 3 around the deficit point. Although peer dynamics constitute the root causes for performance anomalies, they do not necessarily lead to anomalies. For example, in Fig. 5, there is another flash crowd around time 55. But the performance remains in its normal state. Similar flash crowds also frequently happen in channels 1, 3 and 4, but most of them do not lead to anomalies. This implies that it is almost impossible to predict anomalies only based on the change of peer population.

In Sec. 6, we will develop a neural network approach that can predict sharp decreases in peer receiving rates and increases in the playback skip ratio based on network monitoring. Such monitoring is lightweight and only requires the recent measurements of peer population N_t^c, the server rate per peer $\bar{s}_t^c$, which can be easily obtained from the tracking server, and the average number of partners per peer $\overline{P}_t^c$, which can also be accurately reported. Referring to Fig. 2, the predictor actually forecasts the transition of network behavior into deficit states and sends alarms to the system administrator, who will take prevention actions accordingly.

4. LEARNING NETWORK BEHAVIOR IN NORMAL STATES

Learning the network behavior enables performance estimation given a certain server bandwidth supply, which can be used to direct efficient and optimized server provisioning across concurrent channels. As has been mentioned in Sec. 3.1, the network can behave in 3 different states. We use $\mathcal{D}$ to denote the set of deficit states, $\mathcal{N}$ to denote normal states, and $\mathcal{S}$ to denote surplus states. As we are dealing with individual channels in this section, we omit the superscript c in the notations $\bar{\boldsymbol{r}}_t^c$, $\bar{\boldsymbol{u}}_t^c$ and $\bar{s}_t^c$.

As the network response $p(\bar{\boldsymbol{r}}_t|\bar{s}_t)$ can exhibit complex behavior under different states, given the collected real-world traces, the difficulty lies in automatically learning the network response patterns in different states, while classifying these states in the meantime. We plot the datasets $\{(\bar{s}_t, \bar{u}_t)\}$ and $\{(\bar{s}_t, \bar{r}_t)\}$ collected from channel 2 during the period Aug 8-19, 2008, on 2-D planes in Fig. 7 and Fig. 8, respectively. First, note that classification by setting hard thresholds for the server sending rate or the peer receiving rate can hardly separate the states. As the average peer receiving rate in the normal state can fluctuate around the required playback rate in a certain range, as shown in Fig. 3 and Fig. 5, it is hard to determine this range exactly. In other words, it is hard to know how low/high a receiving rate should be, for it to be labeled as deficit/surplus. Even if a high playback skip ratio can help indicate a deficit state, it is still uncertain how high a receiving rate should be so as to be labeled as surplus instead of normal.

To avoid the dilemma of setting classification thresholds, we pro-

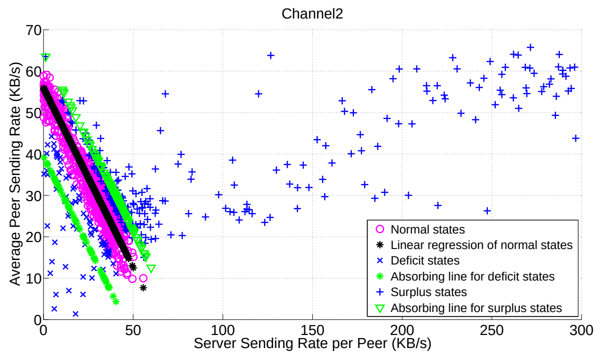

Figure 7: States clustering based on $\{(\bar{s}_t, \bar{u}_t)\}$. The iterative clustering procedure is only applied to the data points with $\bar{s}_t \le 80KB/s$. Data points with $\bar{s}_t > 80KB/s$ are automatically considered as surplus states.

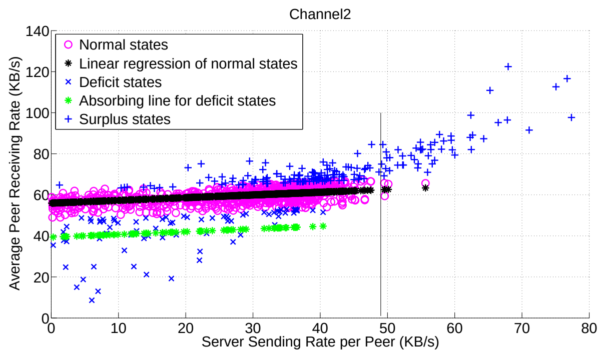

Figure 8: Clustering based on $\{(\bar{s}_t, \bar{r}_t)\}$. The vertical line is used to set an upper limit on the server rate that may be allocated to the channel. Data points with $\bar{s}_t > 80KB/s$ are automatically considered as surplus states.

pose a novel clustering procedure that automatically classifies the data $\{(\bar{s}_t, \bar{r}_t)\}$ collected from a channel c into three classes, deficit $\mathcal{D}$, normal $\mathcal{N}$, and surplus $\mathcal{S}$, and learns the mean response $f^c(\cdot)$ of the normal state:

$$f^c(\bar{s}_t) := \mathbf{E}(\bar{\boldsymbol{r}}_t | \bar{s}_t, (\bar{s}_t, \bar{\boldsymbol{r}}_t) \in \mathcal{N}). \quad (2)$$

The extraction is based on a simple fact that in the normal state of a channel, *increasing the server sending rate per peer will decrease the average peer sending rate.* The underlying reason is that under a pull-based P2P streaming protocol [6], a peer will stop requesting data from its partner peers if it already receives sufficient server bandwidth to support its playback rate.

