

# IMP: ISP-Managed P2P

Shakir James and Patrick Crowley  
Department of Computer Science and Engineering  
Washington University  
St. Louis, MO 63130-4899  
E-mail: {sjames, pcrowley}@wustl.edu

**Abstract**—Internet Service Providers (ISPs) have failed to reduce the cost of peer-to-peer (P2P) traffic. Traffic throttling devices increase user download times, and caches store content that may infringe copyright. We propose ISP-Managed P2P (IMP): a transparent peer-discovery service that returns peers favorable to ISPs. Unlike similar services, IMP does not require the direct support of developers who have no incentive to cooperate. This paper covers the design, implementation, and experimental evaluation of our IMP prototype, which reduces costly, cross-ISP traffic by eight times without significantly increasing user download times.

## I. INTRODUCTION

Peer-to-peer (P2P) applications provide cheap scalability for content providers at the expense of high network costs for Internet Service Providers (ISPs). Content providers do not need to invest in large data centers because peers share – i.e., upload and download – data with each other. ISPs, however, pay for P2P’s self-scalability. Peers increase costly, cross-ISP traffic since they connect to each other without regard for ISPs’ peering agreements and routing policies [1, 3, 31].

ISPs and P2P developers play a game of *cat and mouse*. Traffic throttling devices reduce costs for ISPs but increase download times for users [3]. ISPs first scanned for data traffic on well-known port numbers, and then they used deep packet inspection (DPI) when developers switched to random ports. In turn, developers encrypted messages to bypass DPI. Today, with the legality of P2P throttling in question, ISPs and federal regulators play the game in court [25].

Traffic throttling also hurts ISP’s relationship with P2P developers. Instead of working with ISPs to reduce costs – via protocol changes [7, 14] or collaboration [1, 31] – developers are encrypting more traffic [13, 11, 21]. ISPs must take independent, unobtrusive action to regain developer trust [18]. ISP-Managed P2P (IMP) provides such an approach: it reduces costly, cross-ISP traffic and maintains user download times.

IMP is an ISP-friendly, transparent, peer-discovery service. The application *tracker* returns a random set of peer addresses. IMP, however, responds with addresses beneficial to an ISP traffic-engineering objective such as reducing cross-ISP traffic [18]. Moreover, unlike similar services [1, 31], IMP does not require developers to change their protocol. This approach is important because many applications implement variations of established P2P protocols [4, 8].

IMP is also developer-friendly. Parameters in application messages allow developers to detect and override IMP. The ISP Oracle’s response contains a flag that notifies developers

of IMP, and developers may include a flag in peer-discovery requests that bypasses IMP. Note that these flags do break existing applications. Thus, to *opt-out*, developers can modify their clients to monitor ISP action and override it.

Our work makes four contributions. First, we design a plug-and-play, peer-discovery service. Second, we implement a prototype for BitTorrent [9], a popular file-sharing protocol. Third, we evaluate our prototype on an experimental platform that features tens of routers and hundreds of hosts [17]. Last, our work demonstrates that ISPs can independently reduce P2P network costs without reducing application performance or risking copyright infringement.

The rest of this paper details our work. Section II presents necessary background on BitTorrent and a motivating example. We describe IMP’s design in Section III, a network processor based prototype in Section IV, and its experimental evaluation in Section V. We finish with a survey of related work in Section VI and concluding remarks in Section VII.

## II. BACKGROUND

We describe BitTorrent’s tracker protocol then explain how IMP acts as a proxy for the BitTorrent tracker.

### A. BitTorrent Tracker Protocol

The BitTorrent tracker is a peer-discovery service. A peer needs to find a tracker before it can join a *swarm*, a group of peers sharing the same file. Users download a *metainfo file* from a website to obtain a tracker’s web address. The metainfo file also contains an info field that describes the file of interest. Peers compute an *infohash*, a SHA1 hash of the info field.

A BitTorrent *client* uses the infohash and tracker protocol to query a tracker for peer addresses in a swarm. The client connects to some of the peers in a tracker’s response. These peers are called its *neighbors*. After downloading its first piece of the file, it uploads pieces it has in exchange for those it wants. *Seeds* have the complete file and *leechers* are still downloading pieces.

IMP is plug-and-play because it uses the tracker protocol. Many clients make incompatible extensions to the BitTorrent “protocol”, which they gleaned from the source code of the *mainline* client, the first open-source client [4]. All clients, however, implement a basic set of features to interoperate [9]. The tracker protocol constitutes part of this de facto standard. Therefore, IMP works with all BitTorrent clients.

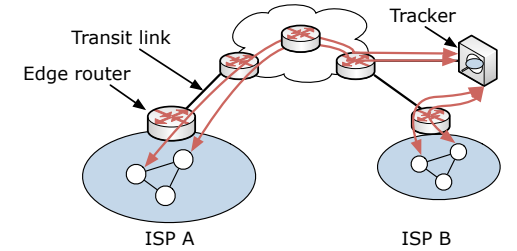
### B. Motivating Example

Fig. 1a shows a tree topology with two ISPs: A and B. The ISPs partition the swarm into two ISP-local subsets. Since ISPs’ own their network links, local traffic is cheaper than cross-ISP traffic on the transit link. One tracker monitors the swarm that all clients join. Since all peers have an equal chance of appearing in the tracker’s response, the tracker ignores the network topology and ISP economics.

