Towards Efficient Traffic-analysis Resistant Anonymity Networks

Stevens Le Blond[1] David Choffnes[2] Wenxuan Zhou[3]
Peter Druschel[1] Hitesh Ballani[4] Paul Francis[1]

[1]MPI-SWS [2]Univ. of Washington/ Northeastern Univ. [3]UIUC [4]Microsoft Research

http://aqua.mpi-sws.org

ABSTRACT

Existing IP anonymity systems tend to sacrifice one of low latency, high bandwidth, or resistance to traffic-analysis. High-latency mix-nets like Mixminion batch messages to resist traffic-analysis at the expense of low latency. Onion routing schemes like Tor deliver low latency and high bandwidth, but are not designed to withstand traffic analysis. Designs based on DC-nets or broadcast channels resist traffic analysis and provide low latency, but are limited to low bandwidth communication.

In this paper, we present the design, implementation, and evaluation of Aqua, a high-bandwidth anonymity system that resists traffic analysis. We focus on providing strong anonymity for BitTorrent, and evaluate the performance of Aqua using traces from hundreds of thousands of actual BitTorrent users. We show that Aqua achieves latency low enough for efficient bulk TCP flows, bandwidth sufficient to carry BitTorrent traffic with reasonable efficiency, and resistance to traffic analysis within anonymity sets of hundreds of clients. We conclude that Aqua represents an interesting new point in the space of anonymity network designs.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-communication networks—*Network Architecture and Design*

Keywords

Anonymity networks, P2P file sharing, Strong anonymity

1. INTRODUCTION

Internet users concerned about their privacy, including whistleblowers and dissident citizens of totalitarian states, depend on reliable means to access Internet services anonymously. As demonstrated by a recent subpoena requiring Twitter to provide connection details of suspected Wikileaks supporters [25], governments can readily discover the network identities of web users. Simple proxy VPN services also can be legally compelled to log and reveal client IP addresses, as demonstrated by a UK-based VPN that recently complied with a US subpoena to trace one of its users [1].

Network anonymization services like Tor provide a higher degree of protection, because individual proxies cannot learn both the destination and client IP address of an anonymized flow [11]. However, Tor is not designed to withstand traffic analysis [9, 20, 23, 29, 32, 33], which means an attacker who can observe the traffic at multiple proxies involved in a Tor circuit (e.g., the ingress and egress proxy) can determine the source and destination of the circuit. In practice, governments can request that ISPs duplicate targeted customers' traffic on-the-fly and forward it through a secure channel [2]. The existence of surveillance facilities like the NSA Spy Center in Utah suggests that government agencies may already be collecting such information at a massive scale [3].

Traffic analysis works by matching the time series of encrypted packets within a circuit at different proxies. To defeat traffic analysis, proxies have to obscure the temporal pattern of individual packet flows (we refer to this process as *traffic obfuscation*). Obfuscation can be achieved by batching packets from different flows or by adding artificial delay or artificial traffic called *chaff*. Obfuscation necessarily exacts a cost in terms of the delay, throughput, or bandwidth requirements of anonymized flows.

Roughly a decade ago, a number of researchers proposed designs for *low-latency, traffic-analysis resistant* anonymity networks [4, 8, 12, 17, 18, 26, 28]. The performance of these systems, however, was rather discouraging. Perhaps as a result there has been a dearth of research in recent years on low-latency traffic-analysis resistant anonymity networks. While there is likely no silver bullet design, we feel that this problem is important enough that researchers should continue to work for solutions that exhibit an acceptable benefit-cost ratio under some set of realistic conditions.

Towards this end, this paper describes the design, implementation, and evaluation of *Anonymous Quanta* (*Aqua* for short), a low-latency anonymity network that resists traffic analysis, can tolerate a bounded number of compromised nodes, and scales well with the number of users. *Aqua* demonstrates substantial performance gains over previous designs for a workload based on thousands of actual BitTorrent users. For instance, we show that *Aqua* achieves k-anonymity within a set of $k = 100$ BitTorrent users with a median cost of 15% additional bandwidth utilization and 20% longer download time.

There are three key insights behind *Aqua*'s design. First, *Aqua* uses a different anonymization and traffic obfuscation

SIGCOMM'13, August 12–16, 2013, Hong Kong, China.

strategy in the core (i.e., on links connecting *Aqua* proxies) than it does at the edges (i.e., on links connecting clients to an *Aqua* proxy). This separation allows *Aqua* to take advantage of the different traffic conditions in core and edge to minimize overhead for a given degree of anonymity. Second, in the core, *Aqua* relies on encrypted, chaffed flows to achieve a payload-independent, uniform traffic rate. It routes client payload flows via multiple overlay paths to take advantage of available capacity and minimize chaff bandwidth overhead. Third, at the edges, *Aqua* forms sets of clients with similar payload traffic patterns, and varies the rate of encrypted, chaffed traffic on client links in the same set uniformly to achieve k-anonymity. Thus, *Aqua* can take advantage of existing spatial and temporal correlation among client payload flows to achieve large anonymity sets at low overhead.

