EigenSpeed: Secure Peer-to-peer Bandwidth Evaluation

Robin Snader
rsnader2@cs.uiuc.edu
Dept. of Computer Science
University of Illinois at Urbana–Champaign

Nikita Borisov
nikita@uiuc.edu
Dept. of Electrical & Computer Engineering
University of Illinois at Urbana–Champaign

Abstract

Many peer-to-peer systems require a way of accurately assessing the bandwidth of their constituent peers. However, nodes cannot be trusted to accurately report their own capacity in the presence of incentives to the contrary and thus self-reporting can lead to poor network performance and security flaws. We present EigenSpeed, a secure and accurate peer-to-peer bandwidth evaluation system based on opportunistic bandwidth measurements, combined using principal component analysis. We test EigenSpeed using data gathered from a real peer-to-peer system and show that it is efficient and accurate. We also show that it is resistant to attacks by colluding groups of malicious nodes.

1 Introduction

In many peer-to-peer systems, nodes need estimates of the available bandwidth to each of its peers in the network. Such estimates are used for many purposes: to load balance routing in the network, or to select a peer from which to download a particular resource, or to enforce upload/download ratios, or even to simply measure network characteristics.

Although each peer is in the best position to determine its own bandwidth, it cannot be trusted to report accurate data when an incentive to lie exists, such as when a network preferentially allocates resources to nodes with higher bandwidths. If self-reported information is used to make security-sensitive decisions, dramatic reductions in overall security can result [2]. Thus, there is a need for a system enabling nodes to evaluate their peers' bandwidths that is resistant to manipulation by malicious nodes.

In this paper, we show how principal component analysis (PCA), a well-studied technique in other areas [8], can be applied to this problem. We show that unmodified PCA contains significant vulnerabilities to malicious nodes, and introduce modifications that exploit the specifics of the bandwidth estimation problem and resist attacks. We show that our final system, EigenSpeed, is secure, efficient, and accurate by testing it with bandwidth data collected from the Tor network [7].

2 Consensus Bandwidth Evaluation

We first define the specifics of the problem we are trying to address. Our goal is to come up with a consensus evaluation of the bandwidth of each peer in a peer-to-peer network. In a fully general network, each peer would need to maintain bandwidth information regarding each other peer. However, on the Internet, a bottleneck link on a connection between two hosts is likely to be close to one of the endpoints, rather than in the network core [1, 10]. This means that we can treat bandwidth as a *node*, rather than *link*, property and still obtain useful results. This view will allow nodes to pool their observations of a particular node and obtain more accurate and globally useful bandwidth assessment. A consensus view is also helpful to ensure that all nodes maintain a similar behavior, preventing partitioning attacks that draw selected nodes towards a particular behavior for further exploitation.

One naïve method for a node to determine the bandwidth of its peers is simply to track the performance of each as it communicates with them. However, this approach is far from ideal: first of all, it will take a long time for a node to gather accurate information about all of its peers, and decisions made before that time will be far from optimal. Additionally, this scheme is vulnerable to partition attacks: a malicious node that wishes to attract the traffic of one of its peers can reserve all or most of its bandwidth for communication with that peer. The target node will then have a greatly skewed view of the network and preferentially choose the malicious node over honest peers. Using this technique, an attacker can, for example, force a particular user in a peer-to-peer system to download a malicious version of whatever file it requests, while hiding this attack from the rest of the network.

In addition to a consensus view, our goal is to satisfy the following requirements:

- The system should introduce low overhead.
- The system should work well even when there are sparse observations in order to be successful in restricted-routing or high-churn networks.
- The system should be resilient to attacks by colluding groups of malicious nodes.

In this paper, we focus on an attacker or colluding group of attackers attempting to increase their measured bandwidth. This is a common case in peer-to-peer networks where nodes are provided with service in proportion to the service they provide to others. It is also applicable to the Tor network, where routers are selected with probability proportional to their bandwidth; if an attacker can artificially inflate their bandwidth enough, they can with high probability insert themselves a path of interest and attempt to link sources and destinations, thus violating anonymity.