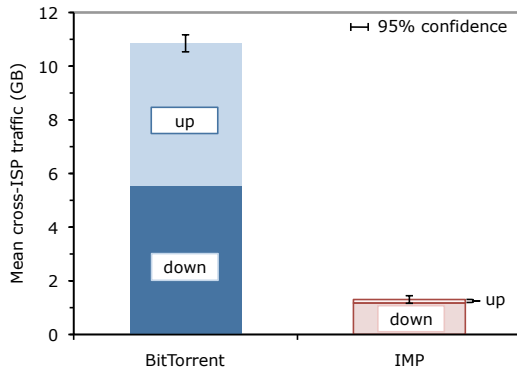
Consider a situation where ISP A aims to reduce cross-ISP traffic. ISP A installs IMP alongside its edge router to achieve this objective. IMP identifies and redirects tracker requests to its ISP Oracle component. Instead of a random subset from the tracker, the ISP Oracle’s response includes network-local peers and one *remote*, non-local, peer [3]. Remote peers help maintain a robust overlay if, for example, all seeds run on ISP B’s network [18].

Fig. 1b shows that IMP offers an eightfold reduction in cross-ISP traffic. In this experiment, 200 leechers share a 100 MB file with 100 leechers on each ISP and a seed on ISP B’s network. The graph shows the average of five measurements for IMP and BitTorrent. A swarm, which uses IMP, downloads 12 copies of the file and uploads one copy over the transit link. A BitTorrent swarm, however, downloads 57 copies and uploads 54.

This example uses a remote tracker. For a local tracker, IMP can monitor remote peers instead of local ones – intercept queries in the reverse direction. If a tracker runs on ISP A’s network, IMP identifies requests from ISP B’s peers and responds with mainly other remote peers. Likewise, ISP B can also install IMP on its network. Our current prototype works for remote trackers and aims to localize traffic.



(a) Tree topology with two ISP: A and B.



(b) Cross-ISP traffic on ISP A’s transit link.

Figure 1. An initial experiment.

### III. DESIGN

We explain how IMP’s components work together, detail its individual components, and outline provisions for other P2P protocols.

#### A. System Overview

Fig. 2 shows how IMP’s components interact. External communication occurs via three interfaces: Router, P2P client, and P2P tracker. IMP talks to the tracker through the P2P tracker interface. The P2P client interface sends responses to client applications, and the Router interface receives incoming packets from the router. All components communicate over UDP channels, which make our design extensible.

Each component performs a specific function. The P2P Enhanced Edge Router (PEER) detects query messages and forwards data to the Proxy Cache and Network Address Translation component (NAT) stores packet headers for IP masquerading. The Proxy Cache queries the ISP Oracle on behalf of clients and harvests non-local, peer addresses from the tracker on behalf of ISP Oracle. It also passes the ISP Oracle’s response to clients via the NAT.

#### B. P2P Enhanced Edge Router

PEER uses port scanning and string matching to identify client requests. It first examines the port number and transport protocol of incoming packets. Next, it compares parts of the message payload with constant bit patterns. If any match fails or an override-flag exists, the message passes through. If not, PEER encapsulates the payload and sends it to the Proxy Cache and sends the packet headers to the NAT.

PEER only forwards requests from large swarms. This reduces the messaging overhead and processing requirements of routers. It counts the number of requests received from different IP addresses for an infohash to estimate swarm size. If the counter exceeds an empirical threshold, PEER starts to forward requests to the Proxy Cache and, if necessary, closes the tracker’s side of the connection.

As an example, consider PEER for BitTorrent clients. First, the filter scans for TCP packets on port 6969. Then, it matches the bit strings: `HTTP` and `GET /announce`. A match identifies the message as a client request. If no developer-override flag (`&isp=0`) exists, PEER uses the request’s infohash to look up and increment the swarm’s request counter. If the counter exceeds the threshold, it sends a TCP RST to the tracker.

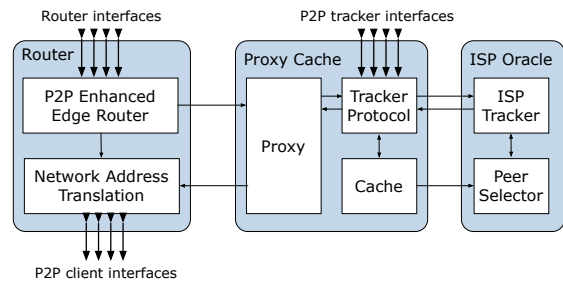


Figure 2. Organization of IMP’s components.

### C. Proxy Cache

The Proxy Cache uses data from PEER to act as a proxy for the ISP Oracle and tracker. The Proxy forwards request data to the Tracker Protocol subcomponent, which uses the tracker protocol to query the ISP Oracle for peer addresses on behalf of the client. The Proxy also forwards the ISP Oracle’s reply to clients via the NAT, which uses data previously received from PEER to rewrite the message’s source address.

The Tracker Protocol also periodically queries the tracker on behalf of the ISP Oracle. The ISP Oracle’s response includes fewer non-local addresses than local ones, so the Tracker Protocol does not need to query the tracker every time it receives a message from the Proxy. And, more important, periodic-querying avoids flooding the trackers. The Tracker Protocol passes the infohash and tracker address to the Cache, which controls its periodic harvesting.

The Cache initiates a tracker query for two reasons: to store a new set of non-local addresses or to update an existing set. It requests new addresses for an unknown tracker or swarm: a new tracker address or infohash. It also updates the peer addresses of its most recently used entries every cache refresh interval. In either case, the Cache sends the addresses to the ISP Oracle via an out-of-band channel.

### D. ISP Oracle

The ISP Oracle responds to Proxy Cache queries via the ISP tracker. Since they use the regular tracker protocol, the ISP tracker works as clients expect. The ISP tracker supports other peer requests such as periodic *keep-alives* and *graceful shutdowns*. Also, in addition to the new notification flag, the ISP tracker’s response includes regular statistics such as the number of leechers and seeds in the swarm.

The Peer Selector implements the ISP’s traffic engineering objective. It determines which peer addresses to include in the ISP tracker’s reply. It receives ISP-local addresses from the ISP tracker and non-local ones from the Proxy Cache. Then it chooses addresses on behalf of the ISP tracker. Our prototype implements a simple localization objective; except for one, all addresses are local [3]. Other objectives are also possible [31].