This complementary property between server and peer sending rates can be observed from Fig. 5 and Fig. 6. When the network operates in the normal state, the sum of server sending and peer sending rate per peer, which is the average peer receiving rate, always fluctuates around the playback rate. However, the complementary property does not hold in other states. In the deficit state, both server and peer sending rates are low, as shown in Fig. 5 around timestamp 30 and in Fig. 6 around timestamp 23. In the surplus state, the server bandwidth allocated to the channel is more than sufficient, as shown in Fig. 5 around timestamp 24, where the average peer receiving rate almost reaches 90 KB/s (720 kbps).

The proposed extracting-learning algorithm, as shown in Table 3, is a novel combination of K-means clustering and polynomial curve fitting [2]. It involves performing regression and clustering alternately in an iterative way. First, we do the initial classification on $\{(\bar{s}_t, \bar{u}_t)\}$; if the server sending rate per peer is less than 20 KB/s, the state is a deficit state, whereas if it is greater than 60 KB/s, the state is a surplus state, otherwise, it is a normal state. However, now the normal set includes those states with a high peer sending rate. These states should be labeled as surplus instead, as the peer sending rate is sufficiently high and the server rate should be further reduced to save cost. Similarly, the deficit set includes states with low server rates but high peer sending rates. These states should belong to the normal set instead.

Next, the iterative regression and clustering procedure is performed to refine the classification and to learn $\mathbf{E}(\bar{\boldsymbol{u}}_t|\bar{s}_t)$ for the normal set. We first fit the current normal set $\mathcal{N}$ to a polynomial $y(\bar{s}_t, \vec{w})$ that best describes all $(\bar{s}_t, \bar{u}_t) \in \mathcal{N}$ in a least squares sense. We then find two other polynomials y' and y'' that are parallel to y and pass the means of the current deficit set $\mathcal{D}$ and surplus set $\mathcal{S}$, respectively. We call y' and y'' absorbing polynomials, since they are used to absorb deficit and surplus states. We subsequently assign each point in the plane to one of the 3 sets by comparing its distances to the 3 polynomials y, y' and y'' obtained in the regression. After the clustering is done, a new iteration of regression-clustering process starts.

Table 3: Iterative Extracting-Learning Algorithm

```
1. Initial Clustering. Let $\bar{r}_t = \bar{s}_t + \bar{u}_t$. If $\bar{r}_t < \alpha$, $(\bar{s}_t, \bar{u}_t) \in \mathcal{D}$.
   If $\bar{r}_t > \beta$, $(\bar{s}_t, \bar{u}_t) \in \mathcal{S}$. Otherwise, $(\bar{s}_t, \bar{u}_t) \in \mathcal{N}$.
2. Linear Regression.
   (a) Regression for Normal States. Let $y(\bar{s}_t, \vec{w}) = w_0 + \sum_{j=1}^{M} w_j \bar{s}_t^j$ be a polynomial of degree $M$, and $\epsilon$ be a
       zero mean noise term. Assume $\bar{u}_t = y(\bar{s}_t, \vec{w}) + \epsilon$, $\forall (\bar{s}_t, \bar{u}_t) \in \mathcal{N}$. Fit $y(\bar{s}_t, \vec{w})$ to all $(\bar{s}_t, \bar{u}_t) \in \mathcal{N}$ using
       the least squares approach. Update $\vec{w}$.
   (b) Absorbing Deficit and Surplus States. Find two other polynomials $y'$ and $y''$ that are parallel to $y$ such
       that $y'(\bar{s}_t, \vec{w}') = w'_0 + \sum_{j=1}^{M} w_j \bar{s}_t^j$ passes the mean of all $(\bar{s}_t, \bar{u}_t) \in \mathcal{D}$, and $y''(\bar{s}_t, \vec{w}'') = w''_0 + \sum_{j=1}^{M} w_j \bar{s}_t^j$ passes
       the mean of all $(\bar{s}_t, \bar{u}_t) \in \mathcal{S}$. Update $w'_0$ and $w''_0$.
3. Clustering. For each $(\bar{s}_t, \bar{u}_t)$, compare the error terms $|\bar{u}_t - y|$, $|\bar{u}_t - y'|$ and $|\bar{u}_t - y''|$. $(\bar{s}_t, \bar{u}_t)$ is assigned to the
   corresponding cluster with the smallest error term. Return to Step 2.
```

We find that polynomials of degree $M = 1$, i.e., lines, suffice to yield good regression here. When $M = 1$, the learned $f^c(\cdot)$ after the algorithm terminates is

$$f^c(\bar{s}_t) = w_0 + (w_1 + 1)\bar{s}_t. \quad (3)$$

The algorithm convergence is fast; 4 iterations suffice to provide the stable classification in Fig. 7. The algorithm can be applied to the dataset $\{(\bar{s}_t, \bar{r}_t)\}$ in a similar way.

From Fig. 8, we see that in normal states, no matter how much server bandwidth is allocated, a peer's receiving rate will be roughly around the playback rate. However, an interesting phenomenon is that, as we increase the server sending rate per peer, there is an increase, although slightly, in the average peer receiving rate. This is because a server can serve data blocks to more positions in a peer's buffer than another peer peer can, because data availability is higher in a server in general. The learned relationship $f^c(\cdot)$ between the server supply and peer receiving rate in each channel c will be used to direct server capacity provisioning across channels.

5. SERVER PROVISIONING

The automated learning of network behavior forms a basis for improving server and peer algorithms. Here, we illustrate the idea with one important application: *multi-channel server capacity provisioning* in the UUSee streaming system. We train the network response

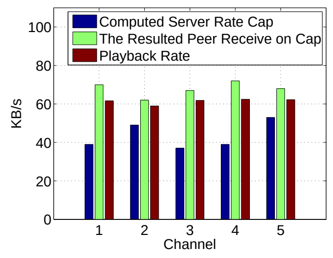

Figure 9: The computed caps on the server bandwidth allocation to 5 channels and the resulted average peer receiving rates.

