

Integrity-aware Bandwidth Guarding Approach in P2P Networks*

Wen-Hui Chiang¹, Ling-Jyh Chen², Cheng-Fu Chou³

¹ University of Southern California, Los Angeles, CA 90089, USA

² Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan

³ National Taiwan University, Taipei, Taiwan

ccf@csie.ntu.edu.tw

Abstract. Most Internet-based collaborative computing systems face the major problem: freeriding. The abundance of freeriders, and load imbalance it creates, punishes those peers who do actively contribute to the network by forcing them to overuse their resources. Hence, the overall system performance becomes to degrade quickly. The goal of this paper is aimed to provide an efficient approach to distinguish the dishonest peers from the honest peers. The key idea of our approach is to make use of the relationship between the perceived throughput and the available bandwidth. First, we do a comprehensive study of available bandwidth estimation tools. Next, we propose integrity-aware bandwidth guarding algorithm, which is designed according to the perceived throughput and the available bandwidth estimation. Finally, the simulation results illustrate that our approach can correctly identify dishonest peers and be of great help in constructing a better overlay structure for many peer-to-peer and multicast applications.

Keywords: peer-to-peer, available bandwidth, freeriding

1 Introduction

A peer-to-peer (or P2P) computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. An important characteristic in peer-to-peer networks is that all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demand on the system increases, the total capacity of the system also increases. Peer-to-peer computing has

* This work was partially supported by the National Science Council and the Ministry of Education of ROC under the contract No. NSC95-2221-E-002-103-MY2 and NSC95-2622-E-002-018.

changed the way people interact in the areas of information sharing and collaboration. Hence, in the past few years, peer-to-peer applications have become more and more popular in the Internet.

Most Internet-based collaborative computing systems, including P2P file sharing system such as Gnutella, Napster, and Kazaa, potentially face the problem of freeriding: that is, users or peers that consume resources of the system without contributing anything in return. Many studies [15][17] have shown up to 70% of Gnutella clients do not share any files, and nearly 50% of all responses are returned by 1% of the peers. This abundance of freeriders, and load imbalance it creates, punishes those peers who do actively contribute to the network by forcing them to overuse their resources (e.g. bandwidth). Hence, the system performance becomes to degrade quickly.

Previous research works [17] in this area has focused primarily on currency-based systems wherein peers gain currency for uploading files, and use currency when downloading files. In [14], they propose an EigenTrust algorithm, in which the global reputation of each peer i is given by the local trust values assigned to peer i by other peers, to construct the reputation management. On the other hand, BitTorrent [16] employs a tit-for-tat incentive mechanism to reduce freeriding and increase user cooperation. However, they also find out that in torrent with a large number of seeders, the BitTorrent tit-for-tat mechanism may not succeed in producing a disincentive for freeriding: in such torrents, freeriders may actually experience faster download times than cooperating peers. Hence, to cope with the above problem, the goal of this paper is how to figure out an efficient approach to distinguish the honest peers from the dishonest peers (e.g., freeriders), which consume resources of the system but contribute little or nothing in return.

Since the Internet traffic is quite dynamic over time, it is not easy to distinguish the dishonest peers from the honest peers on the Internet according the perceived throughput. That is, such throughput-based approach is difficult to distinguish the reason of throughput degradation due to the network congestion or the bad peers. In this work, we aim to propose an efficient method to distinguish the dishonest peers from the honest peers. The key idea of our approach is to observe the relationship between the perceived throughput and the available bandwidth. When a network connection is affected by congestion, the available bandwidth and the throughput should both decrease. On the other hand, when the dishonest peer reduces its sending rate and the perceived throughput might decreases irregularly, but the available bandwidth should be the same or increase.

There are many peer-to-peer-based services and applications in which our integrity-aware bandwidth guarding scheme can be of great help. For example, the fair sharing and peers are encouraged to contribute as much as possible. P2P file sharing system or P2P based multicast applications. By selecting a proper peer to cooperate with, not only the throughput of each peer could increase but also the system performance could improve substantially. We illustrate these points in the section 3. At last, we note that the proposed approach can be used in conjunction with currency-based approach, EigenTrust-based approach, or the reputation system to provide more efficient way to distinguish the dishonest peers.