### E. Protocol Provisions

It is easy to add support for other protocols. We isolate the three protocol-specific subcomponents: Tracker Protocol, ISP tracker, and PEER’s filter. The four other subcomponents are protocol agnostic and communicate with the protocol-specific ones via UDP sockets. The Tracker Protocol and ISP tracker are software subcomponents that use the regular tracker protocol. Thus, two new subcomponents that use different UDP ports can be added on.

PEER’s filter, another protocol-specific subcomponent, uses a smaller fraction of the tracker protocol. It identifies only tracker requests, which are plaintext and part of the de facto standard. Developers tend to encrypt data traffic since identification techniques focus on messages that generate the bulk of network traffic [16]. Even if clients add extensions such as the Tracker Peer Obfuscation [2], they all need to support the basic request message format to avoid fragmentation.

IMP, however, cannot work if developers encrypt tracker requests. This is difficult because it will break all existing applications, and we contend that they have no reason to start since they can easily override IMP with a flag.

## IV. IMPLEMENTATION

We state our requirements of a developmental platform; describe features of a network processor (NP), a suitable choice; and explain how our prototype augments a NP-router.

### A. Considerations

We select a development platform based on three criteria: packet-processing rate, development time, and availability of an associated experimental platform. Application Specific Integrated Circuits, Field-Programmable Gate Arrays, and Network Processors (NPs) offer high-throughput packet processing, but programmable NPs allow the most rapid development-test cycle. The Open Network Lab (ONL) also offers an experimental platform with NP-routers (NPRs) to experimentally evaluate our prototype.

Our implementation uses ONL’s NPR together with a Linux host. NPs process packets faster than general-purpose processors (GPPs) [23], but the ease of programming protocol idiosyncrasies outweighs speed considerations. We implement IMP’s router extensions, PEER and NAT, for the IXP 2800 NP because the NPR runs on the IXP [30]. PEER and NAT use common networking operations such as packet matching and hash functions. Therefore, IMP uses commodity-networking hardware.

The Proxy Cache is an event-driven, multithreaded Python application, and the ISP Oracle is a modified BitTorrent tracker application [15]. This rest of this section, however, focuses on the noteworthy aspects of our NP implementation.

### B. Network Processor Features

We briefly explain how IMP uses the high-performance features of NPs like the IXP; other authors [23, 30] detail the IXP architecture. To overcome the gap between processor and memory speeds, IMP uses the IXP’s multiple processing cores and hardware support for multithreading. Each of the IXP’s 16 Mirco-Engines (MEs) supports eight thread contexts that IMP uses for packet processing. When a thread stalls waiting for memory, IMP keeps the MEs busy by quickly switching to another thread in a round-robin manner.

Additionally, to improve IO performance, IMP uses the IXP’s four memory interfaces: ME-local, Scratch, SRAM, and DRAM. DRAM offers the largest capacity under penalty of the largest access latency. At the opposite end, local memory has the least capacity and lowest latency. IMP uses DRAM for packet buffers and SRAM for smaller, hash tables. ME-local memory holds frequently accessed data such as thread-level mutexes and hash keys.

NPs also offer hardware accelerators that speed up common networking tasks. The IXP 2800 includes a Hash unit for hash functions, cyclic redundancy check (CRC) unit for checksums, and ternary content addressable memory (TCAM) for associative array lookups. IMP uses the TCAM to match

incoming packets, the Hash unit to lookup request counters, and the CRC unit to calculate checksums for message headers.

### C. NP Router Extension

IMP extends the NPR data plane. The NPR’s plugin environment allows users to dynamically load code onto MEs and direct packets to them. Fig. 3 shows the NPR’s data flow with IMP’s plugins. The PEER and NAT plugins use two MEs, access packet buffers in DRAM, and share a region of SRAM for inter-ME communication. Thus, IMP does not alter the core router.

The Multiplexer (Mux) block receives outgoing packet pointers from PEER and NAT, but an incoming packet starts at the Receive block (Rx). Rx stores the incoming packet data in DRAM and, like the other blocks, passes a packet pointer to the next block. Mux multiplexes packets into the main pipeline, and Parse, Lookup, and Copy (PLC) passes along the pointer or drops the packet.

PLC determines a packet’s destiny. It uses the header as a lookup key to match user-defined filters or routes in the TCAM. PEER’s first-stage filter, a user-defined filter, directs packets to its plugin ME. Other packets, en route to an external link, the Queue Manager (QM) places its pointer on an interface queue, Header Format (HF) adds Ethernet header data, and Transmit (Tx) transfers the packet from DRAM to the external link.

### D. PEER Request Counter

PEER’s Request Counter monitors swarm size, the number of peers in a swarm. Recall that PEER forwards requests from swarms that exceed a threshold size. Thus, the Request Counter counts requests from distinct IP addresses for a value of the infohash parameter. Fig. 4 shows the data structure that uses an arbitrary, numeric protocol identifier to select a hash table, and a hash of the infohash or content identifier (cid) to determine the swarm or table entry.

A table entry also contains a counter and peer set that stores the known addresses in the swarm. The peer set is a bloom filter [5]. (Simple bloom filters do not support deletion, but we plan to use an eviction policy to flush table entries.) Initially, the bloom filter’s bits are set to zero. A peer exists in the set if its address hashes to bits that are all one. Otherwise, for an unknown address, we set the bits (to add the address) and increment the counter.

Bloom filters like hash tables offer constant time lookups.