3 Principal Component Analysis

Our main approach for creating a bandwidth evaluation system is to use *opportunistic* observations based on regular interactions between peers. This results in low overhead, as measurements are integrated into the normal operation of the P2P system. We further combine the observations across multiple nodes and arrive at a consensus bandwidth value using principal component analysis.

Each node in the overlay network maintains a vector of the observed performance of other nodes; for each node, we store the bandwidth achieved while communicating with that node. Note that this observation vector does not attempt to measure the *total* bandwidth of the peer nodes, but rather their *current, per-flow* bandwidth. In addition to being more responsive to fluctuations in available bandwidth, this serves as a rough proxy for the bandwidth an additional flow through that node can expect. This vector provides a starting point for evaluating bandwidth, but it can be too sparse, as a node may communicate with only a subset of its peers. Furthermore, the vectors observed at each node will be different, enabling partitioning of the nodes.

To address the first problem, a node can take into account not only its own observations but those made by other nodes. However, we weight the remote observations based on the observed bandwidth of the remote node. This is because a high-bandwidth node can better evaluate the bandwidth of other nodes, whereas a low-bandwidth node is limited by its own bottleneck link.

If we represent the initial observations as a matrix[1] T, with T_{ij} representing the observation of node j by node i, then the new observation, taking remote vectors into account is:

$$T'_{ij} = \sum_{k=1}^{N} T_{kj} \frac{T_{ik}}{\sum_{l=1}^{N} T_{il}}$$

If we let $\overline{T}$ be a normalized version of the matrix T where each row sums to 1, it is easy to see that $T' = T\overline{T}$, or $\overline{T}' = \overline{T}^2$. Given the new observation vector T', we can iterate this process to better weight the remote observations: $T'' = T'\overline{T} = T\overline{T}^2$, $T''' = T\overline{T}^3$.

This process will eventually converge, so long as the communication graph (i.e., the graph of which nodes talk to each other) is connected and not bipartite. The final (normalized) matrix, $\lim_{n\to\infty} \overline{T}^n$, will have each of its rows equal to the principal eigenvector of the matrix $\overline{T}$, e_0. We can therefore use this eigenvector to represent the consensus bandwidth evaluation. In most cases, the convergence will be quite fast as well[2], so e_0 can be approximated by computing $\overline{T}^n$ for relatively small n.

The general technique of computing (or rather approximating) the principal eigenvector is known as principal component analysis. It has been successfully used by several reputation systems, such as EigenTrust [9] and Google's PageRank [3]. In Section 4, we will describe our modifications to PCA that use properties specific to the problem of bandwidth evaluation and make it more resilient to subversion by malicious nodes.

3.1 Evaluations of the Basic Algorithm

We evaluate the basic PCA-based algorithm using a simulated 1000-node network, with node bandwidths based on measurements of the Tor network [7]. We then created a workload consisting of 10 000 flows between the nodes. The flows either directly connected two nodes (representing communication between peers), or were relayed by a third, intermediate node (representing anonymous tunnels in Tor); thus each node was a part of 20 or 30 flows respectively. In each case, peers allocated their bandwidth to each flow using fair queuing.

We then repeated this experiment over multiple rounds, or "ticks," creating 10 000 new flows each time. Peers were chosen either at random, or weighted by their available bandwidth, using the bandwidth estimation results from the previous round. Multiple observations were combined using an exponentially-weighted moving average ($\alpha = 0.25$). We ran the experiment for a total of 1 440 ticks, representing a day's worth of one-minute

[1]In this paper, we assume symmetric links for convenience. This does not represent a limitation of the technique we propose, however; if separate upstream and downstream observations are required, two observation matrices, U and D can be maintained. In this case, the PairMin constraint described in Section 4.2 would ensure that $D_{ij} = U_{ji}$.

[2]The exact requirement is that the Markov process represented by $\overline{T}$ be fast mixing. This will be true if the communication patterns between nodes are random; most structured P2P overlays also exhibit fast-mixing communication patterns.

flows. To compute the consensus evaluation, we iterated the matrix multiplication until $\left|\left|\overline{T}^n\vec{t_0} - \overline{T}^{n-1}\vec{t_0}\right|\right| < \varepsilon$, where $\vec{t_0}$ is a uniform vector. We used a value of $\varepsilon = 10^{-10}$ for all experiments in the paper.