The rest of the paper is organized as follows. In section 2, we first introduce the general framework of our approach, several potential candidates of available

bandwidth estimate tools and explain our algorithm in details. In section 3, we first do an accuracy study of the proposed approach and then illustrate how to integrate our approach to substantially improve the performance of multicast applications or peer-to-peer applications. Finally, future work is given in the section 4.

2 Framework

In this work, we propose a fast and precise bandwidth guarding approach, which is an active detecting method and consider the peer integrity. The major idea of the integrity-aware bandwidth guarding approach is to explore the relationship between the available bandwidth and the throughput and differentiate the honest peers from the dishonest peers. As we discussed before, most of the detecting schemes are based on the observation of the connection throughput. However, such passive detecting methods are difficult to identify the degradation of throughput which is based on peer's selfish behaviour or network congestion. Thus, to address the above problem, we let each host be able to figure out its corresponding host's behaviour by investigating the perceived throughput and the available bandwidth. Since a peer might be able to decrease the connection throughput but it is difficult for him to manipulate the available bandwidth of the path.

There are two major components in our approach and they are (a) selection of a proper available bandwidth probing scheme, and (b) an intelligent irregular throughput detecting algorithm. In the following, we first do a comparison study about several famous available bandwidth detecting schemes and then present how to integrate this available bandwidth probing scheme into our framework to discover the dishonest peers.

2.1 Studies on Available Bandwidth Estimation Tools

Available bandwidth is useful information for the route selection, quality-of-service verification, and traffic engineering in the overlay network or Internet. The definition of available bandwidth is the unused capacity at a link. Figure 1 illustrates the relationship between the link capacity, cross traffic, and the available bandwidth. C is the capacity of the link and $A[0, T]$ is the traffic from time 0 to time T . The available bandwidth is the average unused bandwidth over some time interval T . Thus,

$$B[0, T] = \min(C - \frac{A[0, T]}{T}, 0)$$

In general, the bandwidth estimation techniques are classified into single packet methods and packet pair methods. Single packet methods estimate the link capacity by measuring the time difference between the round-trip time (RTT) to one end of an individual link and that to the other end of the same link. Single packet tools include pathchar, clink, and pchar.

Packet pair methods send groups of back-to-back packets, i.e., packet pairs, to a server which echoes them back to the sender. The spacing between packet pair is

determined by the bottleneck link. Example tools include NetDyn probes, bprobe, nettimer, and Spruce [1]. In this paper, we focus on comparing three available bandwidth estimation tools: Spruce [1], Iperf [7], and pathChirp [2].

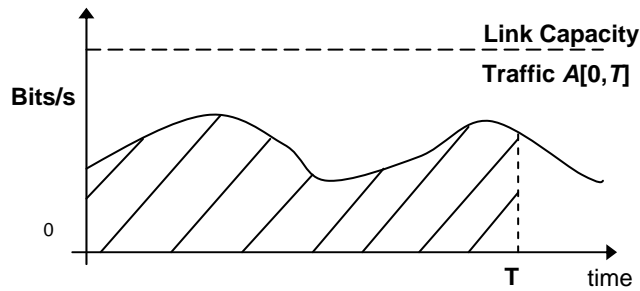


Fig. 1. The Definition of Available Bandwidth

2.2 Comparison of Available Bandwidth Estimation Tools

Since the performance of our integrity-aware bandwidth guarding approach depends on the efficiency of the available bandwidth estimation tool, we need to choose the suitable available bandwidth estimation tool with light-weight probing traffic and accurate estimation in multi-bottleneck network paths. Thus, we compare three available bandwidth estimation tools, i.e., Spruce, pathChirp and Iperf, in single bottleneck, pre-bottleneck and post-bottleneck environments.

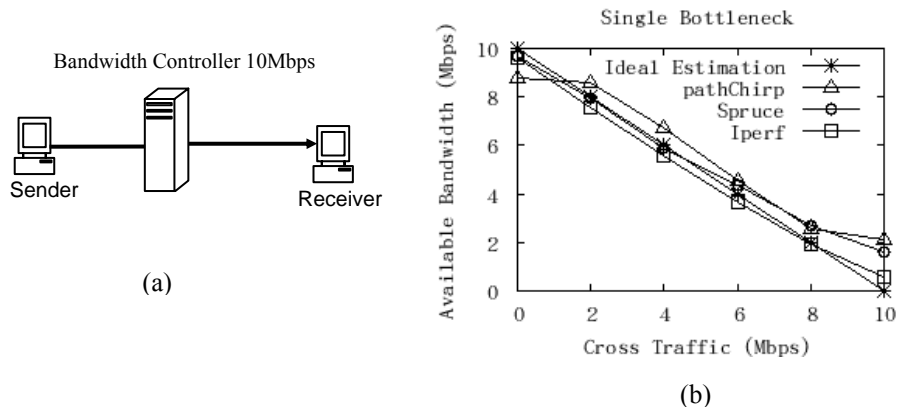


Fig. 2. (a) The setup of one bottleneck experiment. (b) The comparison of estimation tools in a bottleneck path