Aqua's initial target application is peer-to-peer file sharing. BitTorrent is an important application, because it has a large user base and many of its users desire anonymity. A recent study showed that 17% of P2P file-sharing users were already employing anonymizing services and 51% wished to do so in the future [19]. For lack of an alternative, users today resort to Tor, which is not designed to withstand traffic analysis. Moreover, BitTorrent carries much traffic, which presents both a challenge for existing anonymity designs and an opportunity for Aqua: The higher the temporal and spatial correlation among payload flows, the lower the inherent overhead required to obfuscate the source and destination of an individual flow.

This paper makes the following contributions:

- The design of a traffic-analysis resistant anonymity network with high bandwidth efficiency and latency low enough for bulk TCP flows.
- A trace-driven evaluation of *Aqua* on a workload of hundreds of thousands of actual BitTorrent users.
- A comparison of *Aqua*'s performance with several alternative designs: constant rate chaffing, broadcast channels, and P2P overlays.

Our experimental evaluation shows that using multiple paths in the core helps to disperse traffic peaks, which in turn reduces the rate of chaff traffic and the resulting overhead significantly. Also, enforcing uniform variable rate among endpoints with correlated traffic can provide k-anonymity with low overhead.

The rest of this paper is organized as follow. Background and related work is discussed in Section 2. We describe the *Aqua* design and its security properties in Section 3. An experimental evaluation of the *Aqua* design, and some preliminary results with a prototype implementation are presented in Section 4. Finally, we conclude in Section 5.

2. BACKGROUND AND RELATED WORK

Fundamental techniques. The ultimate source or destination of an IP packet can be concealed using a relay node, such as a VPN service. However, this simple approach depends on the integrity of the relay node, which knows the source and destination of any packet passing through it. The approach can be generalized to an *anonymity network*, where packets are forwarded via several relays, such that no single relay knows both source and destination of a packet (or depending on the design, neither source nor destination).

Chaum introduced *mix-nets* to provide anonymous communication and defeat traffic analysis [5]. In addition to relaying messages, a mix hides the correspondence between input and output messages through hop-by-hop encryption and batching. Encryption provides bit-wise unlinkability; that is, it unlinks the bit patterns of messages arriving at the mix and the messages departing from the mix. Batching prevents an attacker from tracing messages based on their arrival and departure times. To defend against compromised mixes, each message can be sent through a sequence of mixes so anonymity is maintained as long as at least one mix in the sequence is honest.

Dining cryptographers (D-C) nets [6, 16] and verifiable shuffles [13, 22, 31] are cryptographic techniques that offer strong resistance to traffic analysis without requiring batching. However, computation and communication costs have generally limited designs based on these ideas to small anonymity sets and low bandwidth efficiency.

At a high level, anonymity networks can be divided into peer-to-peer networks consisting only of clients and infrastructure based networks with dedicated relays separate from the clients. P2P networks [12, 24] are inherently robust to network edge analysis, because an attacker cannot distinguish whether a node is the source, destination or relay of traffic. On the other hand, peers tend to be less reliable, less powerful and more heterogeneous, which makes it harder to provide predictable performance. Lastly, P2P network expose clients to additional legal risks, because they relay traffic for other clients. Infrastructure based networks [11] tend to have more powerful and reliable relay nodes, with known locations and jurisdictions. However, they face network-edge attacks, which require weaker adversarial models (i.e., trusted entry/exit relays) or additional defenses.

High-latency anonymity networks. Designs providing both bit-wise unlinkability and batching are generally referred to as *high-latency anonymity networks*. These designs are implemented by systems like Babel [15], Mixmaster [21], and Mixminion [10] and carry delay-tolerant communications such as e-mails. Mix-nets typically perform public key encryption for each message they process and typically delay messages for hours for the purpose of batching.

Modest-latency anonymity networks. Many applications require both anonymity and modest latency. By refraining from batching, anonymity designs can typically reduce round trip time to seconds or hundreds of milliseconds, making them appropriate for flow-based communication. But, by doing so, they generally have to give up one of bandwidth efficiency (e.g., P^5 [26], Dissent [30]), or resistance to traffic analysis (e.g., Tor).

Onion routing (Tor). The most popular low-latency anonymity design, Tor, is circuit-based. To establish a circuit, a client selects a number of proxies at random and then establishes a session key with each proxy in such a way that each proxy knows only its predecessor and successor in the circuit. Once a circuit is established, the client can encrypt a packet using the session key of the proxies from last to first and send the encrypted packet to the first proxy of the circuit. In turn, each proxy decrypts the packet and forwards it to its successor in the circuit until the unencrypted packet leaves the last proxy of the circuit. Tor is not designed to resist traffic analysis attacks.

Dissent [30] is an infrastructure based anonymity service with a very strong adversarial model, where a single honest

	Aqua	Tor	Tarzan	Dissent	P^5	Mixminion
Architecture	c/s	c/s	P2P	c/s	P2P	c/s
Traffic analysis resistance	high	none	high	very high	high	very high
Latency	low	low	low	low	low	high
Bandwidth efficiency	high	high	medium	low	low	high
Anonymity set size	medium	large	medium	small	small	large

Table 1: Comparison of anonymity networks

proxy is sufficient to ensure anonymity, even in the presence of an attacker who can observe all traffic. The system relies on DC-nets and verifiable shuffles, and cleverly exploits its infrastructure based architecture to scale to hundreds of clients with modest delay and bandwidth sufficient for web browsing. However, the system's capacity and scalability are subject to DC-net scaling limits with respect to the number of proxies, and overhead per payload bit for cryptographic processing.

