Throttling Attackers in Peer-to-Peer Media Streaming Systems

William Conner and Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign, Urbana, IL 61801

Abstract

We present a flexible framework for throttling attackers in peer-to-peer media streaming systems. In such systems, selfish nodes (e.g., free riders) and malicious nodes (e.g., DoS attackers) can overwhelm the system by issuing too many requests in a short interval of time. Since peer-to-peer systems are decentralized, it is difficult for individual peers to limit the aggregate download bandwidth consumed by other remote peers. This could potentially allow selfish and malicious peers to exhaust the system's available upload bandwidth. Our framework provides a solution to this problem by utilizing a subset of trusted peers (called kantoku *nodes) that collectively monitor the bandwidth usage of untrusted peers in the system and throttle attackers. We evaluate our framework through simulation.*

1. Introduction

Many peer-to-peer (p2p) media streaming applications have been developed over the past few years. For example, PROMISE and CoopNet are p2p media streaming systems that support many-to-one streaming [1,2], while SplitStream and oStream focus on multicast scenarios [3,4]. Both on-demand and live p2p streaming scenarios have been considered [5,6].

Although p2p designs offer many benefits, such as reduced costs and scalability, they also offer new opportunities for misbehavior. For example, a selfish peer might request more than its fair share of bandwidth, while a malicious peer might want to intentionally exhaust all of the available upload bandwidth in the system. Due to the lack of a central server, misbehaving peers can distribute their requests throughout the system such that they appear well-behaved to each individual node, but their aggregate behavior would appear to be selfish or malicious overall.

Given this vulnerability, it becomes clear that such p2p systems need effective mechanisms to throttle the aggregate bandwidth consumed by each peer in the system. Unlike client-server architectures where everything can be monitored at a central server, p2p architectures will need a more robust and scalable alternative.

In this paper, we argue that a subset of trusted peers can collectively limit the bandwidth usage of all the other untrusted peers in the system. We refer to these trusted nodes as *kantoku* nodes. Kantoku nodes monitor control messages related to p2p streaming sessions in an effort to determine how much bandwidth each peer is consuming at any given time. We should emphasize that our approach does not completely prevent attacks, rather it adaptively throttles attackers according to their level of abuse to ensure that selfish and malicious nodes cannot consume more than their fair share of bandwidth.

In the next section, we present related work. Section 3 introduces our kantoku framework. A simulation-based performance evaluation follows in Section 4. Finally, we conclude our work in the last section.

2. Related Work

The PlanetLab Central (PLC) is an example of a centralized global resource utilization enforcement infrastructure service that lacks scalability and robustness to failures [8,9].

Alternatively, many decentralized solutions have been proposed that use tickets, tokens, or credit [10,11,12]. However, the expensive steps of credit-path discovery or token exchange associated with these schemes could be too heavyweight for delay-sensitive p2p media streaming applications.

DoS-limiting network architectures that require edge router awareness have also been developed [21]. In contrast, kantoku is completely transparent to routers and runs completely on end hosts. Daswani and Garcia-Molina consider query flooding attacks in an unstructured p2p network, but our work focuses on bandwidth exhaustion that results from media streaming rather than query message flooding [7].

Previously, we have done work on limiting such attacks in p2p media streaming systems [13]. One shortcoming of our first protocol was that it treated all attackers in the same manner by limiting them to their maximum download rate allowed. Our kantoku protocol is more adaptive and flexible because it

adjusts to the attacker by punishing the attacker in proportion to the level of abuse committed by the attacker.

3. Kantoku Framework

In this section, we present our framework for throttling selfishness and malicious denial-of-service (DoS) attacks from downloaders in p2p media streaming systems. Our framework, which we refer to as the *kantoku*[1] framework, utilizes a subset of trusted nodes to cooperatively monitor and limit the amount of download bandwidth consumed by individual nodes. Flash crowds and DoS attacks are two examples when the upload bandwidth available in the system might be exhausted [14]. In both cases, aggressively limiting each node to download no more than its fair share when resources are scarce prevents both selfish and malicious nodes from gaining an unfair advantage over well-behaved nodes.