At first, we set up a simple experiment as shown in Figure 2(a) to evaluate the accuracy of the estimation tools. The bottleneck controller is installed with FreeBSD

and Dummynet. The experimental result is depicted in Figure 2(b). We can see that all tools perform well under a single bottleneck case.

In the next experiment, we would like to create a post-bottleneck scenario for the targeted connection. So, we prepare six computers to construct the emulation experiment. In Figure 3, the sender and the receiver are hosts with Linux Fedora 2.6.11-1.1369_FC4smp.

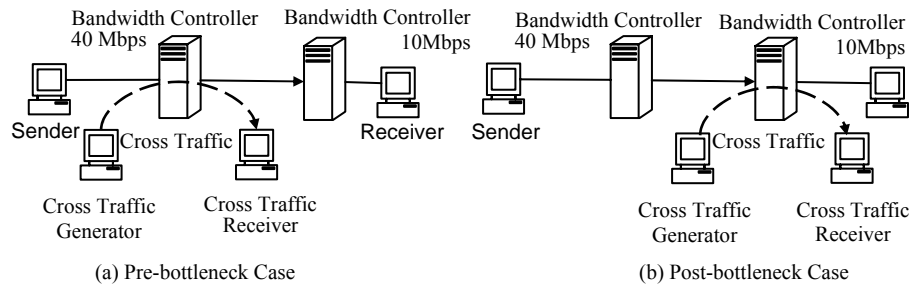


Fig. 3. The pre-bottleneck and post-bottleneck experiment environment

Two bandwidth controllers are the other hosts with FreeBSD 5.4 Release. The cross traffic generator and the receiver are notebook computers with Linux Fedora 2.6.11-1.1369_FC4smp. For two bandwidth controllers, we installed Dummynet [16] to limit the link capacity as we expect. The first bandwidth controller's capacity is 40Mbps, and the second is 10Mbps. In cross traffic generator and cross traffic receiver, we use the Poisson traffic generator to generate the cross traffic. We control the cross traffic generator to generate cross traffic from 25Mbps to 45 Mbps. When the cross traffic is larger than 30 Mbps, the bottleneck happens at the first host. Thus, it is the same as the post-bottleneck experiment.

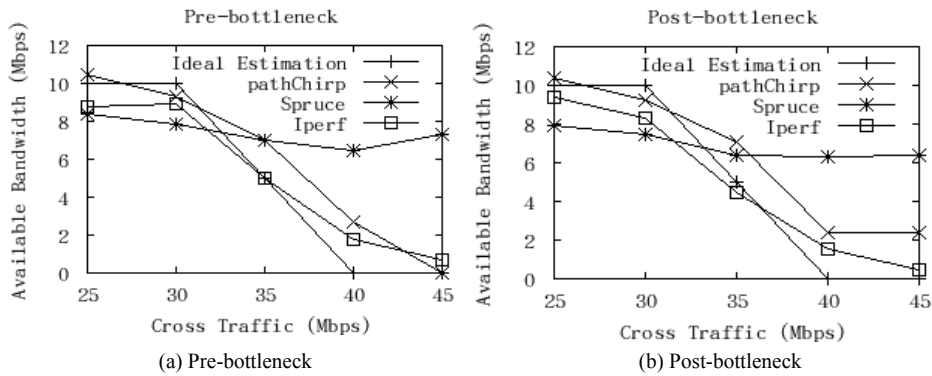


Fig. 4. The pre-bottleneck and post-bottleneck experiment

Figure 4(a) illustrates the pre-bottleneck experimental result and the line with cross

represents the ideal available bandwidth. It equals the capacity of the path minus the cross traffic in the bottleneck. In this experiment, Iperf performs still better since its predicted result is very close to the ideal estimated result. The Spruce performs poorly in the experiment.