P^5 [26] is a scalable peer-to-peer anonymity network robust to passive traffic analysis. The participating peers form a hierarchy of *broadcast channels*, such that each peer joins a small number (2 or 3) of different channels. The peers in a given channel exchange hop-by-hop encrypted packets at a fixed rate, mixing payload traffic with chaff to achieve a fixed target rate. To send a packet to a peer r, the sender encrypts the packet with r's public key, and forwards the message to one of the receiver's channels, in which it is broadcast. By choosing channels with different numbers of members, peers can trade receiver anonymity for communication efficiency.

Tarzan [12] is a low-latency peer-to-peer anonymity network. Tarzan relies on layered encryption to achieve bit-wise unlinkability despite malicious peers. Tarzan uses chaff traffic to obscure traffic patterns and ensures that the traffic of the nodes within a given anonymity set is indistinguishable.

The original design of the Java Anonymous Proxy (JAP) [4] infrastructure based anonymity network had clients exchange constant traffic with the first mix of a cascade, in order to defeat end-to-end traffic analysis. However, this countermeasure was abandoned in the JAP deployment because its bandwidth overhead was considered too high. Similarly, the Freedom Network was initially deployed with some countermeasures against traffic analysis but they were latter removed due to their high bandwidth overhead [27].

Summary. Compared to existing techniques, *Aqua* occupies a different point in the design space, seeking to meet the needs of applications with a need for high bandwidth efficiency, like BitTorrent. It combines a strong adversarial model and modest latency with high bandwidth efficiency and significantly large anonymity sets (hundreds). *Aqua* differs from Tor in its resistance to traffic analysis. *Aqua* differs from P^5 in its design, bandwidth efficiency, and detailed threat model. Relative to Dissent, *Aqua* has a weaker adversarial model (i.e., entry and exit mixes are assumed to be trustworthy), but avoids the scaling limits due to the computational overhead of D-C nets. Unlike P^5 and Dissent, *Aqua* uses unicast routing instead of broadcast, which gives it inherently better throughput for a given level of anonymity and overhead. The technique used by *Aqua* to obscure traffic on its client links has some similarity with Tarzan's mimics. However, *Aqua* splits payload traffic along multiple paths, enabling it to better utilize available bandwidth within existing anonymity sets. Finally, *Aqua* takes advantage of similarity in concurrent payload traffic to achieve anonymity with high bandwidth efficiency. Together, these techniques enable *Aqua* to achieve better bandwidth efficiency than Tarzan and, to the best of our knowledge, all existing traffic analysis resistant anonymity networks. Table 1 summarizes our comparison.

3. DESIGN

3.1 Overview

System Model. *Aqua* consists of mixes, which relay traffic, and clients, which originate and terminate traffic. Clients and mixes are connected by links, which carry encrypted traffic. Each client is attached to exactly one mix at a time, chosen by the client to meet her privacy needs. For a given payload flow, the mixes adjacent to the two communicating clients are called the *edge mixes*.

Aqua adopts an infrastructure based architecture, because dedicated mixes are likely to be more reliable and can be placed in well-known locations and jurisdictions. Moreover, this architecture does not expose clients to legal risks associated with forwarding the traffic of unknown participants like P2P architectures do.

In simple cases, an originating client contacts a receiving client (e.g., a server) at a well known address through *Aqua*. Here, *Aqua* hides the address of the client, a property called *sender anonymity*. In other cases, a receiving client wishes to receive anonymous connections without revealing her own address, a property called *receiver anonymity*. To this end, *Aqua* provides a rendezvous mechanism, described in Section 3.4, which joins two sender-anonymous flows at a *rendezvous mix*, forming a sender-receiver mutually anonymous flow.

Threat Model. We assume an attacker who seeks to infer which pairs of clients communicate via *Aqua*. The attacker is able to observe the time series of encrypted traffic at all clients and mixes as part of a *global, passive traffic analysis attack*. Within a portion of the Internet controlled by the attacker, he can additionally compromise mixes and clients, and modify the time series of encrypted traffic as part of a *local, active traffic analysis attack*. However, it is assumed that legitimate clients choose uncompromised edge mixes for their circuits[1], and that the attacker controls any of the client, the edge mix, or the network path between client and edge mix on at most one end of a *Aqua* circuit.

The attacker can control only a bounded number of clients and a bounded number of mixes. The ratio of active clients to mixes is assumed to be large enough to ensure that there are many payload flows between each pair of mixes at any time. Lastly, we make the common assumption that attackers cannot break the cryptographic primitives or compromise the keys used by mixes or clients they do not control. We will discuss *Aqua*'s anonymity under these attacks in Section 3.5.

Goals. *Aqua* ensures the following anonymity property under the threat model described above:

[1]For instance, by considering the mixes' location, jurisdiction, history and operator relative to the type of communication the client seeks to perform.