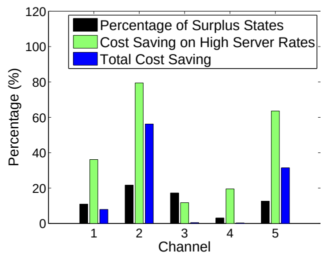

Figure 10: The server cost saved from server rate limiting in 5 channels.

$\bar{r}_t^c = f^c(\bar{s}_t^c)$ using data from the first 6 days for each of the 5 channels. The proposed server provisioning algorithms are then applied to the rest of the 12 days based on the trained model. We try to see how much saving on server cost or enhancement on performance the learning-based algorithms can bring to the UUSee system in this period.

5.1 Server Rate Limiting

The upshot of peer-assisted streaming is to maximally utilize peer resources for uploading. The server bandwidth should be allocated sparingly only to result in a receiving rate that sustains smooth playback at each peer. However, from Fig. 8 we observe that the server bandwidth allocated often exceeds the amount required by the channel. To prevent surplus allocation and resource waste, we can set a cap on server rate so that the per peer server bandwidth allocated to channel $c \in \mathcal{C}$, $\bar{s}_t^c$, cannot exceed R^c. Note that no heuristics can easily determine the value of such a server rate cap. However, such a task becomes feasible with the understanding of the relation between server sending and peer receiving rates.

To find the server rate cap R^c, we increase $\bar{s}_t^c$ until the probability that the channel operates in surplus states exceeds a threshold α. Fig. 8 illustrates the process of finding R^2 for channel 2. When $\alpha = 0.6$, the computed server rate caps for channels 1 – 5 are 39, 49, 37, 39, and 53 KB/s, respectively, as shown in Fig. 9. When the server rate allocated to each channel equals to its cap value, the resulted average peer receiving rates are 70, 62, 67, 72, and 68 KB/s, respectively, which are higher than the playback rates to guarantee the playback quality. Any further allocation of server bandwidth beyond the computed cap will lead to surplus allocation and incur a waste of resource.

Fig. 10 plots the savings on server cost from server rate limiting in the 5 channels as compared to the current UUSee system, assuming the cost is linear to the total bandwidth usage over the 6-day period for evaluation. We can see that there are significant savings at those timestamps when the server rate in UUSee exceeds the cap, while the savings on the total server cost in the period is also significant in certain channels, e.g., there is a total saving of 58% in channel 2.

5.2 Server Rate Allocation

Given a certain amount of server bandwidth U_t available at time t, it should be optimally allocated to a set of channels $\mathcal{C}$ so that the aggregate receiving bandwidth $\sum_{c\in\mathcal{C}} N_t^c \bar{r}_t^c$ is maximized. An increased receiving bandwidth at peers would improve their viewing experience or allow the streaming of media data with better quality and a higher playback rate. This requires the estimation of the total receiving bandwidth in each channel, given a certain amount of server bandwidth allocated. We assume that the server rate limiting mechanism is already applied to each channel to prevent surplus allocation. Associating each channel $c \in \mathcal{C}$ with a priority parameter p^c, the server bandwidth allocation problem can be formulated as

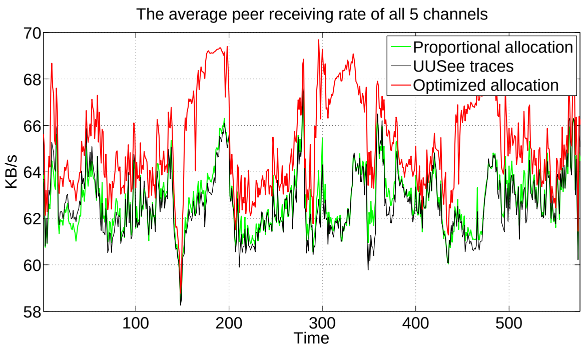

Figure 11: The average peer receiving rate across all 5 channels 1) in the UUSee traces, Aug 16-19, 2008, 2) under the proportional server rate allocation, and 3) under the optimized server rate allocation.

$$\text{Optimized Allocation:} \quad \max \sum_{c\in\mathcal{C}} p^c N_t^c \bar{r}_t^c \tag{4}$$

subject to

$$\begin{aligned} &\textstyle\sum_{c\in\mathcal{C}} \bar{s}_t^c N_t^c \le U_t, & \\ &\bar{r}_t^c = f^c(\bar{s}_t^c), & \forall c \in \mathcal{C}, \\ &0 \le \bar{s}_t^c \le R^c, & \forall c \in \mathcal{C}, \end{aligned}$$

where the time-invariant function $f^c(\cdot)$ is the $\bar{s}_t^c$ — $\bar{r}_t^c$ relationship in normal states learned by the extracting-learning algorithm in Table 3 based on recently collected statistics. When $\bar{r}_t^c = f^c(\bar{s}_t^c)$ is fitted by a line, problem (4) is a linear program where the optimal $\bar{s}_t^c$ can be computed efficiently.