Before explaining kantoku in detail, we must first describe our application model, p2p lookup model, and attack model. We conclude this section with a detailed description of our protocol and algorithms used.

3.1. Application Model

Our p2p media streaming application consists of a set of distributed nodes that stream media to one another. Each individual streaming session consists of one or more senders and a single receiver, which is similar to many current p2p media streaming systems [1,2]. Our application assumes a simplified session control protocol where nodes initiate streaming sessions with START messages and terminate sessions with STOP messages. For the purposes of our application model, we do not assume a particular media type (e.g., audio or video), but we do assume multiple description coding (MDC) could be used as mentioned in [2]. Figure 1 depicts the many-to-one streaming application model that we have just described with four uploading peers *W*, *X*, *Y*, and *Z* each streaming a description *desc* for the same video to a single downloading peer *P*. Our model assumes that every node has a maximum upload capacity and a maximum download capacity. These maximum capacities could be determined by the application's download policy (e.g., no downloading at an aggregate rate higher than 1 Mbps), external policy (e.g., ISP limits), or even physical limitations (e.g., dial-up connection limited to 56 Kbps). We assume that coordination among uploaders is receiver-driven as explained in [1].

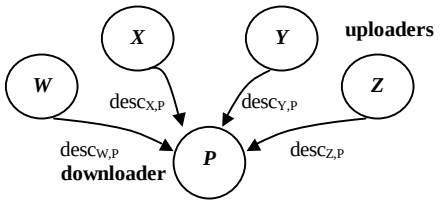

Figure 1. Many-to-one streaming with MDC

Formal definitions that we use throughout this section appear below in Table 1.

r_{ij}	current data rate from node *i* to node *j*
$Dmax_j$	maximum download rate allowed for node *j*
D_j	current aggregate download rate for node *j*
	$= \sum_{i=1 \text{ to } n} (r_{ij})$
	must be $\leq Dmax_j$ for each node *j*

Table 1. Formal application model definitions

Referring to the model in Table 1 for some node *i*, we cannot expect node *i* to voluntarily enforce the constraint D_i not exceed $Dmax_i$, because it would be limiting its own benefit (e.g., node *i* might want to download more descriptions for higher quality video). Therefore, download rate limitations must be enforced by some other means, which is the motivation for our kantoku framework.

3.2. P2P Lookup Model

Our framework assumes that the underlying p2p routing substrate can map any given key in a specified identifier range to a single corresponding node based on node identifiers from that same range. The particular p2p lookup protocol used determines both routing as well as which nodes are responsible for storing which keys. Although several types of p2p systems might be suitable for our purposes, we note that distributed hash tables (DHTs) for structured peer-to-peer systems provide this needed functionality [15,16,17]. At least one p2p media streaming system has already been implemented on top of a DHT [1,15].

3.3. Attack Model

Our attack model considers attacks that can lead to bandwidth exhaustion. For example, selfish peers might request more descriptions than their bandwidth limitations allow in order to receive higher quality video or audio. Malicious peers might want to

[1] In Japanese, the term *kantoku* roughly translates to "overseer."

overwhelm the system by initiating a large number of sessions in a short amount of time. Using our application model as a reference, a selfish or malicious node *j* could make several requests to different nodes such that D_j is greater than $Dmax_j$, but for each individual node *i* that is uploading to node *j*, $r_{ij} \leq Dmax_j$. Consider Example 1 below, which is based on Figure 1 (all rates are in Kbps).

Example 1
$r_{WP} = 350$, $r_{XP} = 350$, $r_{YP} = 350$, $r_{ZP} = 350$
$Dmax_P = 500$, $D_P = 1400$

In Example 1, none of the uploaders (i.e., *W*, *X*, *Y*, or *Z*) can locally determine that downloader *P* is downloading at a rate almost three times its maximum download rate allowed. Even if given $Dmax_P$, each uploader *U* only knows r_{UP} and therefore cannot determine that *P* is in violation (i.e., $D_P > Dmax_P$).