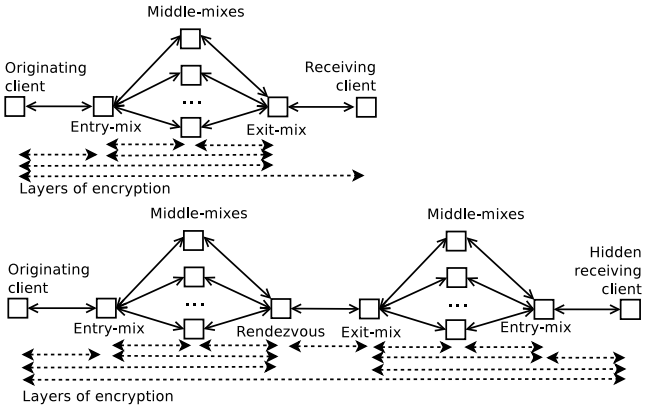

Figure 1: Top: A single circuit terminated by a receiving client provides sender-anonymity. Bottom: Two *Aqua* circuits joined at a rendezvous mix provide a sender-receiver mutually anonymous circuit.

- k**-anonymity.** The attacker cannot determine which legitimate client among a set of k clients is communicating with a given target client or rendezvous mix.

Moreover, *Aqua* has the following performance goals:

- **Scalability to large k.** *Aqua* scales to large k, i.e., to large anonymity sets.
- **Modest end-to-end latency.** Because *Aqua* targets applications like BitTorrent, its latencies must be low enough to achieve good bulk TCP performance.
- **High bandwidth.** As *Aqua* targets bandwidth-intensive applications, it must be able to use the capacity of the underlying network effectively.

Roadmap. In the rest of this section, we discuss the components of *Aqua*'s design. *Bitwise unlinkability* is ensured through layered, hop-by-hop encryption. Payload flows are routed via a circuit consisting of an entry mix, a set of middle mixes, and an exit mix. *Traffic obfuscation* is achieved by different mechanisms in the core and at the edge of the *Aqua* network. In the core, *Aqua* maintains constant rate encrypted traffic among mixes, where payload traffic is augmented with chaff traffic to maintain the rate. Multipath routing via different middle mixes exploits available payload bandwidth in the core while keeping chaff traffic to a minimum.

To obfuscate traffic at the edge, *Aqua* dynamically chooses sets of clients with correlated payload traffic patterns. The encrypted traffic rate of client links within a set is then coupled, by shaping payload traffic and augmenting it with chaff. As a result, the observed time series of encrypted traffic on links in a set reveals nothing about payload flows, ensuring k-anonymity. By assigning clients to an anonymity set with similar payload traffic patterns, *Aqua* can exploit correlated client traffic to achieve large anonymity sets while keeping chaff traffic and payload traffic shaping low. Finally, *rendezvous* provides receiver anonymity, by concatenating two sender-anonymous circuits via a rendezvous mix (see Fig. 1).

3.2 Bit-wise Unlinkability

Bit-wise unlinkability is achieved through layered and hop-by-hop encryption over a dedicated circuit determined by the originating client. When a client wishes to initiate a bi-directional flow with another client, it selects two mixes to serve as *edge mixes* for the duration of the flow. The client establishes an onion circuit [14] involving the two edge mixes and the destination. The destination could be a receiving client, or a rendezvous mix in case of a receiver anonymous circuit (see Section 3.4).

Packets exchanged between the edge mixes may be sent on the direct link between the edge mixes, or traverse a *middle mix* selected from the other mixes. Different packets from a given flow may traverse different middle mixes. By forwarding traffic via middle mixes, *Aqua* can accommodate payload traffic in excess of the constant rate of the link between a pair of mixes. Every edge mix pair maintains an encrypted session so that middle mixes cannot identify the circuit id of packets they forward.

Security. Because the traffic on each link is encrypted using a secret key shared only between adjacent nodes, the bit-wise content appears random and uncorrelated with the traffic on other links. Thus, the attacker can learn nothing by observing the content of network traffic. If a middle mix is compromised, it can learn the edge mixes of encrypted packets it forwards, but not the circuit id or clients of the packet's flow.

3.3 Traffic-analysis Resistance

Despite bitwise unlinkability, an attacker can observe and correlate the time series of encrypted packets on different links. Changes in the payload rate of a flow, dynamic capacity changes of network links (e.g. due to congestion), or manipulation of encrypted traffic by an active attacker can cause correlated changes in the time series of encrypted traffic along a flow's path. To defeat such *traffic analysis*, the anonymity network must craft the time series of packets on each link such that the attacker is unable to infer which clients are communicating. Such traffic obfuscation can be accomplished by batching payload traffic from different flows, splitting payload traffic across multiple paths, adding artificial delay to payload traffic, or adding artificial chaff traffic.

Aqua uses a combination of chaffing and delayed flow start-up for traffic obfuscation. Moreover, *Aqua* uses a different strategy for traffic obfuscation in the core and at the edges of the network. In the Aqua core, we use *uniform rate chaffing*, because there is enough statistical multiplexing of inter-mix traffic that aggregate rate changes are infrequent. The multi-path routing at the core further helps to smoothen traffic imbalances among the mixes. At the network edge, we devise a novel *dynamic chaffing* strategy for traffic obfuscation, which can take advantage of temporal and spatial correlation among different clients' payload traffic to achieve k-anonymity in large sets at low overhead, despite the bursty nature of individual flows. We describe both methods in more detail below.