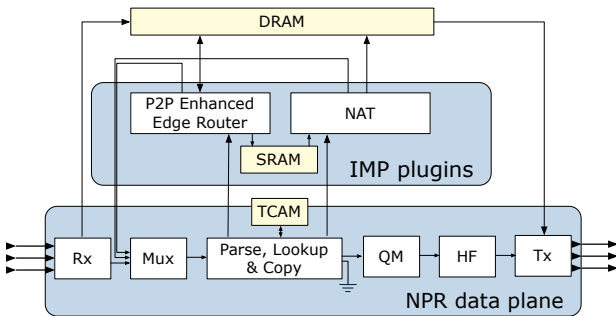


Figure 3. Extension of ONL router’s data plane.

Filters, however, use less space at the expense of false positives. A 9KB bloom filter, with 6 hash functions (the optimal number) and a less than 0.5% false positive rate, can store 7,500 IP addresses: 33 times less memory than storing 32-bit addresses.

## V. MEASUREMENTS

We perform three sets of experiments to systematically evaluate our prototype [12]. First, we study a wide range of parameters at two values to determine the factors that explain the majority of variation in performance. Second, we study these important effects—the tracker and its interaction with locality and topology—over a wide range of values. Last, we demonstrate that IMP is plug-and-play.

### A. Experimental Methodology

We state our goals and then select metrics, a workload, and parameters to vary. We also describe how we instrument our prototype to measure performance.

1) *Goals*: We aim to compare the cross-ISP traffic and client performance of a swarm using the BitTorrent tracker with or without IMP. Based on this goal, our system consists of four components: local peers, remote peers, transit links, and a tracker. The tracker, the key component under study, is the BitTorrent tracker alone, or assisted by IMP’s ISP Oracle. Local peers run on hosts inside the ISP’s network, and remote peers run on external hosts. Cross-ISP traffic traverses the transit links that connect local peers to remote ones.

2) *Metrics*: Cross-ISP traffic measures IMP’s benefit for ISPs, whereas download time, sharing ratio, and not-interested frequency measure its utility for users. The definitions of these performance criteria follow:

- Cross-ISP traffic: sum of the bytes transferred over the ISP’s transit links.
- Download time: interval between requesting the first piece of the file and downloading the last piece.
- Sharing ratio: number of bytes uploaded divided by the number downloaded. (Fairness)
- Not-interested frequency: number of times a peer receives a message from one its neighbors indicating that they have the same pieces of the file. (Piece diversity)

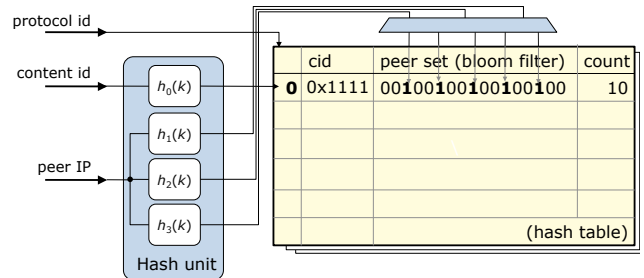


Figure 4. Request Counter’s data structure.

3) *Workload*: We select a flash crowd workload because it exerts the most pressure on the transit links [3]. The initial release of a popular file creates a rush of leechers called a flash crowd. Our scripts model a flash crowd by spawning an initial seed then immediately starting leechers. For each experiment, we use a pseudo-random file and start leechers in a random order. Leechers stay connected as seeds after downloading the complete file. (We verify that whether a leecher stays on as a seed or disconnects immediately has negligible impact on cross-ISP traffic in a flash crowd.)

4) *Parameters*: We fix other system parameters. We use the worst-case bandwidth allocation policy for client download rates [7]; Choffnes and Bustamante showed that a local-peering strategy leads to the lowest download rates when the access bandwidth is lower than the transit one. Thus, we set our access link bandwidth to 500 MB/s and the other links, including transit links, to 1 GB/s. The speed of ONL end hosts' Ethernet interface is also 1 GB/s.

5) *Factors*: Table I shows the network configurations we use in our experiments. Each edge router where IMP runs has a different number of transit links. In addition to topology, we vary five parameters [3, 31]:

- Tracker: BitTorrent (BT) tracker or ISP Oracle (IMP).
- Filesize: the size of file (MB).
- Seed location: the initial seed is either local or remote.
- Swarm size: the total number of leechers in the swarm.
- Locality: the ratio of local leechers to swarm size.

6) *Instrumentation*: ONL's Remote Laboratory Interface (RLI) allows us to instrument our prototype. Using the RLI, we construct network topologies and monitor traffic. We measure cross-ISP traffic as the sum of bytes sent (upstream) and received (downstream) at the router's ports that connect to the transit links. Similarly, to set link bandwidth, we change the associated port rates. (The NPR's QM limits the "long-term" outgoing port rate.) To measure client metrics, we process the log files from an instrumented version of the mainline client v4.0.2 [15] with a 40 kB/s upload rate and unlimited download rate.

TABLE I. SUMMARY OF NETWORKS TOPOLOGIES.

Topology	# Routers	# Links		# Hosts
		transit	other	
Tree	10	1	34	27
Ring	8	2	34	27
Star	10	3	32	27
Mesh	10	4	34	27

### B. Important Effects

We efficiently determine which of our six factors account for the most variation in cross-ISP traffic. We use a fractional

factorial experimental design ( $2^{k-p}$ ) where each of our ( $k = 6$ ) factors assumes two *levels*, alternative values. This estimates the effects, which is the impact of the factors and their interactions, but requires one-quarter ( $p = 2$ ) of the number of experiments in a full factorial design ( $2^6$ ).

Table II shows our choices for factor levels. Disk space limits our filesize levels, and the number of ONL hosts dictates swarm sizes. We increase the effective location size by running 14 mainline clients per host. (We verify that up to 16 mainline clients can be multiplexed on a host without a noticeable variation in our client metrics.) Although these swarm sizes are smaller than Internet-scale one, they represent swarms on smaller networks such as a university network. The levels of locality and topology represent a high and low number of local leechers and transit links.