Figures 1 and 2 show the fractional bandwidth estimated by EigenSpeed plotted against the actual bandwidth of each node. We find that two-hop flows (the "Tor scenario") give less accurate results, as a flow's bandwidth can be impacted by an unrelated bottleneck; bandwidth-weighted router selection also decreases accuracy due to non-uniform sampling. But even in the worst case, the log–log correlation between EigenSpeed's estimation and actual node capacity exceeds 0.9.

In some cases, the relative rank of a node is more important than its absolute bandwidth [14]. Figures 3 and 4 plot the estimated rank vs. the actual rank. The accuracy is once again quite good, with a correlation of over 0.99 in the worst case and over 0.9995 in the best case.

Finally, we examined the convergence time of EigenSpeed, i.e., the number of iterations n such that $\left|\left|\overline{T}^n t_0 - \overline{T}^{n-1} t_0\right|\right| < \varepsilon$. Figure 5 plots this n for each tick of the simulation. Initially, the observation matrix is sparse, resulting in slower convergence. However, within less than 15 ticks (each of which represents a minute), convergence time is down to 10 iterations or fewer. Our unoptimized Perl implementation of EigenSpeed can perform an iteration for $1\,000$ routers in approximately one second, so for the medium-sized peer-to-peer networks we are targeting, the expected CPU load of EigenSpeed should be quite low. When the number of routers is increased five-fold, to $5\,000$, this time compute a single iteration increases to approximately 26 seconds, which is in line with the time complexity ($O(\log(n))$ iterations, each taking $O(n^2)$ to complete for n routers) of EigenSpeed. For larger but less dense graphs, the per-iteration time can be reduced to $O(m)$ operations for m non-zero entries in the observation matrix by using sparse matrix techniques.

4 Practical and Security Considerations

In this section, we extend the basic algorithm to handle nodes joining and leaving (*i.e.*, network churn) and to exploit some features of bandwidth estimation in order to better resist attacks by malicious nodes.

4.1 New Nodes and Churn

As discussed above, the PCA algorithm functions well with a sparse graph; *i.e.*, one where nodes have communicated with relatively few other nodes. However, it does require that the observation matrix represent a connected graph. This is potentially problematic, since new nodes joining the network have not yet communicated with any other nodes, corresponding to an all-zero row in the matrix. Complicating the situation is the possibility that some new nodes have communicated only with each other, forming a nontrivial disconnected component. EigenSpeed accounts for this by detecting the giant component in the communication graph and labeling nodes outside the component as "unevaluated," leaving them out of the main PCA computation.

Figure 6 shows the effect of churn on the convergence rate of EigenSpeed; the vertical line corresponds to an average node lifetime of 16 minutes that has been observed in P2P networks [11]. The convergence time remains manageable even in the face of high churn rates.

4.2 Sinks and PairMin

So far we have seen how EigenSpeed works when all nodes are being honest. Malicious nodes, however, can skew the results of the basic algorithm. One possible attack is for a node to become a "sink" by setting its bandwidth observation vector for any other node to 0. It is easy to see that such a node's bandwidth estimate will increase with each iteration at the expense of other nodes.

Pure sinks are, of course, easy to detect, but a node can also become a "near-sink" by setting its observations of other nodes to be very low. More formally, a near-sink in row i will have $T_{ij} \ll T_{ji}$. Such nodes have traditionally been a problem for PCA-based algorithms. PageRank and EigenTrust address this attack by introducing a minimum weight for any observation vector entry by adjusting the transition matrix:

$$T' = \alpha T + (1-\alpha)\frac{1}{N}\mathbf{1_N}$$

However, this works poorly for bandwidth estimation because it compresses the range of observations, which, in the case of bandwidth, spans several orders of magnitude. It also does not completely eliminate the effect of malicious nodes; this can be seen in Figure 8. We created a clique of nodes that pretend to have high bandwidth for communicating with each other, and zero bandwidth for communicating with the rest of the network. These nodes appear as outliers above the $y = x$ line and are still evaluated as considerably higher than the honest nodes, despite the PageRank defense.