We set up a post-bottleneck experiment as shown in Figure 4(b) to compare these 3 available bandwidth estimation tools. We can see that Iperf still performs better and is able to get the most accurate estimation. The estimated bandwidth by the pathChirp is little higher than the actual value. We note that the Spruce failed to estimate the available bandwidth under such post-bottleneck case.

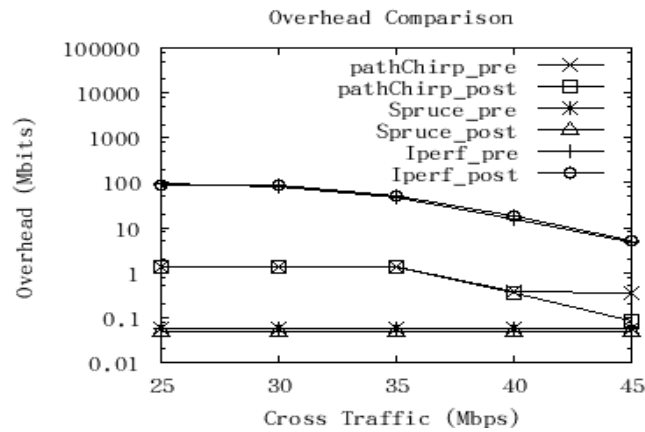


Fig. 5. The overhead comparison of pathChirp, Spruce, and Iperf

Now we turn our attention to the probing overhead of the estimation tools. In other words, we hope that the probing overhead is as low as possible, i.e., the probing data does not affect the normal traffic. Hence, we do the comparison study of probing overhead. In Figure 5, we see that Iperf usually has probing overhead more than 10Mbps. The overhead of the pathChirp is around 0.1Mbps and the Spruce's overhead is less than 0.01Mbps. In conclusion, according to the accuracy and overhead comparison, we decide to choose pathChirp as our available bandwidth estimation tool. In fact, it performed well even in multi-bottleneck network environment with light weight probing overhead.

2.3 Irregular Throughput Detection Algorithms

To figure out a dishonest peer, the main idea of our approach is to observe the relationship between the received throughput and the available bandwidth. When a network connection is affected by congestion, the available bandwidth and the throughput should both decrease. However, when the user cheats by cutting down its transmission rate, the observed throughput tends to decrease irregularly but the available bandwidth is likely to be the same or increase. According to this important observation, we design our algorithm to figure out the irregular patterns and locate the immoral behaviors of the dishonest peers.

Now we first introduce the notations, as shown in Table 1, to explain the details of our algorithm. We define *Expected Throughput*, $R_E[t]$, which is the expected throughput at time t without cheating. Based on $R_E[t]$, we compare the current throughput and expected throughput to check if the current throughput is reasonable or not. We apply *Exponential Weighted Moving Average* (EWMA) to compute $R_E[t]$ in our algorithm. In **Algorithm 1** depicted in Figure 6, we update $R_E[t]$ every time interval τ . During time interval τ , we calculate the *average perceived throughput* $R_{avg}[t-\tau]$ and compare $R_{avg}[t-\tau]$ with $R_E[t]$.

If $R_E[t]$ decreases more than $\alpha R_E[t]$ and *available bandwidth* $A[t]$ increases more than $(1+\rho)A[t-\tau]$, the throughput in time interval τ will be classified as cheating traffic and the traffic in time interval τ will not be counted in $R_E[t]$. α is defined as $\frac{1.6 * R_{std}[t-\tau]}{R_{avg}[t-\tau]}$, the coefficient of variance of throughput in time interval τ . We

assume that the variation of the traffic in a short term could be approximated as a normal distribution and use the 95% confidence interval, i.e., 1.6 times the standard deviation

Table 1. Definitions of Notations

$R_E[t]$	Expected throughput (achieved rate) at time t
$R_{avg}[t-\tau]$	Average throughput in time interval τ
$R_{std}[t-\tau]$	The standard deviation of throughput in time interval τ
$A[t]$	Available bandwidth at time t
P	The available bandwidth estimation error rate
Ω	The weighted parameter used in counting expected throughput
A	The coefficient of variance of throughput in time interval τ