The effect of the tracker and its interactions is most important. Fig. 5 shows that it explains 82% of variation in cross-ISP traffic. In particular, its interaction with locality and topology account for one-fifth of the total. This confirms our intuition that cross-ISP traffic depends on the fraction of local leechers and the number of transit links. The effect of filesize also appears important. (Filesize also accounts for 99% of the variation in median download times.) The effect of swarm size and locality both account for 2% of variation in cross-ISP traffic. The last two factors, seed location and topology, account for 1% and 0% of the variation. Thus, three effects warrant further study: the tracker, its interaction with locality, and its interaction with topology.

TABLE II. SIX FACTORS AND THEIR TWO LEVELS

Factor	Levels	
	-1	1
Tracker	BitTorrent (BT)	ISP Oracle (IMP)
Filesize	100MB	400MB
Swarm size	100	400
Topology	tree	mesh
Seed location	local	remote
Percentage locality	20 % (low)	80 % (high)

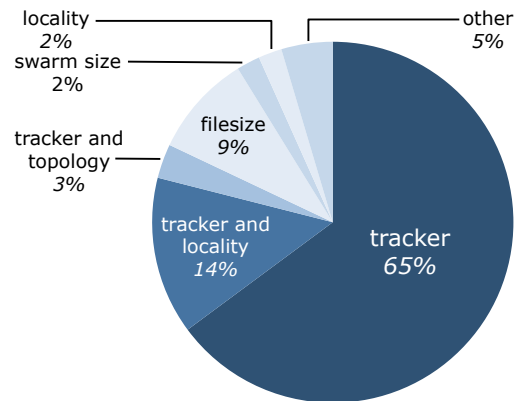


Figure 5. Effects and variation in cross-ISP traffic they explain.

### C. Tracker

To compare the two trackers, we vary the type of tracker and fix other factors: 100 MB file, 100 local and 100 remote leechers, a remote seed, and a tree topology. Our ONL experiment uses 50 hosts and 10 NP-routers. We report the average of five repetitions. And, as in all our experiments, we install one instance of the IMP on the edge router where we measure traffic.

IMP also maintains client download times. Fig. 1b (in Section II) shows that IMP provides an 8x reduction in cross-ISP traffic for ISPs, and Table III shows a negligible increase in median download times for users. IMP clients take 12 seconds longer on average than BT ones to download all pieces of the file. Since BT clients and their neighbors have different pieces of the file 1,100 times more often, they share 17 % more pieces. This higher sharing ratio, however, should result in a larger difference between BT and IMP client download times.

We use an analysis of paired observations to test if these differences are significantly different from zero. Table III shows that BT and IMP with 95% clients have the same median download times. However, there is only a 5% chance that the observed differences in the swarm’s sharing ratio and not-interested frequency are due to experimental error. Since IMP returns a smaller fraction of non-local peers than BT and our experiment uses an initial, non-local seed, it seems natural for local clients to have more identical pieces and share less. Therefore, even if swarm download times are the same, we expect that local download times should surely increase.

TABLE III. BT AND IMP PERFORMANCE FOR ALL CLIENTS.

Metric	Median performance comparison	
	Mean difference	95% confidence interval
Download time (s)	-12	(-58, 34) <sup>a</sup>
Sharing ratio	0.17	(0.15, 0.19)
Not-interested freq.	-1100	(-1300, -790)

a. Not significant.

Fig. 6 confirms our suspicion that IMP increases local client download times. Local clients share less and have higher download times. The higher diversity of pieces among IMP’s remote clients, which are more likely to find the seed, leads to a higher sharing ratio and a substantially lower download time. Therefore, IMP does not hurt client download time overall because remote clients compensate for local ones. At the same time, the higher remote sharing ratio (and IMP’s lower

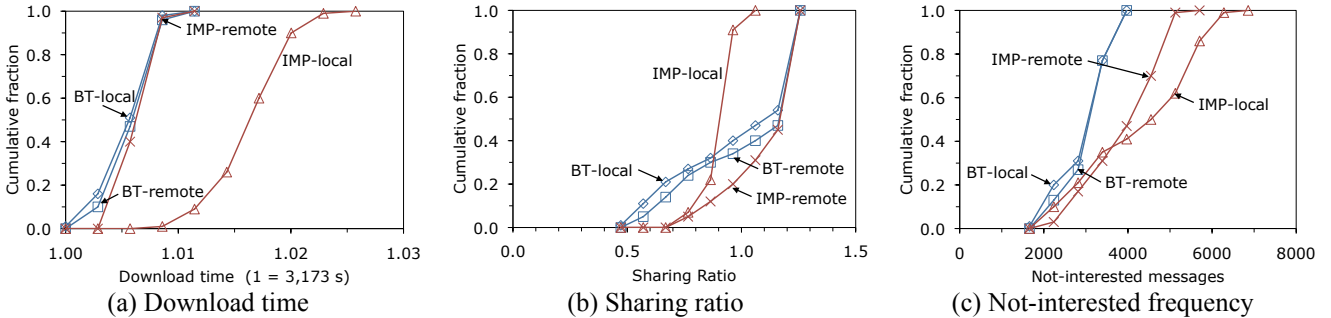


Figure 6. Effect of the oracle on local and remote client performance.

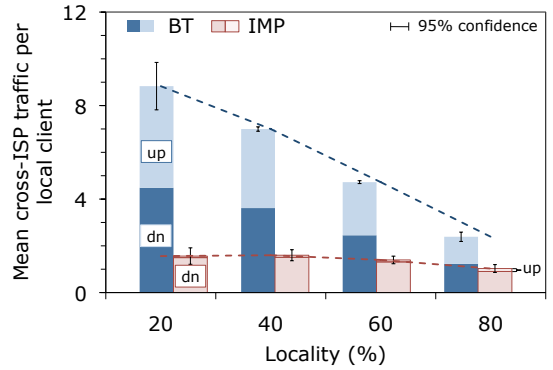


Figure 7. Cross-ISP traffic per local client.

downstream cross-ISP traffic) means that ISP Oracle’s localization-objective increase internal-network traffic [31], but for the remote ISP – the one without IMP installed!