To evaluate the benefit of the learning based optimized server provisioning, we compare its performance with the original UUSee traces and the *proportional allocation*. In proportional allocation, the amount of server bandwidth allocated to channel c is proportional to N_t^c, i.e.,

$$\text{Proportional Allocation:} \quad \bar{s}_t^c = \frac{U_t}{\sum_{c\in\mathcal{C}} N_t^c}, \quad \forall c \in \mathcal{C}. \tag{5}$$

We simulate the system performance by replaying the traces in the period Aug 8-19, 2008 as in the original system. Assume collected statistics come into the database D on a daily basis. We pre-train $f^c(\cdot)$ and R_c using the data of first 6 days. As solving (4) is extremely fast, we perform server provisioning across the 5 channels for every timestamp t in the remaining 6 days. Since learning is also efficient, starting from the 6th day, we perform the extracting-learning algorithm every day to adjust $f^c(\cdot)$ and R_c based on the data available thus far. U_t is set to be the total available server bandwidth in the original traces of this 5 channels at time t. Assume all the channels are equally important, i.e., $p_c = 1$ for all $c \in \mathcal{C}$. For each computed server allocation value $\bar{s}_t^c$, we find the resulted $\bar{r}_t^c$ based on the traces by computing the conditional mean of $\bar{r}_\tau^c$, given that $(\bar{s}_\tau^c, \bar{r}_\tau^c) \in D$ and $\bar{s}_\tau^c = \bar{s}_t^c$. For example, for channel 2 in Fig. 8, this is equivalent to finding the mean of the corresponding $\bar{r}_t$ values under a certain value of $\bar{s}_t$.

In Fig. 11, we plot the resulted average peer receiving rate that is averaged across all the peers in all 5 channels in the last 4 days. We observe that the proportional server allocation is very close to the current UUSee system performance. This confirms that the current pull-based UUSee system passively allocates server bandwidth by simply responding to peers' requests, and thus, the more requests from peers in a channel, the more server bandwidth the channel obtains. In contrast, the learning based optimized server allocation has

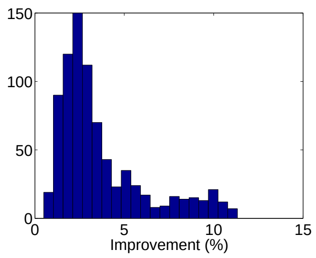

Figure 12: Histogram of the aggregate bandwidth improvements in all 5 channels at different times due to optimized server provisioning.

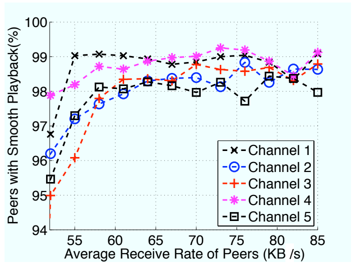

Figure 13: The ratio of peers with smooth playback in a channel as the average peer receiving rate varies.

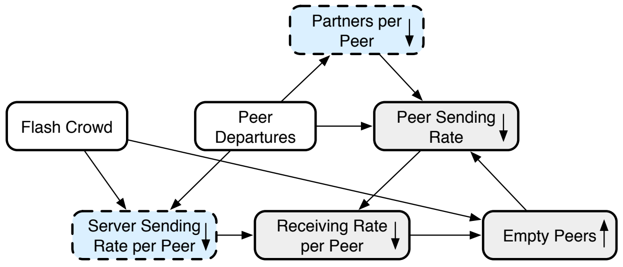

Figure 14: The root causes of performance degradation. While flash crowds and peer departures are the root causes of anomalies, the server sending rate and average number of partners per peer can help predict anomalies.

a salient benefit. This indicates the insufficiency of the passively responding mechanism of the UUSee servers, and justifies the necessity of understanding the network behavior, especially the relationship between server supply and the aggregate bandwidth achieved in each channel, when doing server provisioning.

Fig. 12 plots the histogram of the aggregate bandwidth improvements in the system at different times due to optimized allocation. We can see that if the performance-aware server provisioning is applied, there is still a significant portion of time at which the system aggregate receiving bandwidth could be increased by over 10%. Since only 5 channels are considered in our experiments, we conjecture that a real system with thousands of channels could improve from performance-aware server provisioning by a much larger margin. With improved server efficiency, videos of higher bitrates can be streamed to users without upgrading the server capacity.

Alternatively, we could link the server allocation $\overline{s}_t^c$ to other performance metrics, e.g., the channel playback quality $\overline{q}_t^c$. This can be done by statistically learning the peer playback behavior combined with the learned network behavior $\overline{r}_t^c = f_c(\overline{s}_t^c)$. For example, Fig. 13 shows that there is an approximate concave relationship between $\overline{r}_t^c$ and $\overline{q}_t^c$ under a given channel playback rate, and thus a similar concave relationship between $\overline{s}_t^c$ and $\overline{q}_t^c$. We can thus formulate a convex optimization problem that finds the optimal server rates allocated to different channels so as to maximize the aggregate quality. Other quality metrics include PSNR, which is an objective video quality evaluation and can be characterized as a concave function of bitrate for different types of videos [3].

6. ANOMALY PREDICTION

Flash crowd joins and large-scale simultaneous peer departures are common events in P2P systems. In these scenarios, the system behavior often becomes unstable and is prone to move into deficit states with both insufficient server and peer sending rates, causing performance anomalies. In this section, we propose a neural network based forecast mechanism that predicts performance anomalies based on lightweight observations from the network with manageable false positives. Currently, such anomaly prediction heavily relies on human expert knowledge. Network administrators can troubleshoot performance issues more accurately with the aid of the predictor and take prevention actions accordingly.

6.1 Root Causes of Anomalies

First, it is important to note that in the normal state of the network, a drop of either server sending rate or peer sending rate will not cause performance degradation. The decrease in one quantity is often because there is an increase in the other, and the network has a automatic mechanism to maintain the average peer receiving rate around the channel playback rate. This has been observed from Fig. 5 and Fig. 6 when the network is in its normal state. During a performance anomaly, however, the *complementary relationship* between server sending and peer sending rates no longer holds. In this case, the system may move into the deficit state, where the total bandwidth supply begins to drop quickly.