We take a different approach to this problem, exploiting the fact that the bandwidth between a pair of two nodes is observed at each node independently. This means that the matrix T (before normalization) should be symmetric. We enforce this constraint by setting the entries T_{ij} and T_{ji} to be the minimum of the two. We also reset all entries on the diagonal to be 0, since we do not trust a node to honestly represent its own bandwidth. This means that a node that tries to become a sink will end up cutting itself off from the rest of the network and will be considered "unevaluated" by the giant component

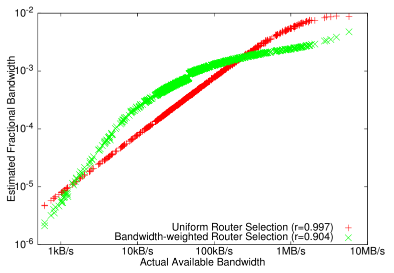

Figure 1: The accuracy of EigenSpeed's bandwidth capacity estimation for 1-hop routes

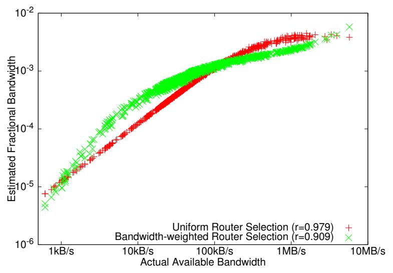

Figure 2: The accuracy of EigenSpeed's bandwidth capacity estimation for 2-hop routes

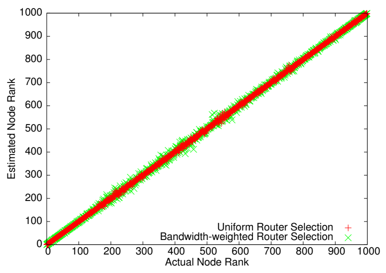

Figure 3: The accuracy of EigenSpeed's bandwidth rank estimation for 1-hop routes

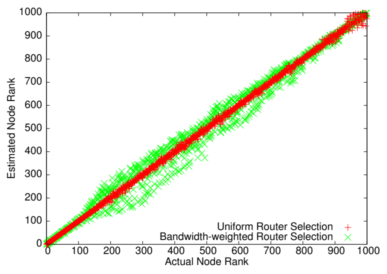

Figure 4: The accuracy of EigenSpeed's bandwidth rank estimation for 2-hop routes

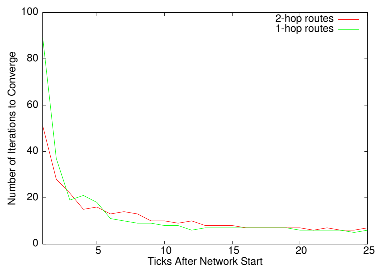

Figure 5: Typical convergence times for EigenSpeed as nodes gather observations.

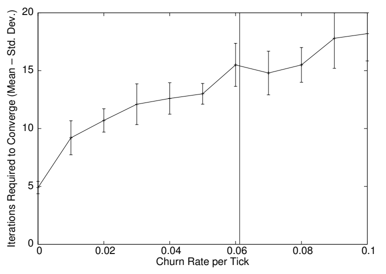

Figure 6: The effect of network churn on the convergence time.

detector in the previous section. For near sinks, the effect is still mitigated by the PairMin approach: Figure 7 shows the effect of a clique with and without PairMin. We simulated a "best possible" near-sink clique by setting the observations of other nodes to 0 but turning off giant component detection. With PairMin, the estimated bandwidth for all nodes remains close to the $y = x$ line.

4.3 Limited Convergence

With the PairMin modification, the normalized transition matrix $\overline{T}$ represents the transition matrix for a random walk along an undirected, weighted graph, where the edge between nodes i and j has the weight T_{ij}. The stationary distribution of this random walk (equivalently, the principal eigenvector of $\overline{T}$) is known to have the form:

$$(e_0)_i = \frac{\sum_{j=1}^{n} T_{ij}}{\sum_{j=1}^{n}\sum_{k=1}^{n} T_{kj}}$$

In other words, the probability of being at node i is the sum of the weights of all edges of i, divided by twice the sum of all the edge weights.