### D. Tracker and Locality

We use a full factorial experimental design with five replications to analyze the impact of the tracker and locality on performance. We use four locality levels: 20%, 40%, 60%, and 80%. Our design fixes the other factors: 10MB, 400 leechers, remote seed, and star topology. (We now use a 10MB file to speed up our experiments and reduce shared, resource utilization.) We ran 40 experiments, which each used 27 hosts and 8 NP routers.

IMP’s savings in cross-ISP traffic does not depend on the number of available local clients. Fig. 7 shows that local client for IMP and BT generate less cross-ISP traffic on average as the fraction of local clients increases. However, the overlapping confidence intervals (error bars) indicate that IMP’s cross-ISP traffic at 20% is not significantly different than at higher locality levels. Moreover, Fig. 8 shows that IMP slightly increases download times. However, this time, it appears that the increase in IMP’s local clients’ sharing ratio alone does not explain the observed differences.

To separate the effects of the tracker and locality on download time and cross-ISP traffic, we use Analysis of Variance (ANOVA) of a multiplicative model. Table IV shows that our model explains 98% of the variation in cross-ISP traffic, of which all components are important. On the other hand, Table 4 shows that 88% of the observed variation in download times is due to effects other than the tracker, locality and its interaction.

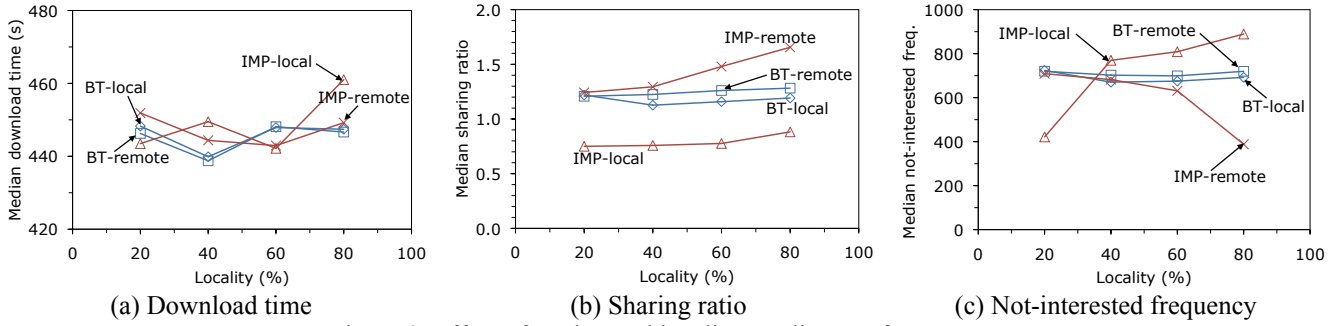


Figure 8. Effect of tracker and locality on client performance.

TABLE IV. ANALYSIS OF VARIATION TABLE FOR TRACKER AND LOCALITY EXPERIMENTS. (SS = SUM OF SQUARES; DF = DEGRESS OF FREEDOM)

Component	Cross-ISP traffic					Client download time				
	SS	% var	Df	F-comp.	F-value	SS	% var	Df	F-comp.	F-value
Oracle	3.30	79.38	1	1,583	4.15	0.00	2.00	1	0.73 <sup>a</sup>	4.15
Locality	0.59	14.06	3	93.47	2.90	0.00	6.02	3	0.73 <sup>a</sup>	2.90
Interaction	0.21	4.95	3	33.91	2.90	0.00	3.78	3	0.46 <sup>a</sup>	2.90
Error	0.07	1.60	32			0.01	88.2	32		

a. Not significant.

### E. Tracker and Topology

We analyze the effect of the tracker and topology on performance using a similar experimental design and multiplicative model. This experiment, however, kept locality at 50% and used four network topologies: mesh, ring, star, and tree where the edge router has one, two, three, and four transit links. Again, we repeated each of the 8 experiments five times.

Fig. 9 suggests that IMP reduces cross-ISP traffic on every network topology. It appears that the choice of tracker determines transit traffic; for a tracker, there is little difference in cross-ISP traffic as topology changes. On the other hand, Fig. 10 suggests that the topology has a larger impact on download times than the tracker; leechers on the tree topology finish faster ( $1 = 234$  s).

Table V shows that IMP reduces cross-ISP traffic and maintains download times regardless of the type of network topology. The effect of the network topology alone and the interaction between tracker and topology are not significant. In particular, an IMP swarm generates 4 times less traffic than BT

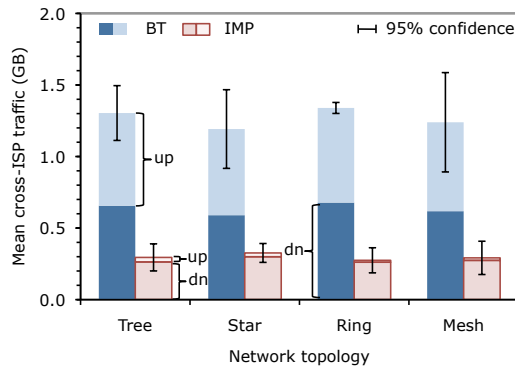


Figure 9. Effect of tracker and topology on cross-ISP traffic.

on an average network topology. For median download times, experimental errors account for 66% of variation. Thus, effects other than the tracker and topology explain most of the observed variation. Furthermore, only the topology is important to download times and only the star topology, which has four transit links, difference is significant.