As shown in Fig. 14, the root causes of large-scale performance degradation is due to flash crowds or peer departures. Both kinds of peer dynamics can cause the average server sending rate that each peer receives to change. This is because in a pull-based streaming system, servers respond to requests passively and thus there is no secure match between the server rate allocated to a channel and the number of peers in this channel. When the number of peers changes drastically, the average server rate each peer receives can easily drop. If the server sending rate per peer decreases, the average peer sending rate will not ramp up as in normal states due to the complementary property, since the newly joined peers do not have the ability to serve yet. As a result, the receiving rate per peer will decrease, resulting in a higher number of peers with low buffer counts. The increase of empty peers will undermine the peer sending ability which in turn hurts peer receiving rates. Thus, a cascaded degradation of a series of performance metrics is triggered.

Such a process is typically observed in Fig. 5 around time 25-35. A flash crowd join happens around time 25. Since the new peers do not have the ability to serve yet, most peers have to request data from the server, and the server contribution soon becomes the dominant portion in each peer's receiving rate. However, at time 27, many peers start to leave the channel. The departure of these peers, which may be receiving high server sending rates, leads to a drop in the average server rate received per peer. As the recently joined peers still have low sending ability and the passively responding servers do not have a mechanism to compensate bandwidth proactively, the channel is trapped into a cascaded performance degradation.

Large-scale peer departures can trigger another series of detrimental events. Even if the server sending rate per peer is stable, simultaneous departures of a large number of peers will cause the remaining peers to lose their existing partners, incurring a delay for partner rediscovery. Until the topology is reformed, the average peer sending ability is undermined, which triggers the cascaded degradation as shown in Fig. 14. An example of this phenomenon is illustrated in Fig. 6. We can see the server rate remains low most of the time in this channel, as the channel mainly relies on peers' contribution for data uploading. However, a large number of peers leave the channel around time 20, which causes a sharp decrease in the number of partners per peer and the peer sending rate. At this point, since the server only passively responds to requests, it does not compensate bandwidth proactively, and thus the cascaded degradation happens.

6.2 Prediction using Neural Networks

Our anomaly prediction is performed separately in each channel at each timestamp t. The target $\vec{\boldsymbol{T}}_t$ to be predicted is a 2-D vector. Let $\vec{\boldsymbol{T}}_t = (0, 1)$ if a performance anomaly happens at time t, and $\vec{\boldsymbol{T}}_t = (1, 0)$ otherwise. We set $\vec{\boldsymbol{T}}_t = (0, 1)$ if and only if the decrease in average receiving rate $\overline{r}_t$ exceeds a threshold β_r and at the same time the ratio of empty peers exceeds β_e. Otherwise, we set $\vec{\boldsymbol{T}}_t = (1, 0)$ to denote a normal state.

Since the average peer receiving rate $\overline{r}_t$ is the sum of the average server rate per peer $\overline{s}_t$ and average peer sending rate $\overline{u}_t$, we can predict the decrease of $\overline{r}_t$ by monitoring the past trend of $\overline{s}_t, \overline{s}_{t-1}, \ldots$ and $\overline{u}_t, \overline{u}_{t-1}, \ldots$. The average server sending rate per peer can be easily calculated from $\overline{s}_t = \overline{S}_t/\overline{N}_t$, where the total server rate allocated to the channel $\overline{S}_t$ and the online number of peers $\overline{N}_t$ are readily available at a centralized session tracker.

Nevertheless, obtaining accurate peer sending rates is hard, since it requires each peer to closely monitor its outgoing bandwidth and report it accurately. To keep observation collection lightweight, we monitor the average number of partners per peer $\overline{P}_t$ instead of peer sending rates. Using $\overline{P}_t$ is an even better choice than using $\overline{u}_t$ in prediction because of several reasons. First, there is a close relationship between the peer sending ability and the number of partners per peer, as shown in Fig. 5 and Fig. 6. This is because in a pull-based system, a peer with better sending ability will be requested by more other peers and thus involved in more connections. Second, A small average number of partners per peer could indicate the illness of the system, such as the occurrence of large-scale peer departures and topology destruction. In addition, the number of partners of a peer changes much more slowly with less variation, and thus has more "memory" of the network behavior, which can be used to predict relatively further future. Each peer can also easily report its exact number of partners to the predictor online, where the average number of partners per peer $\overline{P}_t$ is calculated accurately. Therefore, to predict performance anomalies, we only require a lightweight observation of $\overline{P}_t$ and $\overline{s}_t = \overline{S}_t/\overline{N}_t$ sampled at recent timestamps.

The input into the predictor is thus an L-dimensional vector

$$\vec{\boldsymbol{x}}_t = (x_1, \ldots, x_L) = (\overline{s}_{t-L_1+1}, \ldots, \overline{s}_t, \overline{P}_{t-L_2+1}, \ldots, \overline{P}_t)/100, \quad (6)$$

which is composed of the most recent L_1 measurements of per peer server sending rate normalized by 100, and L_2 measurements of the average number of partners per peer normalized by 100. Our goal is to predict the target $\vec{\boldsymbol{T}}_t$ given an observation vector $\vec{\boldsymbol{x}}_t$ at time t, based on the supervised learning from collected data. Many well-developed machine learning tools, such as support vector machines (SVMs) and neural networks (NNs) [2], can efficiently solve this problem. Here we use a simple 2-layer neural network for prediction.