Algorithm 1 Compute Expected Throughput

```

1: if  $|R_E[t] - R_{avg}[t-\tau, t]| < \alpha R_E[t]$ 
   and  $A[t] \leq (1 + \rho)A[t-\tau]$  then
2:   return
    $[t] = R_E[t-\tau] \times (1 - \omega) + R_{avg}[t-\tau, t] \times \omega$ 
3: else
4:   return  $R_E[t] = R_E[t-\tau]$ 
5: endif

```

Fig. 6. Algorithm 1 Compute Expected Throughput

In Algorithm 2 illustrated in Figure 7, we use the *Expected Throughput*, $R_E[t]$, and the available bandwidth to identify if the peers are cheating. α and ρ are defined as the same as in Algorithm 1. In Algorithm 2, we monitor the throughput and the available bandwidth as shown in second and third lines. If the throughput drops too fast and the available bandwidth increases more than $(1+\rho)A[t-\tau]$, we consider the peer may be cheating and increase the parameter *possible_cheating*. When the *possible_cheating* is more than μ , which is the threshold value and we use that to identify the peer as “dishonest peer.” The *possible_cheating* might increase while the network conditions return to the normal condition. This mechanism avoids the situation that the peer suffers sudden abnormal network conditions and may return to the normal condition in next time interval τ .

Algorithm 2 Irregular Throughput Detection

```

1: for all interval  $\tau$  do
2:   if  $R_E[t] - R_{avg}[t - \tau, t] > \alpha R_E[t]$  then
3:     if  $A[t] \geq (1 + \rho)A[t - \tau]$  then
4:        $possible\_cheating \leftarrow possible\_cheating + 1$ 
5:       if  $possible\_cheating > \mu$  then
6:         return cheating_detected
7:       end if
8:     else
9:        $possible\_cheating \leftarrow possible\_cheating - 1$ 
10:      return nothing
11:    end if
12:  end if
13: end for

```

Fig. 7. Algorithm 2 Irregular Throughput Detection Algorithm

We also take the available bandwidth $A[t]$ into consideration. If the $A[t]$ increases more than $(1+\alpha)A[t-\tau]$, the traffic may be cheating. We define ρ the threshold of estimation error. By experiments shown in Figure 8, we observe that the estimation error is less than 5% in 129 seconds, i.e., 60 rounds. Hence, we set ρ equal to 5% and τ more than 129 seconds in **Algorithm 1** because our experiment could have error rate almost $\pm 5\%$ in 129 seconds. ω is the weighted parameter used in computing expected throughput. In this work, we let ω equal to 0.8.

3 Results

In this section, we evaluate the performance of the integrity-aware bandwidth guarding approach through the simulations, which are done by using the NS-2[10] simulator. We consider two different systems: peer-to-peer applications and multicast applications, in which the integrity-aware bandwidth guarding method can be of great help. The TCP-Friendly Rate Control (TFRC) protocol is used in all the simulations. In all simulations, we adopt the pathChirp to estimate the available bandwidth.

The performance metrics are (a) the error rate: the ratio of correct detection of dishonest peers, (b) the average waiting time: the mean time to retrieve the targeted

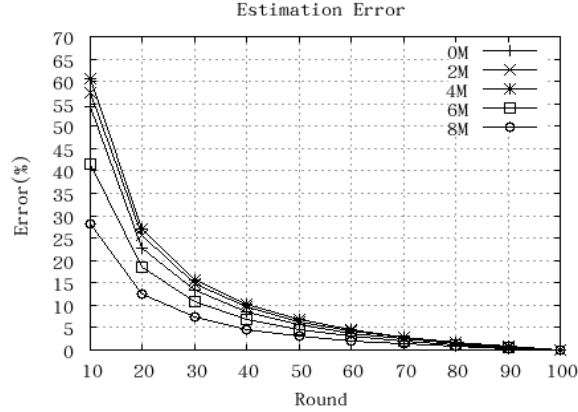


Fig. 8. The estimation error of pathChirp

file over all the peers, and (c) the longest waiting time: the amount of time for the slowest peer to retrieve the targeted file.

3.1 Accuracy Study: Peer-to-Peer Applications

In this experiment, we would like to illustrate the accuracy of our approach, i.e., the successful ratio of using our algorithm to tell the cheating peers from the honest ones. We use Brite [18] to generate the topology, which is in accordance to the Waxman model [19]. There are total 100 nodes and 200 links in our simulation topology. The bandwidth of each link is determined to follow the heavy-tailed distribution and the range of the bandwidth of the link is from 1 Mbps to 20 Mbps. In addition, we use the traditional throughput-based scheme as the baseline comparison.