### F. Multiple Clients Types

We aim to show that IMP works out of the box. To that end, we evaluate our prototype with three BitTorrent clients: mainline, CTorrent [10], and Vuze [29]. Popularity [20], Linux compatibility, and availability of source code dictated our choice of clients. Other than adding code to log download times to CTorrent, the clients were not changed. Each of our three replications of our simple experiment uses a 1GB file, 24 leechers, 50% locality, a local seed, and a remote seed.

Our results show that IMP works without developer support. Fig. 11 shows that IMP reduces cross-ISP traffic and Table VI suggest that it maintains client download times. Since Vuze uses a distributed hash table to “gossip” peers addresses, we cannot say with 95% confidence that the difference in cross-ISP traffic is significantly different from zero for Vuze alone.

TABLE VI. CLIENT DOWNLOAD TIMES WITH BT AND IMP.

Client	50 <sup>th</sup> percentile		95 <sup>th</sup> percentile	
	BT	IMP	BT	IMP
CTorrent	154	157	162	161
Mainline	186	182	192	188
Vuze	159	166	165	173
Mixed	156	160	190	174

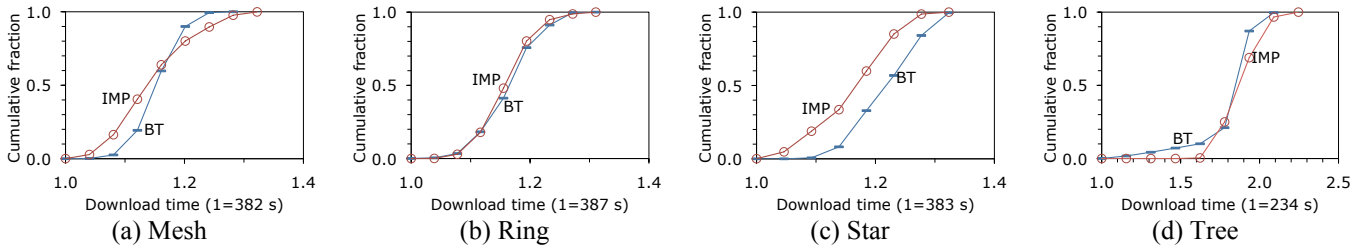


Figure 10. Effect of tracker and topology on client performance.

TABLE V. ANALYSIS OF VARIATION TABLE FOR TRACKER AND TOPOLOGY EXPERIMENTS. (SS = SUM OF SQUARES; DF = DEGREE OF FREEDOM)

Component	Cross-ISP traffic					Client download time				
	SS	% var	Df	F-comp.	F-value	SS	% var	Df	F-comp.	F-value
Oracle	4.08	91.84	1	391.59	4.15	0.00	1.27	1	0.62 <sup>a</sup>	4.15
Topology	0.00	0.08	3	0.12 <sup>a</sup>	2.90	0.00	24.1	3	3.89	2.90
Interaction	0.03	0.57	3	0.81 <sup>a</sup>	2.90	0.00	8.65	3	1.40 <sup>a</sup>	2.90
Error	0.33	7.51	32			0.01	66.0	32		

a. Not significant.

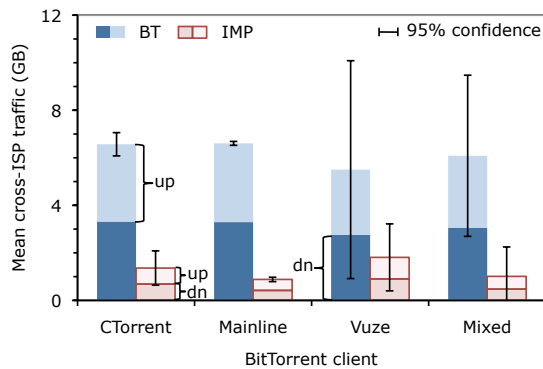


Figure 11. Effect of tracker and client on cross-ISP traffic.

## VI. RELATED WORK

Researchers have approached the problem of reducing P2P costs from three perspectives. First, ISPs install network devices such as content caches and traffic-shaping devices [14, 22]. Similar to IMP, these devices do not change the P2P protocol. However, traffic-limiting increases BitTorrent download times [3]. Although caches do not increase download times, they make ISPs distribute content that maybe copyrighted.

Second, antithetical proposals change the P2P protocol. Bindal et al. [3] showed via simulation that favoring ISP local peers reduces cross-ISP traffic without increasing BitTorrent download times. Choffnes and Bustamante reused network information from content distribution servers to drive biased peer selection [7]. Distributed hash tables also use network measurements to construct proximity-aware overlays networks [6, 19]. However, protocol extensions alone cannot learn about ISP traffic policies and peering agreements [14, 31].

Last, for that reason, some authors suggest that ISPs and developers work together. ISPs provide an *oracle* service, and developers change their application to communicate with the

oracle. Aggarwal et al. [1] proposed that applications send a list of peer addresses to the oracle for sorting by proximity. In addition to this “distance interface”, Xie et al.’s oracle [31] provided data on ISP usage policy and available resources (e.g. caches) via policy and capability interfaces. IMP’s ISP Oracle can return addresses based on usage policy without requiring application changes.

For completeness, we acknowledge that other authors have alluded to devices like IMP. Karagiannis et al. [14] postulated that a “proxy-tracker”, on the edge of the network, could block the request of local peers and redirect them to active local ones. Bindal et al. [3] suggested that ISPs deploy a hybrid traffic-shaping device that rewrites tracker responses to include local peers. Both simulation studies did not describe any mechanisms for implementing such devices.

## VII. CLOSING REMARKS

In summary, our work demonstrates that ISPs can act unilaterally, transparently, and constructively to reduce P2P network costs. Our prototype works without the support of developers who have no incentive to collaborate. Using message flags, IMP informs developers of its action and allows them to override it. Our measurements on real systems show an 8x reduction in costly, cross-ISP traffic without a significant impact on client performance. IMP offers ISPs a practical first step towards remedying their relationship with P2P developers.