The neural network is uniquely identified by its weights $\vec{\boldsymbol{w}}$ in its 2 layers. It is essentially a nonlinear function that given an L-dimensional input $\vec{\boldsymbol{x}}_t = (x_1, \ldots, x_L)$, outputs the probability $Pr_E(t)$ that an anomaly happens at time t. In other words, the weights $\vec{\boldsymbol{w}}$ encode the relation between the input and output vectors, which in our case is the relation between the observed $(\overline{s}_{t-L_1+1}, \ldots, \overline{s}_t, \overline{P}_{t-L_2+1}, \ldots, \overline{P}_t)$ and the possibility of a performance anomaly at time t.

Fig. 15 illustrates the prediction process of our 2-layer neural network. It first transforms the input $\vec{\boldsymbol{x}}_t = (x_1, \ldots, x_L)$ into a layer of M hidden units $(z_1, \ldots, z_M)$, and then transforms the hidden units to an output vector $\vec{\boldsymbol{y}} = (y_1, y_2)$ as follows:

$$\begin{cases} z_j = \sum_{i=1}^{L} w_{ji}^{(1)} x_i + w_{jo}^{(1)}, & j = 1, \ldots, M, \\ y_k = \sum_{j=1}^{M} w_{kj}^{(2)} h(z_j) + w_{ko}^{(2)}, & k = 1, 2, \end{cases}$$

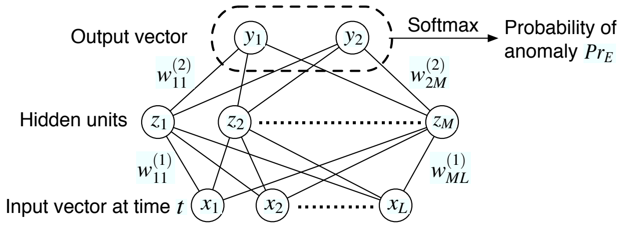

Figure 15: Anomaly prediction using the 2-layer neural network. Given the input observations $\vec{x}_t$ at time t, the neural network will output the probability $Pr_E(t)$ that an anomaly happens.

where $[w_{ji}^{(1)}]$ and $[w_{kj}^{(2)}]$ are the weights for linear transformation from $\vec{\boldsymbol{x}}$ to the M hidden units and from the hidden units to the output $\vec{\boldsymbol{y}} = (y_1, y_2)$, respectively. $h(x) = 1/(1 + e^{-x})$ is the logistic sigmoid function applied to the hidden units to introduce nonlinearity into the transformation. The neural network finally outputs the probability that an anomaly happens at time t, using softmax outputs:

$$Pr_E(t) = \Pr(\vec{\boldsymbol{T}}_t = (0, 1)|\vec{\boldsymbol{x}}_t) = \frac{e^{y_2}}{e^{y_1} + e^{y_2}}. \quad (7)$$

The weights $[w_{ji}^{(1)}]$, $[w_{kj}^{(2)}]$ of the neural network can be trained using the error backpropagation algorithm [2], which efficiently finds the appropriate weights that minimize the prediction error in the training data.

6.3 Removing False Positives

In our experiments, we find that if the prediction is solely based on the past measurements of server rate per peer $\overline{s}_t$ and the average number of partners per peer $\overline{P}_t$, there will be a large number of false positives. Although the neural network will seldom fail to predict a true anomaly, it could generate many false alarms where the anomaly does not happen. To remove false positives, we make use of the statistics on the number of online peers. As has been shown in Fig. 14, the root causes of performance anomalies are due to peer dynamics and the anomalies only happen when the number of online peers changes drastically. Furthermore, we also observe from the traces that at the beginning of a flash crowd, when the number of peers start to increase, each peer can actually receive sufficient server rate, as shown in Fig. 5. It is only during the latter phase of the flash crowd, when the peers start to leave the network, that the server rate per peer will drop dramatically. Due to the above reasons, we only consider an anomaly alarm made by the neural network valid, if there is a sharp drop in the online peer population N_t at this time.

However, the question is how sharp a decrease in the peer population should be considered significant? Referring to Fig. 5 again, there is a dramatic drop of the number of online peers from 3487 at time 54 to 1305 at time 58, but no anomaly happens in this period. However, in Fig. 6, when the number of online peers changes from 208 at time 18 to 71 at time 21, the anomaly does happen. The reason is that a larger peer population is more tolerant to the topology destruction caused by peer departures. This motivates us to take the following consideration into account: for a larger peer population, there should be a larger decrease in the number of online peers for it to be considered significant enough.

Therefore, to remove false positives, we only consider an alarm at time t valid if the following selector is satisfied:

$$\max_{j=1,\ldots,\Delta t} (\log N_{t-j} - \log N_t) > \log \gamma \cdot \frac{N_t}{100}, \quad (8)$$

where Δt is a small integer between 1-10 and γ is an important sensitivity parameter. Alternatively, we can view the selector as a coarse-

Table 4: **Prediction Errors over 12 days in 5 channels.**

	NN Prediction					NN with FP removal				
Channels	1	2	3	4	5	1	2	3	4	5
# Anomalies	4	16	5	2	7	4	16	5	2	7
# FN	0	0	0	0	1	0	1	0	0	1
# Alarms	12	41	14	6	28	4	16	4	2	10
# FP	8	28	10	4	20	0	3	0	0	3

grained predictor, whose prediction is refined by the neural network prediction. Thus, γ should be set to a small value for the selector to be generous. To find the appropriate value for γ given the training data, γ is set to its maximum value so that the selector will not falsely remove any true positives (true anomalies) in the training set.