In the first experiment, we randomly select 50 connections and we vary the percentage of the dishonest peers from 20% to 80%. After some period of time, dishonest peers start to reduce by half of their original sending rate.

From figure 9(a), we observe that the performance between the throughput-based approach and our integrity-aware approach is comparable to each other. This is because when the load on the network is low, the throughput is mainly determined by the sending rate of the peer, i.e., the throughput is able to imply the behavior of the peer. That is, when the dishonest peer decreases its sending rate, it is enough to distinguish most dishonest peers by observing the change of their throughput only. To explore the benefit of our integrity-aware bandwidth guarding approach, in the following experiment, we consider a scenario in which the background traffic is introduced into the network after 300 seconds such that all the connections suffer serious network congestion from that time. Moreover, there is no dishonest peer in this experiment.

From Table 2, it is not surprising that the throughput-based detection approach works poorly, i.e., the error rate is 100%. This can be explained as follows. The throughput-based approach can not tell that the degradation of the throughput is due to the peer behavior or the actual network condition. On the contrary, since our

approach considers both available bandwidth and throughput, the error rate is able to keep quite small, i.e., the error rate is 6% in this case. Therefore, it is essential to consider both available bandwidth and throughput to detect the cheating peer correctly.

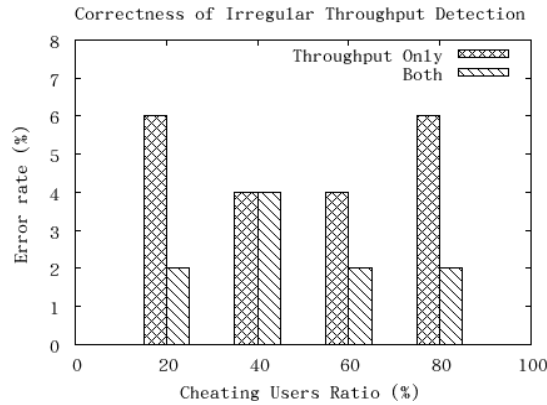


Fig. 9. The Error Rate of Irregular Throughput Detection Algorithm

Table 2. Error rate of honest peers

	Throughput Detection	Available bandwidth and throughput
Honest peers	100%	6%

3.2 Performance Study: Multicast Applications

The motivation for this experiment is that we would like to illustrate the advantages by using our bandwidth guarding approach to figure out a better peer and construct a proper overlay structure for many systems or applications. That is, we want to investigate the impact of selecting a proper peer on the construction of multicast structure. We note that the performance of multicast systems primarily depends on the underlying multicast tree structure. Usually, a host would like to be placed as close as to the root to get the content first because the content is delivered from the root to the leaf node. For example, a dishonest peer would like to claim that it has quite large transmission bandwidth initially when the multicast tree is built. Hence, once the multicast tree is constructed based on the false information, all the peers under that dishonest host probably suffer a long waiting time until they get the required content or file.

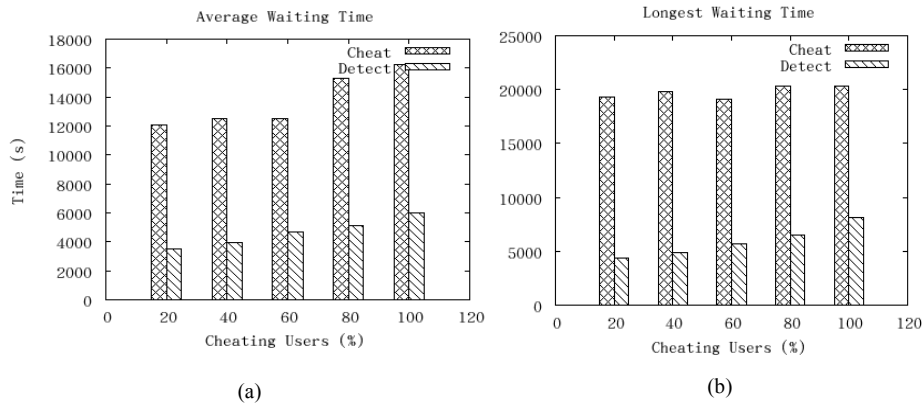


Fig. 10. (a) The average waiting time in different ratio of cheating peers (b) The longest waiting time in different ratio of cheating peers

The simulation environment is as follows. There are 50 nodes, which want to join this multicast group. If a peer is a dishonest peer, it announces that its transmission bandwidth is from 2 to 5 times of its original transmission bandwidth. After the root is decided, the naïve method starts to construct the minimum spanning tree according to the claimed bandwidth of each peer. On the other hand, the root can use our integrity-aware bandwidth guarding approach to build the suitable minimum spanning tree. The performance metrics are the average waiting time and the longest waiting time. The waiting time refers to the period of time which a peer spends to retrieve the required content.