A natural question is: how does IMP protect ISPs from copyright infringement? ISPs that use IMP do not contribute to piracy because they do not have any knowledge of the file’s content [28]. Furthermore, IMP offers a location service, which is a “safe harbor” [27]. Since IMP facilitates sharing, a takedown notice from content owners will not stop the infringement. Nonetheless, ISPs can avoid liability by complying with a notice [27].

Our approach does not apply to trackerless torrents, which use distributed hash tables. Since PEER monitors requests to a central tracker, IMP would need a distributed detection

mechanism to find peers. However, there are still more than 1,000 trackers in the wild [26]. And, in the foreseeable future, trackerless torrents will likely supplement trackers [24]. Nevertheless, IMP's overall approach of unilateral, transparent, and constructive action is widely applicable.

Avenues for further study include experiments in the wild, other P2P protocols, and smarter peer-selection. We recently installed monitoring hardware on a university network and may also install IMP after an initial measurement of P2P traffic. For other P2P protocols, we aim to support Gnutella, another popular file-sharing protocol. To drive peer-selection, we intend to consider other parameters such as policy and link congestion [31]. Also, we intend to explore the possibility of sharing peer information between multiple IMP devices.

#### ACKNOWLEDGMENT

We would like to thank the ONL team for timely support and Ken Wong for his advice.

#### REFERENCES

- [1] V. Aggarwal, A. Feldmann, and C. Scheideler, "Can ISPs and P2P users cooperate for improved performance?" in *SIGCOMM* 2007.
- [2] BitTorrent enhancement proposal 8, Tracker peer obfuscation, version 11049, April 2008. [http://www.bittorrent.org/beps/bep\\_0008.html](http://www.bittorrent.org/beps/bep_0008.html)
- [3] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in BitTorrent via biased neighbor selection," in *ICDCS* 2006.
- [4] BitTorrent specification wiki, Bittorrent protocol specification v1.0. <http://wiki.theory.org/BitTorrentSpecification>
- [5] A. Broder and A. Mitzenmacher, "Network applications of bloom filters: a survey," in *Internet Mathematics* 2004.
- [6] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, "Exploiting network proximity in distributed hash tables," in *FuDiCo* 2002.
- [7] D. Choffnes and F. Bustamante, "Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems," in *SIGCOMM* 2008.
- [8] Clip2, "The Gnutella Protocol Specification v0.41", Document Revision 1.2, 2001. [www9.limewire.com/developer/gnutella\\_protocol\\_0.4.pdf](http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf)
- [9] B. Cohen, The BitTorrent protocol specification, version 11031, 2008. [http://www.bittorrent.org/beps/bep\\_0003.html](http://www.bittorrent.org/beps/bep_0003.html)
- [10] CTorrent 1.4 DEVEL. <http://ctorrent.sourceforge.net/>
- [11] Ipoque, "Internet study 2008-2009," 2009.
- [12] R. Jain, "The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling," Wiley-Interscience, New York, NY, April 1991.
- [13] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos, "Is P2P dying or just hiding?," in *GLOBECOM*, 2004
- [14] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. "Should internet service providers fear peer-assisted content distribution?" in *IMC* 2005.
- [15] A. Legout, N. Liogkas, E. Kohler, and L. Zhang, "Clustering and sharing incentives in bittorrent systems," in *SIGMETRICS* 2007.
- [16] A. Madhukar and C. Williamson, "A longitudinal study of P2P traffic classification," in *MASCOTS* 2006.
- [17] Open Network Lab. <http://www.onl.wust.edu>, 2010.
- [18] M. Piatek, H. V. Madhyastha, J. P. John, A. Krishnamurthy, and T. Anderson, "Pitfalls for ISP-friendly P2P design," in *HotNets* 2009.
- [19] P. R. Pietzuch, J. Ledlie, and M. Mitzenmacher, and M. I. Seltzer, "Network-aware overlays with network coordinates," in *ICDCS* 2006.
- [20] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, "Do incentives build robustness in BitTorrent?," in *NSDI* 2007.
- [21] The Register, "Surge in encrypted torrents blindsides record biz," 2007. [http://www.theregister.co.uk/2007/11/08/bittorrent\\_encryption\\_explosion/](http://www.theregister.co.uk/2007/11/08/bittorrent_encryption_explosion/)
- [22] Sandvine. Sandvine: Intelligent broadband network management. <http://www.sandvine.com>
- [23] J. Turner, P. Crowley, J. DeHart, et al., "Supercharging planetLab – a high performance, multi-application, overlay network platform," in *SIGCOMM* 2007.
- [24] TorrentFreak, "BitTorrent's future? DHT, PEX and magnet links explained," November 2009. <http://torrentfreak.com/bittorrents-future-dht-pe-x-and-magnet-links-explained-091120/>
- [25] TorrentFreak, Comcast can block BitTorrent again, Court rules, April 2010. <http://torrentfreak.com/comcast-can-block-bittorrent-again-court-rules-100406/>
- [26] Torrentking.org, Torrent tracker list. April 2010.
- [27] U.S. copyright office summary. The digital millennium copyright act of 1998, pp. 12-13, December 1998.
- [28] F. von Lohmann, "Peer-to-peer file sharing and copyright law: a primer for developers," in *IPTPS* 2003.
- [29] Vuze 4.2.0.4 BitTorrent client. <http://azureus.sourceforge.net/>
- [30] C. Wiseman, J. Turner, et al., "A remotely accessible network processor-based router for network experimentation," in *ANCS* 2008.
- [31] H. Xie, R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz, "P4P: Provider portal for (P2P) applications," in *SIGCOMM* 2008.