The resulting eigenvector is, intuitively, a very good estimate of a node's bandwidth, since it corresponds to the sum of all external observations of a node. In addition, we can compute this eigenvector much more quickly than with the standard PCA approach. However, the eigenvector is subject to manipulation by malicious nodes. PairMin guarantees that a single node cannot meaningfully alter its observation vector without being penalized; however, a group of colluding nodes can inflate their observations of each other arbitrarily. Since the estimated bandwidth is the sum of all observations for a node, setting even a single observation to a very high value can severely affect the sum. This suggests a modification to the clique attack, which we will call a "fat pipe" attack: the malicious nodes state a very high bandwidth in their observations of each other, and retain the correct bandwidth values in their observations of other nodes (to address PairMin).

This attack can be slightly mitigated by reducing the maximum reportable bandwidth, though at some point this will start to underweight highly provisioned nodes. It turns out that a better defense can be found in our original approach to estimating the principal eigenvector. Our estimate converges quickly when the graph is fast-mixing, which is true whenever all observations are honest. However, under the above attack, the graph will have two fast-mixing components—the honest nodes and the malicious clique—that have a narrow link between them—the normalized values of $\overline{T}_{ij}$ where i is malicious and j is honest will be quite low. Thus mixing *between* the components will happen very slowly and many iterations will be required to reach the stationary vector.

We can exploit this by using a starting vector t_0 that contains a small set of p trusted nodes, with $(t_0)_i = 1/p$ for each trusted node i and 0 otherwise. (If it is hard to find trusted nodes, each node can in fact use itself as the only trusted node.) We then estimate the consensus vector by performing a limited number of iterations of the PCA algorithm; i.e., we compute $\overline{T}^n t_0$, for a small n. If we pick n correctly, $\overline{T}^n$ will have good mixing within each component, but limited flow between the honest and

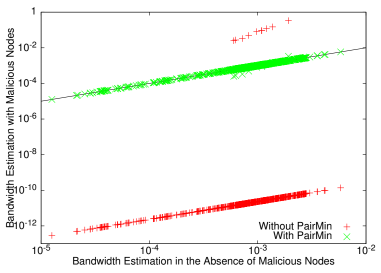

Figure 7: EigenSpeed in the presence of malicious users with and without PairMin; the $x = y$ line is plotted for convenience.

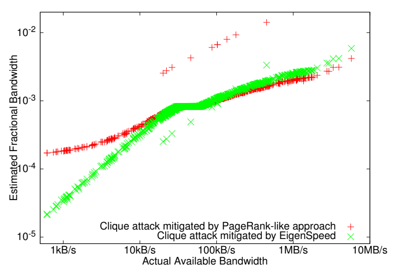

Figure 8: A comparison of a PageRank-like algorithm ($\alpha = 0.85$) and EigenSpeed in the presence of malicious nodes.

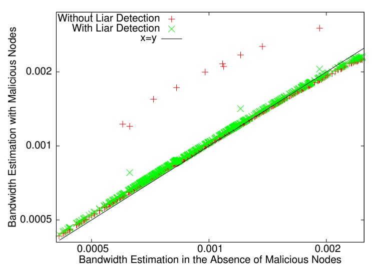

Figure 9: The attack of Section 4.3, with and without Liar Detection, zoomed to show malicious nodes.

malicious components. Thus, if all the trusted nodes are honest, the consensus vector will put limited weight on the malicious nodes.

In practice, we found that we can estimate a good choice for n by using our normal termination condition, i.e., the smallest n such that $||\vec{t_n} - \vec{t_{n+1}}|| < \varepsilon$. The algorithm turns out to be not very sensitive to the choice of ε: we use $\varepsilon = 10^{-10}$, but 10^{-5} still results in quite accurate estimation and 10^{-15} still limits the bandwidth of the malicious clique. Figure 9 shows the effect of the limited number of iterations on the fat pipe attack.