In Figure 10, we can see that the multicast application with the help of our approach always outperforms the multicast application with the naïve approach in term of the average waiting time and the longest waiting time. Furthermore, even a small amount of dishonest peers, such as 20% participants, are introduced into the system, the system performance still degrade quickly. That is, the dishonest peers claim that their transmission bandwidth is large and usually have better opportunities to be placed close to the root. Hence, all the hosts located under that dishonest peer will experience a longer waiting time to get the required content. Therefore, the integrity-aware bandwidth guarding algorithm can be used to substantially improve the performance of multicast applications.

4 Future Work

In this work, we have proposed an integrity-aware bandwidth guarding algorithm. This approach is able to correctly identify dishonest peers and avoid misjudging honest peers, which are affected by poor network conditions. In addition, our algorithm can be used to construct a better underlying structure, which can substantially improve the performance of multicast applications. Our ongoing work is focusing on how to integrate our approach with other approaches to develop a better reputation system, which could provide more incentives to attract honest peers.

References

1. J Strauss, D Katabi, F Kaashoek, A measurement study of available bandwidth estimation tools, in Proceedings of the ACM SIGCOMM, 2003
2. Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, Les Cottrell, "pathChirp: efficient available bandwidth estimation for network paths," in Proceedings of the 3rd Passive and Active Measurements, 2003
3. Allen B. Downey, Using pathchar to estimate Internet link characteristics, in Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, 1999
4. Jean-Chrysostome Bolot., End-to-end packet delay and loss behavior in the internet, In Proceedings of ACM SIGCOMM, San Francisco, CA, September 1993
5. K Obraczka, G Gheorghiu, The performance of a service for network-aware applications, In Proceedings of the SIGMETRICS symposium on Parallel and distributed tools, 1998
6. K Lai, M Baker, Nettimer: A tool for measuring bottleneck link bandwidth, in Proceedings of the USENIX Symposium on Internet Technologies, 2001
7. A Tirumala, F Qin, J Dugan, J Ferguson, K Gibbs, <http://dast.nlanr.net/Projects/Iperf>
8. L Rizzo, Dummynet: a simple approach to the evaluation of network protocols, In proceedings ACM SIGCOMM Computer Communication Review, 1997
9. M. Handley, J. Padhye, S. Floyd, and J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, IETF RFC 3448, January 2003
10. S. McCanne and S. Floyd. NS: Network Simulator. <http://www-mash.cs.berkeley.edu/ns/>
11. EW Zegura, KL Calvert, S Bhattacharjee, How to model an internetwork, In the Proceedings of IEEE INFOCOMM, 1996
12. Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers, BRITE: An Approach to Universal Topology Generation, In Proceedings of the MASCOTS '01, Cincinnati, Ohio, August 2001
13. E. Adar and B. Huberman. Free Riding on Gnutella. First Monday, 5(10), Oct. 2000. S. Kamvar, M. Schlosser, and H. Garcia-Molina.
14. The EigenTrust Algorithm for Reputation Management in P2P Networks. In WWW 2003, 2003.
15. D. Hughes, G. Coulson, and J. Walkerdine. Freeriding on Gnutella Revisited: the Bell Tolls? IEEE Distributed Systems Online, 6(6), 2005
16. N. Andrade, M. Mowbay, A. Lima, G. Wagner, and M. Ripeanu, Influences on Cooperation in BitTorrent, In SIGCOMM P2P-ECON workshop, 2005.
17. P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge. Incentives for Sharing in Peer-to-Peer Networks, In WELCOM, 2001.
18. A Medina, A Lakhina, I Matta, J Byers, BRITE: An Approach to Universal Topology Generation Proceedings of MASCOTS, 2001
19. B. M. Waxman. Routing of multipoint connections. IEEE Journal on Selected Areas in Communications, 6(9):1617-1622, Dec. 1988.