3.3.1 Traffic obfuscation in the core

Aqua's traffic obfuscation in the core is conceptually simple: all mixes transmit to all other mixes at a constant rate. Mixes partition time into small periods called *batch periods*. During each period, each mix sends the same number of

same-sized packets to every other mix. This includes packets transmitted by a mix in its roles as both edge mix and middle mix. The packets transmitted by a mix during period t include the payload data received during period $t-1$.

Specifically, each mix transmits n packets to each of m mixes during a batch period. At the beginning of a period, each mix assigns the payload data received in its role as a middle mix, and assign it to the designated edge mixes. If payload data worth more than n packets are assigned to a given edge mix, the excess packets are dropped. Then, the mix randomly assigns payload data it received in its role as an edge mix to middle mixes with available payload capacity in its packet slots. If there is more payload data than available slots, excess packets are dropped. If there are more slots than payload data (the common case), then chaff data is added as needed to fill the n packets. To achieve a constant rate, each mix schedules a packet for transmission a fixed interval after the previous packet was acknowledged (retransmitting dropped packets as needed).

To efficiently accommodate temporal variations in aggregate payload rates in the core, our design allows the mix-to-mix link rates to vary over time, but the target rates on all links are identical at any instant.

Security. Consider the security properties of mixes exchanging bidirectional encrypted traffic at a fixed target rate. Due to flow and congestion control, the achieved rate on a link may differ from the target. The important thing to note, however, is that the actual rate on a link reflects only the capacity and congestion state of the underlying network, and reveals nothing about the payload it carries. Thus, the constant rate traffic is perfectly resistant to passive traffic analysis. Moreover, actively delaying or dropping traffic on a link does not reveal any useful information, because the downstream node's outgoing stream rate will not be affected (it simply adds more chaff). Changes in the target rate on all inter-mix links over time reveal some information about the aggregate payload traffic rate in the core. Given the assumed high degree of flow multiplexing in the core, however, it reveals nothing about individual flows or the communicating partners.

3.3.2 Traffic obfuscation at the edges

To resist traffic analysis, *Aqua* must also obscure traffic on client links. The target rate of encrypted traffic on a client link can change dynamically, to accomodate variations in a client's payload traffic efficiently. However, any target rate adjustment on a client link must coincide with an equivalent adjustment by a set of clients that form a anonymity set called a *kset*. The target rate of a client link is the sum of the target rates of all ksets the client is currently a member of. This technique allows *Aqua* to accommodate variations in clients' payload traffic rates while ensuring that any given client's observable rate change is indistinguishable from that of k other clients. By seeking to form ksets out of clients with correlated payload traffic, *Aqua* can reduce overhead for a given anonymity set size.

Design. A client announces to its edge mix when it has an incoming or outgoing flow it would like to start, along with its available bandwidth capacity. The mix in turn announces this information to other mixes, without revealing the client's id.

Mixes wait until they have collectively received about $k+\Delta$ (Δ defined below) announcements each for incoming and outgoing flows, or a timeout occurs. When possible, mixes match announcements for flows with a similar expected rate. When a timeout occurs with less than k matching announcements, additional clients with available link capacity are randomly selected to join a kset, even if they have no flows to start. At this point, the mixes ask the clients in the kset and their edge mixes to simultaneously increase their current client link rates by a specified kset rate. The kset rate is equal to the minimum available capacity of all selected clients, or the maximal requested flow rate, whichever is lower. We describe a simple, greedy algorithm for kset formation in more detail in Section 4.3.

Once the clients in a kset have jointly increased their rates, they must eventually decrease their rates by the kset rate simultaneously, to avoid leaking any information about individual flows. Clients whose flows end early must continue to send chaff at the required rate. They are free to start new flows within the anonymity set to fill the available capacity. When one of the k clients fails or departs abruptly, the remaining members of an anonymity set must immediately reduce their rates by the kset rate. To tolerate such departures more gracefully, sets may be formed with more than k clients initially, as described below.

In practice, the implementation operates in time-synchronized *epochs*. A typical epoch time might be 30 seconds. *Aqua* collects flow announcements and forms ksets during the present epoch, and starts these ksets at the beginning of the next epoch. In *Aqua*, all clients send and receive at a low constant baseline rate. This make sense for applications with many small flows and background signaling traffic, like BitTorrent.

Forming ksets. To achieve k-anonymity, a kset must in practice include $k+\Delta$ members. The extra Δ clients are required to compensate for kset members attached to compromised edge mixes and clients controlled by the attacker. While not required for anonymity, some number of extra kset members can also reduce the need for ungraceful kset shut-downs due to client or mix failure. Consider the case of M mixes, $m*M$ of which are controlled by the attacker, and C clients, $c*C$ of which are controlled by the attacker. Uncompromised clients attach to random mixes, but the compromised clients attach evenly to the uncompromised mixes.