4.4 Liar Detection

As can be seen in Figure 9, malicious nodes still benefit from misreporting their bandwidth. The estimation factor for these nodes is inflated by a factor of 4. This is a relatively minor problem in absolute terms, but if a node's relative rank is used [14], one of the nodes was able to increase its rank from 287 to 3.

To reduce the extent of this problem, we perform "liar detection." Once we obtain an estimate for the consensus bandwidth vector e_0, we can calculate the *liar metric* for a node i by comparing it with the initial observation vector of that node: $||e_0 - \overline{T}_i||_4$.[3] If the liar metric for a node exceeds a threshold L, we call it a liar. We can then either eliminate it from consideration completely, or, more conservatively, add it to the unevaluated set. After this, the computation of the consensus vector is repeated, and a new round of liar detection is applied, iterating until no more liars are identified.

The final algorithm is shown as Algorithm 1. In the absence of new or malicious nodes, it reduces to the simple PCA algorithm. Figure 9 shows the effect of liar detection on the fat pipe attack. Seven out of the ten malicious nodes are identified as liars and are removed from the calculation (they do not appear on the graph). The remaining nodes are seen to obtain only a slight advantage over telling the truth. Liar detection works hand-in-hand with limited convergence: a node that severely misrepresents its observation vector will be detected as a liar, whereas a node that deviates only slightly will not obtain much benefit due to limited convergence.

```
EVALUATE()
recalculate ← true
for $i \leftarrow 0$ to $N-1$
do $T_{ii} \leftarrow 0$
$\overline{T}_{ij} \leftarrow \frac{\min(T_{ij},T_{ji})}{\sum_k \min(T_{ik},T_{ki})}$
while recalculate = true
do $\vec{t} \leftarrow$ GETINITIALVECTOR()
    recalculate ← false
    $C \leftarrow$ GETDISCONNECTEDNODES()
    for each $c$ in $C$
    do MARKUNEVALUATED($c$)
        REMOVENODEFROMMATRIX($c, T$)
    while $\left|\left|T\vec{t} - \vec{t}\right|\right| > \varepsilon$
    do $\vec{t} \leftarrow T\vec{t}$
    for $i \leftarrow 0$ to $N-1$
    do if $\left|\left|t - t_i^0\right|\right| > L$
          then MARKNODEMALICIOUS($i$)
               REMOVENODEFROMMATRIX($i, T$)
               recalculate ← true
// $\vec{t}$ now holds final bandwidth estimates
```

Algorithm 1: Final evaluation algorithm

5 Related Work

Since their inception, peer-to-peer networks have used heterogeneous bandwidth of nodes to improve their behavior. The original Napster peer-to-peer system allowed users to report their connection type (from a list ranging from "14.4 Modem" to "T3") but made no effort to verify this information. Indeed, the data available [11] show a considerably higher fraction of nodes reporting these extrema than the observed bandwidths would support.

Tor, the second-generation onion-routing project, also uses self-reported bandwidth to do load balancing and increase network performance [6]. Routers altering their self-reported bandwidth have been shown to be able to effectively compromise a large fraction of Tor paths [2].

In previous work, we have proposed using a small sample of randomly selected peers to make routing decisions [14]. This approach is resilient to misreported bandwidths, but it enables partitioning attacks and this has prevented its adoption. A small sample also leads to

[3] We use the ℓ^4 norm to count smaller errors more lightly.

much lower accuracy than EigenSpeed.

The BitTorrent file-sharing protocol also uses direct observations in its tit-for-tat protocol. However, since data about these ratios is not shared between peers, attacks such as the "Large View" exploit [13] can exploit the optimistic feature of tit-for-tat to download at a high speed from the network without ever uploading.

A related problem is the estimation of latency between peers. Here, the dimensionality of the problem can be reduced by creating virtual coordinates using spring relaxation techniques [5]. Veracity protects these coordinates by introducing checks by a random (but verifiable) set of monitor nodes [12]. The authors suggest that the same approach could be used for bandwidth, but low-bandwidth nodes will have difficulty monitoring high-bandwidth ones. EigenSpeed, in essence, is able to use *all* peers as monitors without introducing extra traffic.