Kantoku is designed to throttle selfish and malicious downloaders. Currently, malicious uploaders (e.g., nodes intentionally sending bogus content) are not considered, but will be included in future work.

3.4. Protocol

Our kantoku protocol handles attacks that attempt to exhaust all of the available upload bandwidth in the system. Basically, our kantoku nodes collectively monitor the download rate of each untrusted peer in the system and use this information to throttle misbehaving peers.

3.4.1. Assigning Untrusted Nodes to Kantoku Nodes

In addition to the p2p media streaming application's overlay network, kantoku nodes form their own separate p2p overlay network. Creation and maintenance of both overlay networks is done by the underlying p2p routing layer [15,16,17]. The kantoku architecture is depicted in Figure 2.

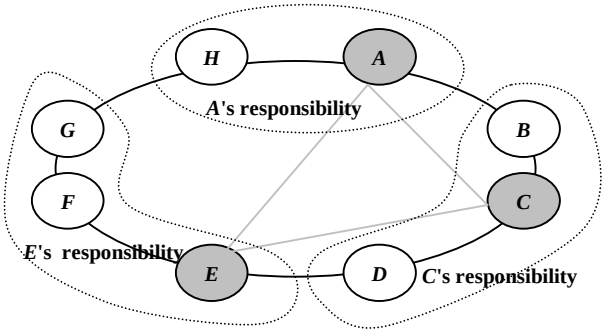

Figure 2. Kantoku architecture

Figure 2 includes untrusted nodes (shown in white) and kantoku nodes (shown in gray). The underlying application overlay (shown in black) runs in parallel with the kantoku overlay (shown in gray). Untrusted node assignments to kantoku nodes are indicated by the dashed outlines. Based on the p2p lookup protocol used, we can assign each untrusted node to a kantoku node by treating the untrusted node's identifier from the application overlay as an object key in the kantoku overlay as explained in [13]. In this example, untrusted nodes are assigned to the closest kantoku node in the virtual ring based on their identifier.

3.4.2. Selecting Kantoku Nodes

One major challenge in our framework is how to select nodes trustworthy enough to serve as kantoku nodes. One possibility is to use one of the reputation mechanisms that appears in the literature with kantoku nodes chosen among nodes whose reputation exceeds a certain threshold [19,20]. Another possibility would be for the p2p media streaming service provider to provide some kantoku nodes with trusted hardware. Unlike a central server setting, the service provider can adjust the number of trusted peers as necessary. Kantoku node selection strategies are part of our future work.

3.4.3. Monitoring Peer Download Rates

Kantoku nodes may upload and download media streams, but they have the additional responsibility to monitor the current download rate of the untrusted nodes. In order to initiate or terminate a media streaming session, each untrusted node must send all control messages through its assigned kantoku node that will certify the control messages (via a digital signature) and forward those certified messages to the peer that will do the actual uploading. Uploading nodes will ignore all control messages that have not been certified by a kantoku node. The identity of kantoku nodes can be verified with a digital certificate associating trusted status with a kantoku node's identifier and public key.