6.4 Performance Evaluation

In our evaluation, we assume an anomaly happens at time t, i.e., $\vec{T}_t = (0, 1)$, if the decrease in average peer receiving rate $\bar{r}_{t-1} - \bar{r}_t > \beta_r = 8KB/s$, and $\bar{r}_t < 55KB/s$, and the ratio of empty peers exceeds $\beta_e = 10\%$. We use $(\bar{s}_{t-1}, \bar{s}_t, \bar{P}_{t-4}, \ldots, \bar{P}_t)/100$ as the input observation, i.e., $L_1 = 2$, $L_2 = 5$. The number of hidden units in the neural network is set to be $M = L = L_1 + L_2 = 7$. We assume the server rate limiting scheme in Sec. 5.1 has been applied and excessive server sending rates are chopped to the server rate caps. As a result, the dataset only consists of channel average statistics in normal and deficit states.

For each channel, the data of one day with performance issues is chosen as the training set. The weights of the neural network are trained for 1000 iterations using error backpropagation [2], which only takes 40 seconds on a MacBook with 2.26GHz Intel Core2 Duo processor. This means that the neural network can be easily retrained as new data are collected. To determine a proper selector, we first set $\Delta t = 5$ in (8). For each channel, γ is set to its maximum value such that the selector will not falsely remove any true positives (true anomalies) in the training set of this channel. Note that we only train the neural network and the selector based on the data of one day, and test the prediction performance by applying the trained model to the entire 12-day period. This will test the ability of our prediction algorithm to generalize to larger data sets.

To optimize performance, the predictor decides there is a performance issue if $Pr_E > 0.9$ or the increase of Pr_E exceeds 0.15. We also smoothed out the input in both training and prediction by letting each of the L input values to be the average of this value and its corresponding previous measurement. We group the consecutive performance anomalies and also group the consecutive alarms made by the predictor. A sequence of consecutive $(0, 1)$ in the targets is deemed as one single performance anomaly. A sequence of consecutive $(0, 1)$ in the predicted targets is deemed as one single anomaly alarm. An alarm is correct, i.e., is a true positive, if and only if it overlaps with at least one performance anomaly.

The prediction errors of all 5 channels over the 12-day period from Aug 9-19, 2008 are shown in Table 4. We can see that the neural network (NN) approach has very few false negatives (FNs), which means it seldom falsely predicts an anomaly as normal, although it does incur many false positives (FPs). With FP removal by checking the change in the online peer population, the number of false positives are successfully reduced to a level that the network administrator can tolerate. For channel 2, where the most anomalies happen, the performance contrast between NN and NN with FP removal can be observed from Fig. 16 and Fig. 17, which plot the prediction performance on each day in this channel.

To take a closer look at how the predictor works, we plot the prediction results of channel 2 on Aug 9, 2008 for NN and NN with FP removal in Fig. 18 and Fig. 19, respectively. The data on Aug 19 is used to train the model. γ for the selector is computed as 1.8. We can see NN alone reports 6 alarms, among which 5 are false positives. For example, NN reports the 4th and 5th alarms forecasting anomalies because it observes a decrease in the server rate per peer, whereas the number of partners per peer does not increase (referring to Fig. 5, note the different time limits in Fig. 5). In NN with FP removal, however, the selector checks the decrease in the number of online peers at this time, e.g., from 3487 to 1305 at the 4th alarm. For $N_t = 1305$, such a decrease is not significant enough according to (8). Thus, the 4th alarm is identified as a false positive and removed. The 1st and 2nd alarms are removed as well, since there is no big change in the number of online peers around time of these false alarms.

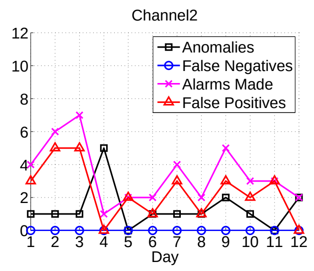

Figure 16: The neural network approach applied to channel 2 over 12 days. The model is trained using the data on Aug 19, 2008.

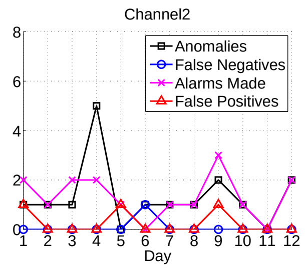

Figure 17: Neural network with false positive removing based on peer population statistics applied to channel 2 over 12 days. $\gamma = 1.8$.

7. RELATED WORK

Since mesh-based P2P media streaming has been proposed in CoolStreaming by Zhang *et al.* [13], such systems have been successfully deployed on the Internet. Significant research efforts [5, 6, 11] have been devoted to the measurements of their performance. This paper represents one of the first attempts to characterize the dependencies among the performance statistics inside P2P multimedia streaming systems. We have shown how the learned dependencies can be leveraged to optimize system operation and forecast performance issues due to irregular demand patterns.

Recently, there has been an increasing interest in applying inference, learning and statistical tools to diagnose and improve the reliability of large-scale networking systems. Bahl *et al.* [1] propose Sherlock that discovers the dependencies among services, software and physical components in a large enterprise network based on network traffic, and leverages such dependencies to detect and localize problems. Chen *et al.* [4] further study the performance and limitations of automated dependency discovery from network traffic, and propose Orion that discovers dependencies in an enterprise network, using the delay information between every pair of network flows. Mahimkar *et al.* [8] focus on characterizing performance issues in a large-scale IPTV network, and propose Giza that applies multi-resolution data analysis to localize regions and physical components in the IPTV distribution hierarchy that are experiencing performance problems.