First, to limit the impact of compromised mixes, *Aqua* requires that each mix contribute an equal number of clients ($(k+\Delta)/M$) to each kset. To compensate for clients attached to compromised mixes, we need to add $m(k+\Delta)$ clients to a kset; to compensate for compromised clients attached to the uncompromised mixes, we need to add $(1-m)(k+\Delta)c$ clients to a kset. To tolerate client and mix failures, we add an empirical number r clients. Thus, $\Delta = r + k\frac{mc-m-c}{m+c-mc-1}$. For instance, when $k=100; r=0; c=m=0.1; \Delta=23.45$.

The required number of clients ($(k+\Delta)/M$) contributed to a kset by each mix is generally a fractional value. Therefore, some mixes have to contribute one more client than others in practice. The mixes that contribute an additional client must be chosen deterministically to avoid any bias towards compromised mixes, and fairly to avoid load imbalance. *Aqua* uses consistent hashing among the mixes to make a deterministic choice, parameterized by a global kset sequence number for load balance.

Security. Let us consider the subgraph of client links whose rates change in response to a flow start-up/shut-down. The set of clients who might be communicating is simply the

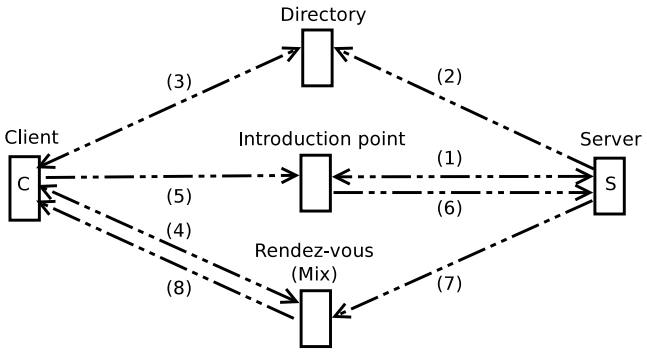

Figure 2: Rendezvous design. Each line is a full three-hop circuit (Entry/Middle/Exit Mix). For the flow itself (not shown), the rendezvous Mix acts as the exit mix of the client's circuit and is concatenated to the server's circuit.

number of clients connected to the subgraph. *Aqua* enforces the following conditions regarding the observable link rates:

1. *A rate increase by r bits/s in response to a flow start-up must simultaneously affect a subgraph s that connects at least $k + \Delta$ clients.*

2. *A rate decrease in response to a flow shut-down, mix failure or client departure that reduces s to less than k members must be accompanied by a simultaneous rate reduction by r in the entire s.*

These conditions ensure that the attacker cannot infer which client within a set of k clients is communicating. The reason is that any observable rate change is consistent with the start-up or shut-down of a flow by any client within the set of k clients. Each kset is effectively an instance of a uniform rate chaffed network, like the one used in *Aqua*'s core.

3.4 Rendezvous

For clients who wish to be reached anonymously (receiver anonymity), as is the case in BitTorrent, *Aqua* provides a rendezvous mechanism similar to Tor's hidden service mechanism. This design comprises four components: a hidden receiver, a directory server, the rendezvous point, and the introduction point. The hidden receiver is a client who offers a (hidden) service (e.g., BitTorrent) to other clients of the anonymity network. The directory server keeps track of where to contact hidden receivers. The rendezvous point relays the payload data traffic between a client and the hidden receiver. The introduction point is where the hidden receiver listens for connections.

We illustrate this design for rendezvous in Fig. 2. Note that all lines in Fig. 2 represent full three-hop *Aqua* circuits, thus providing anonymity for both client and hidden receiver with respect to the other components. A hidden receiver starts by requesting that an introduction point listens for incoming connections from clients (1). The receiver then publishes the contact information of its introduction point in the directory service (2). Clients find out about the receiver out of band and look up its introduction point in the directory (3). For robustness, a receiver can have several introduction points. The client then requests that a rendezvous listens for incoming connections from the receiver on its behalf (4) and notifies the receiver through the introduction point (5 & 6). Finally, the receiver connects to the client (7 & 8) after which, the two can communicate through the rendezvous. Note that the final communications path has six mixes. The rendezvous node itself acts as the edge mix at the far end of both the client's and the hidden receiver's circuit.

Security. Rendezvous provides both sender and receiver anonymity, because it concatenates two sender-anonymous circuits. Because each side chooses its own set of edge mixes, neither client's anonymity depends on choices made by the other client.

3.5 Attacks

Next, we discuss attacks within our threat model and how *Aqua* defends against them.

Passive traffic analysis attack. The traffic rate on links in the core does not depend on individual payload flows. Traffic analysis on core links is therefore unproductive. At the edges, all rate changes coincide on at least k client links. Therefore, the attacker cannot tell which of the k clients are communicating.

Active traffic analysis attack. Here, the attacker manipulates the flow of encrypted, chaffed *Aqua* traffic by delaying, dropping or replaying packets. Doing so has no impact on the rate of downstream chaffed traffic, so it does not help the attacker trace payload flows. For that, the attacker needs access to a client's payload flow at the other end of a *Aqua* circuit. However, this case is ruled out by the threat model.