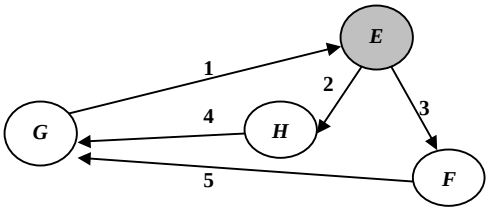

Figure 3. Initiating a kantoku-monitored session

Having observed these control messages, the kantoku nodes can update each untrusted peer *P*'s current download rate D_P and a list $Uploaders_P$ of peers from which *P* is currently downloading. Whenever a kantoku node receives a control message from an untrusted node for which it is not responsible, the kantoku will forward the message to the kantoku node that is responsible for maintaining that untrusted node's download information using p2p lookup scheme for the kantoku overlay network. Upon receiving a control message from a node for which it is responsible, a kantoku node will compute the probability $DropProb_P$ with which any packet in one of *P*'s downloading sessions should be dropped. $DropProb_P$ will then be forwarded to all nodes appearing in $Uploaders_P$. All nodes receiving $DropProb_P$ will probabilistically drop packets destined for peer *P* with probability equal to $DropProb_P$. Determining $DropProb_P$ for each untrusted peer *P* is explained in Section 3.4.4. Figure 3 shows an example of how monitoring is accomplished, based on the scenario depicted in Figure 2, where node *G* would like to initiate a streaming session with node *H*. Also, assume that *G* is already currently downloading from node *F*. The steps labeled in Figure 3 are listed below.

1. *G* wants to download stream from *H* and sends START session control message to kantoku node *E* to which G is assigned.
2. Using D_G and $Dmax_G$, *E* computes $DropProb_G$ and piggybacks $DropProb_G$ on certified START control message forwarded to *H*.
3. Using $Uploaders_G$, *E* sends newly updated $DropProb_G$ to all nodes currently uploading to *G* (i.e., *F* in this example). *E* updates $Uploaders_G = Uploaders_G \cup H$.
4. *H* begins to stream media to *G* with packet dropping probability equal to $DropProb_G$.
5. *F* continues to stream media to *G*, but with newly updated packet dropping probability equal to $DropProb_G$.

The procedure for terminating a session follows similar steps. We should note that all control messages are digitally signed to prevent malicious nodes from spoofing session initiation requests that would make an innocent node appear to be an attacker. Next, we explain how to compute drop probability *DropProb*.

3.4.4. Computing Packet Dropping Probability

An integral part of our protocol is computing the packet dropping probability, $DropProb_P$, for all sessions belonging to a particular downloading peer *P*. $DropProb_P$ allows us to aggressively reduce traffic from selfish and malicious peers trying to exceed their download rate limit.

We use an idea from core-stateless fair queueing to compute $DropProb_P$ for each untrusted peer *P*. Core-stateless fair queueing (CSFQ) was originally developed in the context of networking to achieve fair bandwidth allocation for a contiguous network region consisting of edge routers and core routers [18]. Edge routers maintain per-flow state and label packets belonging to a particular flow with the estimated rate *r* of that flow. Using *r* and a locally estimated fair share rate α, core routers probabilistically drop packets according to the algorithm presented in [18], which is designed to drop packets from flows exceeding their estimated fair share with higher probability. Since core routers do not maintain per-flow state, they are considered to be stateless.

We can view kantoku nodes as trusted nodes that maintain per-downloader state (i.e., D_P and $Dmax_P$) for each untrusted node *P* assigned to it. Based on this per-downloader state, kantoku nodes compute $DropProb_P$ and notify all uploaders to *P* that they should drop all packets in a stream destined for *P* with probability equal to $DropProb_P$. Equation 1 shows how we compute packet dropping probabilities.

$$DropProb_p = \max\left(0, \beta \cdot \left(1 - \frac{Dmax_p}{D_p}\right)\right) \quad (1)$$

Rather than using an estimated fair share rate as done in [18], we use the maximum download rate allowed $Dmax_P$. Also, we added a parameter β, where β is real number greater than 0, which allows us to tune how aggressively we want to throttle the sessions of users exceeding their download rate maximum limits. Higher values of β lead to more aggressive throttling. Referring to Equation 1, we can see that nodes not exceeding their download rate maximum will have a *DropProb* equal to zero. However, nodes exceeding their maximum download rate will have a *DropProb* proportional to the amount by which they exceed their download limit.

An alternative strategy that we could have used would be to completely cut off any peer *P* that attempts to exceed its download rate limit $Dmax_P$. However, we feel that such a strategy is too harsh because it does not give us the flexibility to distinguish between slight selfishness (e.g., D_P exceeding $Dmax_P$ by 1 Kbps for some peer *P*) and malicious attacks (e.g., a DoS attack with D_A greater than 100 times $Dmax_A$ for some

attacker *A*). Kantoku punishes attackers according to their level of abuse.