Diagnosing and learning the operating rules of P2P multimedia streaming systems pose unique challenges to networking researchers. In contrast to traditional client-server applications in enterprise networks and IPTV systems, P2P networks are designed to be resilient by dynamically changing the set of hosts with which a peer com-

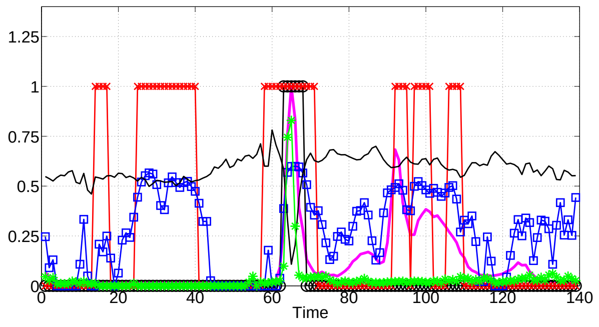

Figure 18: Prediction performance for Channel 2 on Aug 9, 2008, using the neural network alone. The legend is shown in Fig. 19. The predicted probability Pr_E is subtracted by 0.4 so that an anomaly is predicted if $Pr_E > 0.5$ or the increase in Pr_E exceeds 0.2.

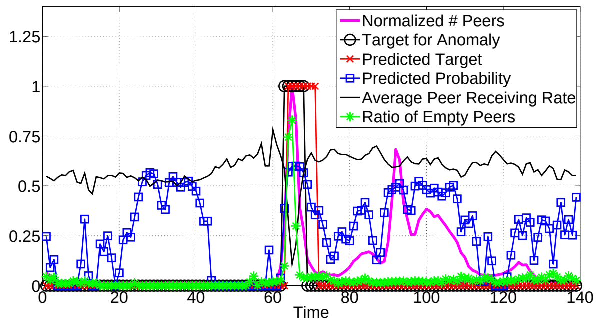

Figure 19: Prediction performance for Channel 2 on Aug 9, 2008, using the neural network combined with false positive removal based on population statistics. The false positives are removed at those points when population changes are not significant enough. $\gamma = 1.8$.

municates. As a result, the flow or service level dependencies of these applications can change from time to time. In this paper, we take a novel approach of aggregating the statistics at channel levels and characterizing the dependencies among the different time-series that are related to channel average performance. Instead of localizing the problems in a particular physical or software component, we use such learned dependencies to improve server and peer decisions, and predict the performance issues caused by collective events happening in the network such as flash crowds or simultaneous peer departures.

Peterson *et al.* [9] propose Antfarm, a BitTorrent-like P2P file sharing system based on managed swarms. When multiple swarms co-exist, Antfarm gathers information on swarm dynamics and directs bandwidth allocation to each peer so as to minimize the overall download latencies. Wu *et al.* [10] also propose a server capacity provisioning algorithm for P2P live streaming systems, which proactively adjusts the server bandwidth allocated to concurrent channels, based on the demand and performance prediction. In this paper, we also leverage the knowledge of system characteristics to aid efficient allocation of resources. However, our algorithm for learning channel operating rules have advanced the state of the art, as they focus on relationships between more detailed performance metrics in the system on a fine-grained time scale, e.g., within an interval of 10 minutes in the UUSee traces.

8. CONCLUDING REMARKS

In this paper, we propose a new learning framework that allows peer-assisted media streaming systems to discover the inherent relationships among demand, supply and various performance metrics during their operations, based on periodic statistics collection. We show how our learning framework can be used to aid performance-aware server capacity provisioning across concurrent streaming channels, and to forecast performance anomalies caused by peer dynamics such as flash crowds and large-scale peer departures. We demonstrate the effectiveness and benefits of the proposed algorithms using real-world traces collected from UUSee live streaming during the 2008 Beijing Olympics. An integrated framework of measurement collection, automated learning, performance prediction and decision making opens a promising area for future research, and paves the way for more diagnostics and intelligence in practical peer-assisted systems.

9. REFERENCES

[1] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services Via Inference of Multi-level Dependencies. In *Proc. of SIGCOMM'07*, Kyoto, Japan, 2007.

[2] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[3] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou. Utility Maximization in Peer-to-Peer Systems. In *Proc. of ACM SIGMETRICS '08*, Annapolis, Maryland, USA, June 2008.

[4] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions. In *Proc. of OSDI '08*, 2008.

[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, December 2007.

[6] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang. Inside the New Coolstreaming: Principles, Measurements and Performance Implications. In *Proc. of IEEE INFOCOM*, 2008.

[7] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming. In *Proc. of IEEE INFOCOM*, 2007.

[8] A. Mahimkar, Z. Ge, A. Shaikh, J. Wang, J. Yates, Y. Zhang, and Q. Zhao. Towards Automated Performance Diagnosis in a Large IPTV Network. In *Proc. of SIGCOMM'09*, Barcelona, Spain, 2009.

[9] R. S. Peterson and E. G. Sirer. AntFarm: Efficient Content Distribution with Managed Swarms. In *Proc. of 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*, April 2009.

[10] C. Wu, B. Li, and S. Zhao. Multi-Channel Live P2P Streaming: Refocusing on Servers. In *Proc. of IEEE INFOCOM '08*, Phoenix, Arizona, 2008.

[11] Y. Huang and Tom Z. J. Fu and Dah-Ming Chiu and John C.S. Lui and Cheng Huang. Challenges, Design and Analysis of a Large-scale P2P-VoD System. In *Proc. of SIGCOMM'08*, Seattle, Washington, USA, August 17-22 2008.

[12] M. Zhang, Y. Xiong, Q. Zhang, and S. Yang. A Peer-to-Peer Network for Live Media Streaming: Using a Push-Pull Approach. In *Proc. of ACM Multimedia 2005*, November 2005.

[13] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proc. of IEEE INFOCOM*, 2005.