Compromised middle mix. The attacker cannot decrypt the contents or circuit ids of packets its forwards, but learns the edge mixes between which a packet travels. To find out which client is communicating, the attacker would have to (i) estimate the aggregate payload traffic between a pair of edge nodes based on the isolated sample packets it sees, (ii) analyze traffic at the two edge mixes to determine the set of attached clients who share ksets, and (iii) infer which of these clients are communicating by correlating rate changes in a kset with changes of the observed aggregate payload rate. With a single flow between a pair of mixes, this could conceivably allow an attacker to narrow the candidate client set to $(k + \Delta)/M$; per our assumption, however, a large number or flows (and thus ksets) exist between any pair of mixes at any time, making this attack infeasible in practice.

Compromised rendezvous mix or hidden receiver. A client's anonymity is not affected by a compromised rendezvous mix or hidden receiver, because it depends only on the edge mixes chosen by the client.

Compromised clients (Sybil attack). Clients controlled by the attacker effectively reduce the anonymity provided by each kset they participate in. *Aqua* compensates by increasing kset sizes based on the given bound on the proportion of compromised clients, to ensure a minimum of k uncompromised clients in the set. Thus, the attack is not effective as long as the proportion remains within the bound.

Long-term intersection attacks. In a long-term intersection attack, the attacker takes advantage of repeated communication between a pair of clients to observe which clients are *(almost) always* part of a kset. Over a long time, the intersection of these successive ksets will shrink towards the communicating clients. The attack requires that a pair of clients communicate repeatedly and in a way that the at-

tacker can predict (e.g., by infering that clients are likely to communicate whenever they are online). In general, users of anonymizing networks must take care to avoid predictable communication patterns (e.g., not go online only when communicating) to avoid this attack.

4. EVALUATION

In this section, we use trace-driven simulations to analyze the bandwidth overhead of traffic chaffing and the latency of the *Aqua* design. We show that *Aqua* offers strong privacy guarantees at low overhead in terms of throughput and latency.

We evaluate the performance of *Aqua* relative to other representative approaches for anonymous communication. To provide an apples-to-apples comparison of the network overhead of alternative designs, we extract the key mechanisms for anonymity and traffic analysis resistance for each anonymity system and evaluate their impact on performance. At a high level, we find that other anonymous system designs either sacrifice performance or scalability, whereas *Aqua* achieves both on BitTorrent workloads. To summarize our key results, we find that on this workload, *Aqua* has low overhead (10-30%) and low throttling (10-50%), significantly lower than of other anonymity system designs. It accomplishes this with latency that is a small fraction (12%) larger than in onion routing systems.

We discuss the dataset that we use for this analysis in the next section. Then we describe the goals of our evaluation in Sec. 4.2 and detail the systems models that we evaluate in Sec. 4.3. We present the results of our evaluation in Sec. 4.4 and discuss limitations of our simulation-based analysis in Sec. 4.5. We close with some preliminary results from a prototype implementation of *Aqua* in Tor in Sec. 4.6.

4.1 Dataset

We use trace-based simulations to evaluate the efficiency and privacy of Aqua and some of its related work. Specifically, we use the Ono dataset, a large collection of trace data gathered from BitTorrent users [7]. This data is gathered by Ono, a plugin for the Vuze BitTorrent client that attempts to bias peer connections toward relatively nearby hosts to reduce cross-ISP traffic in P2P systems. In addition to providing this service, the plugin collects anonymous information about transfer rates, ping latencies and traceroute paths between BitTorrent peers for connections established by participating users. Ono collects no Personal Identifiable Information (PII) nor information about the files that users download; users can opt out of data collection at any time.

To analyze the overhead in terms of transfer rates, we use flow samples from real BitTorrent transfers. The data contains per-flow transfer rate samples recorded at 30-second intervals for each peer in the trace. On average there are over 1,000 users online at any time. We use the trace of November 2010, which includes approximately 20 million samples per day. We infer the bandwidth capacity of each peer using the maximum transfer rate observed during the month. To analyze the delay overhead, we use latencies measured from traceroutes issued by the BitTorrent hosts. This data includes more than 200 million measurements.

4.2 Goals

We focus on evaluating the *network* overheads of the *Aqua* design and comparing them with alternative designs for anonymous communication. Specifically, we quantify the costs of each system according to the following metrics:

Overhead. We define the overhead as the number of chaff bytes each endpoint sends divided by the number of all bytes (chaff and payload) sent. The overhead captures the amount of additional bandwidth consumed by a design to resist traffic analysis.

Throttling. We define throttling as follows. For each endpoint, we define the number of throttled bytes as the absolute difference between the number of payload bytes sent in the BitTorrent trace and those sent by a given simulated system during the same period. We then divide the number of throttled bytes by the total number of payload bytes sent in the BitTorrent trace. The throttling indicates the slowdown imposed by an anonymity network as compared to using unmodified BitTorrent.

Latency. We define latency overhead as the additional delay due to using multiple hops through an anonymity network compared to using direct Internet paths.

Model (Example)	Technique	Anonymity set
Constant	constant rate C	N
Broadcast (P^5)	broadcast group k	k
P2P (Tarzan)	peer group n	$min(n^{hops}, N)$
Aqua	k-set	k

Table 2: Models evaluated in this section. The total number of participating hosts is N.

4.3 System Models

We evaluate four models of anonymity systems, representative of designs described in Sections 2 and 3. Table 2 gives an overview of the different models. We assume that hosts in the core of the network (mixes) have sufficient capacity to support all flows in the system without throttling. Further, we quantify the anonymity of each system as the number of clients within an anonymity set.