4. Performance Evaluation

To evaluate the kantoku framework, we ran several simulations of a p2p media streaming application with various numbers of kantoku and untrusted nodes participating. In each simulation, between 20% and 25% of the nodes were malicious attackers executing a DoS attack. The simulation parameters corresponding to Equation 1 from Section 3.4.4 were the following: $\beta = 1.75$ and $Dmax_P = 700$ Kbps for each peer *P*.

At random times throughout the simulation, each well-behaved node would initiate a single 30-second streaming session at 50% of its maximum allowed download rate. At the beginning of each simulation, each attacker would request five streams for the entire simulation duration totaling 250% of its maximum allowed download rate. Attackers and well-behaved nodes used the same pool of uploaders to ensure that kantoku would be evaluated under conditions when upload bandwidth is scarce.

The metric we used to evaluate kantoku's effectiveness was the fraction of content packets received by the downloader (i.e., packets that were not dropped by the uploader due to upload bandwidth capacity constraints). We refer to this metric as the *quality* received by the downloader.

We ran three types of simulations. In *None*, no effort was made to try to limit the download rate of participating nodes. In *Rate Limiting* (*RL*), attackers attempting to initiate sessions that put them over their download rate limit were denied the additional session. In *kantoku*, we follow the algorithm described in Section 3.4. Our results for various scenarios (i.e., 500-node and 1000-node networks) appears in Table 2.

k = number of kantoku nodes
g = number of well-behaved nodes
m = number of malicious nodes
type = type of simulation
gq = quality received by well-behaved nodes

k	***g***	***m***	***type***	***gq***
10	368	122	*None*	0.668
10	368	122	*RL*	0.691
10	368	122	*Kantoku*	0.998
50	713	237	*None*	0.695
50	713	237	*RL*	0.735
50	713	237	*Kantoku*	0.995

Table 2. Effectiveness of kantoku on quality received

As shown in Table 2, *kantoku* outperforms both *None* and *RL*. Compared to *None* and *RL*, *kantoku* improves the quality of well-behaved nodes, which is our goal. *None* performs poorly because nothing is done to prevent abuse, which allows attackers to easily exhaust the upload bandwidth available. The reason that *kantoku* outperforms *RL* is because *kantoku* is punitive in the sense that abusers are punished proportionally to their level of abuse meaning that attackers might be throttled to a level below their maximum allowed rate, which makes more bandwidth available for well-behaved peers. *RL* simply limits all nodes to their maximum download rate, but still allows aggressive attackers to continue downloading at this maximum rate despite their misbehavior.

RL could be augmented to drop all attackers' streaming sessions as soon as they attempt to exceed their limit, but this approach might be too harsh and inflexible as we discussed in Section 3.4.4.

To further support our findings, Figure 4 shows the aggregate data rate received by well-behaved nodes over time in our 1000-node simulation run for our simulations of *RL* and *kantoku*. With *kantoku*, well-behaved nodes clearly received higher data rates overall.

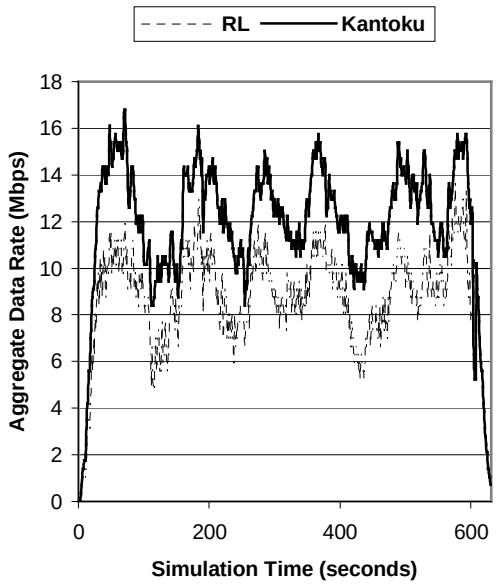

Figure 4. Aggregate data rate received by all well-behaved peers in the system

5. Conclusion

We have presented our kantoku framework, which uses a subset of trusted nodes to effectively monitor the