As mentioned above, PCA is used in many applications, including PageRank [3] and EigenTrust [9]. When resilience to dishonest nodes is needed, the usual approach is to modify the transition matrix to include a small chance of transitioning to a random node. This is necessary since these applications cannot exploit the symmetry of T as we do in this paper. The disadvantages of this approach were discussed in Section 4.2.

6 Conclusions and Future Work

We have presented EigenSpeed, a novel algorithm for distributed estimation of bandwidth in a peer-to-peer system, that is accurate, efficient, and secure. We demonstrated that EigenSpeed produces accurate results by testing it with a set of peer bandwidths collected from the Tor network and showed that it is resilient to both network churn and malicious attacks. EigenSpeed can be used to achieve consensus bandwidth estimation and balance load across all nodes in a peer-to-peer network in a secure fashion while maintaining similar accuracy to self-reporting.

In our future work, we will explore how to compute the consensus bandwidth in a decentralized fashion, following an approach similar to that of EigenTrust. We will also evaluate the accuracy of EigenSpeed as extended to support asymmetric upload and download bandwidths and investigate the possibility of new attacks introduced by such a change.

We also plan to perform a whole-system evaluation of EigenSpeed as a component of a peer-to-peer network such as Tor in a testbed environment. Finally, we will derive formal bounds on the influence of malicious peers.

References

[1] AKELLA, A., SESHAN, S., AND SHAIKH, A. An empirical evaluation of wide-area internet bottlenecks. In Crovella [4], pp. 101–114.

[2] BAUER, K., MCCOY, D., GRUNWALD, D., KOHNO, T., AND SICKER, D. Low-resource routing attacks against Tor. In *Workshop on Privacy in Electronic Society* (November 2007), T. Yu, Ed., pp. 11–20.

[3] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th international conference on World Wide Web (WWW'98)* (1998).

[4] CROVELLA, M., Ed. *3rd ACM SIGCOMM Conference on Internet Measurement, October 27–29, 2003, Miami Beach, FL, USA* (New York, NY, USA, 2003), ACM.

[5] DABEK, F., COX, R., KAASHOEK, F., AND MORRIS, R. Vivaldi: a decentralized network coordinate system. In *SIGCOMM* (New York, NY, USA, 2004), E. Zegura and J. Rexford, Eds., ACM, pp. 15–26.

[6] DINGLEDINE, R., AND MATHEWSON, N. Tor path specification. `http://www.torproject.org/svn/trunk/doc/spec/path-spec.txt`.

[7] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router. In *USENIX Security Symposium* (Aug. 2004), M. Blaze, Ed., USENIX Association, pp. 303–320.

[8] HOTELLING, H. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology 17* (1933), 417–441.

[9] KAMVAR, S. D., SCHLOSSER, M. T., AND GARCIA-MOLINA, H. The eigentrust algorithm for reputation management in P2P networks. In *International Conference on World Wide Web* (New York, NY, USA, 2003), Y.-F. R. Chen, L. Kovács, and S. Lawrence, Eds., ACM, pp. 640–651.

[10] LAKSHMINARAYANAN, K., AND PADMANABHAN, V. N. Some findings on the network performance of broadband hosts. In Crovella [4], pp. 45–50.

[11] SAROIU, S., GUMMADI, P. K., AND GRIBBLE, S. D. A measurement study of peer-to-peer file sharing systems. In *MMCN* (Jan. 2002).

[12] SHERR, M., LOO, B. T., AND BLAZE, M. Veracity: A fully decentralized service for securing network coordinate systems. In *International Workshop on Peer-to-Peer Systems* (2008), A. Iamnitchi and S. Saroiu, Eds.

[13] SIRIVIANOS, M., PARK, J. H., CHEN, R., , AND YANG, X. Free-riding in BitTorrent networks with the large view exploit. In *IPTPS* (Feb. 2007).

[14] SNADER, R., AND BORISOV, N. A tune-up for Tor: Improving security, performance and anonymity in the Tor network. In *Network and Distributed System Security Symposium* (Feb. 2008), C. Cowan and G. Vigna, Eds.