download rate of all untrusted peers in the system and throttle misbehaving peers accordingly. We have evaluated our framework through simulation and initial results indicate that kantoku is very effective at ensuring that well-behaved nodes receive their fair share of available upload bandwidth while punishing attackers appropriately.

References

[1] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava. "PROMISE: peer-to-peer media streaming using CollectCast." *11th ACM Conference on Multimedia*, 2003.

[2] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. "Distributing streaming media content using cooperative networking." *12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2002.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. "SplitStream: high-bandwidth multicast in a cooperative environment." 19th ACM Symposium on Operating Systems Principles, 2003.

[4] Y. Cui, B. Li, and K. Nahrstedt. "oStream: asynchronous streaming multicast in application-layer overlay networks." IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1, January 2004.

[5] X. Zhang, J. Liu, B. Li, and T.-S. Yum. "CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming". *IEEE INFOCOM*, 2005.

[6] N. Magharei and R. Rejaie. "PRIME: peer-to-peer receiver-driven mesh-based streaming." *IEEE INFOCOM*, 2007.

[7] N. Daswani and H. Garcia-Molina. "Query-flood DoS attacks in Gnutella." *9th ACM Conference on Computer and Communications Security*, 2002.

[8] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. "A blueprint for introducing disruptive technology into the Internet." *1st ACM Workshop on Hot Topics in Networking*, 2002.

[9] B. Chun and T. Spalink. "Slice creation and management." PlanetLab Design Note 03-013, July 2003.

[10] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. "SHARP: an architecture for secure resource peering." *19th ACM Symposium on Operating Systems Principles*, 2003.

[11] A. Nandi, T.-W. Ngan, A. Singh, P. Druschel, and D. Wallach. "Scrivener: providing incentives in cooperative content distribution systems." *Middleware*, 2005.

[12] N. Liebau, V. Darlagiannis, A. Mauthe, and R. Steinmetz. "Token-based accounting for p2p-systems." *Kommunikation in Verteilten Systemen (KiVS)*, 2005.

[13] W. Conner, K. Nahrstedt, and I. Gupta. "Preventing DoS attacks in peer-to-peer media streaming systems." *13th Annual Conference on Multimedia Computing and Networking*, 2006.

[14] J. Jung, B. Krishnamurthy, and M. Rabinovich. "Flash crowds and denial-of-service attacks: characterization and implications for CDNs web sites." *11th International World Wide Web Conference*, 2002.

[15] A. Rowstron and P. Druschel. "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems." *Middleware*, 2001.

[16] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. "Chord: a scalable peer-to-peer lookup service for internet applications." *ACM SIGCOMM*, 2001.

[17] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. "Kelips: building an efficient and stable p2p DHT through increased memory and background overhead." *2nd International Workshop on Peer-to-Peer Systems*, 2003.

[18] I. Stoica, S. Shenker, and H. Zhang. "Core-stateless fair queueing." IEEE/ACM Transactions on Networking, Vol. 11, No. 1, February 2003.

[19] E. Damiani, S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. "A reputation-based approach for choosing reliable resources in peer-to-peer networks." *9th ACM Conference on Computer and Communications Security*, 2002.

[20] S. Marti and H. Garcia-Molina. "Identity crisis: anonymity vs. reputation in p2p systems." *3rd IEEE International Conference on Peer-to-Peer Computing*, 2003.

[21] X. Fu and J. Crowcroft. "GONE: an infrastructure overlay for resilient, DoS-limiting networking." *16th ACM International Workshop on Networking and Operating Systems Support for Digital Audio and Video*, 2006.