Constant-rate. In the constant-rate model, online endpoints exchange constant traffic with their edge mix at their capacity rate. Online but inactive endpoints exchange only chaff traffic with their edge mix. When endpoints become active, they replace chaff traffic with payload traffic. *The constant-rate model achieves an anonymity equal to the number of online endpoints in the system.*

Broadcast. [*Example:* P^5 and DC-Nets] The broadcast model partitions online endpoints into fixed size broadcast groups. Endpoints within each group share a broadcast channel, so every message to a single endpoint is broadcast to all endpoints in the same group. Group members always send and receive traffic at a constant rate equal to the minimum capacity of all group members, pad payload traffic with chaff traffic as needed, and throttle payload traffic when its speed exceeds this rate. To limit throttling, we assign endpoints to broadcast groups based on their capacity in our simulations. *The broadcast model provides k-anonymity among the endpoints of each broadcast group.*

P2P. [*Example: Tarzan*] The P2P model assumes traffic is routed only between neighbors in the P2P overlay. Each pair of neighbors maintains a bidirectional, constant-rate stream, into which payload traffic can be inserted. The constant rate of each stream should satisfy two constraints: a) the sum of rates of streams from/to an endpoint must be no larger than

its sending/receiving capacity; b) any stream's rate must be no larger than the sender's sending rate and the receiver's receiving rate. *The anonymity of the P2P model increases exponentially with the number of P2P hops.* Specifically, it is $min(n^h, N)$, where n is the number of P2P connections, h is the number of hops and N is the size of the system. We use $h = 3$ and $n = 2, 5, 10$.

Aqua. *Aqua* is the design presented in Section 3, where endpoints form ksets and use mixes to route their traffic. The *Aqua* trace-driven simulations model ksets for various k. The simulator obfuscates traffic by adding chaff or throttling connections based on the nominal rate of the kset (i.e., the minimum available capacity in the kset). A peer is not allowed to exit the system until its kset is torn down; if the associated flow is inactive, the peer must continue to send chaff traffic at the kset rate. In our simulations, *Aqua* assigns peers to ksets using a greedy online algorithm that matches peers with similar capacities. During each round all N peers that need to join ksets are ordered according to their capacity. We assign the first k peers to the same kset, the second k peers to another kset, and so on until all peers are assigned to a kset. The last (N *mod* k) peers must postpone transfers until their kset is complete (*e.g.,* from peers with new flows starting in the next round). *Aqua provides k-anonymity among the members of a kset.*

4.4 Results

Next, we present the simulation results. We find that on the BitTorrent workload, *Aqua* incurs substantially less overhead and throttling than other anonymity system designs (*e.g.,* as little as 1/3 of the overhead and 1/2 of the throttling when compared to an alternative P2P-based anonymity system), and its delay from multiple hops is not sufficiently large to impact performance for P2P file sharing (the delay is not significantly larger than onion routing).

4.4.1 Endpoint Performance

At the endpoints, the network cost of anonymizing systems is overhead from chaff traffic and throttling to maintain traffic-analysis resistance. With the exception of *Aqua*, we find that the evaluated models sacrifice performance, scalability or both. Importantly, ksets efficiently provide anonymity for the large numbers of flows generated by BitTorrent; further, *Aqua*'s infrastructure-based mixes provide the relatively large bandwidth resources required for resisting traffic analysis.

Methodology. To evaluate the performance of each system design, we use the timeseries of BitTorrent flows form the Ono dataset. In particular, when a peer in the trace initiates a flow, we simulate the protocols described in the previous section. Each system imposes overhead and throttling, which we track and compare to the empirical traces as a baseline representing performance without any anonymity.

To evaluate large kset sizes (*i.e.,* 1000) in *Aqua*, we need more concurrent users than exist in our dataset. To generate a larger set that is representative of the original dataset, we stack daily traces by modifying timestamps t in the trace to be $t/(24 * 60 * 60)$. This means two peers online at the same time of day are online concurrently in the stacked traces regardless of which day of the month they were online.

Overhead. We show the overhead of constant-rate, broadcast (*e.g.,* P^5), P2P (*e.g.,* Tarzan), and *Aqua* designs in Fig. 3. *Aqua* incurs significantly less overhead than other designs, mainly because ksets efficiently match peers' payload flows, requiring only moderate chaff traffic. Below, we describe the results in more detail.

Constant-rate. In the Constant-rate design, peers exchange chaffed traffic at their capacity rate. In Fig. 3, the median overhead over all peers is 0.95 because half the peers utilize at most 5% of their bandwidth capacity over the duration of the trace. As a result, these peers must generate 95% of chaff to maintain their target rate. Hence, the overhead is inversely proportional to the peers' bandwidth utilization, which leads to high overhead, even in the case of BitTorrent.

Broadcast. With the Broadcast design, each message is broadcast to every group member. With k group members, one unit of payload traffic generates $(k - 1)$ units of chaff. Thus, the overhead of each group member is approximately *(k-1)/k* on average, approaching 1 as k increases. We see in Fig. 3 that the median overhead of the broadcast design is always above 0.95. For example, for k=10, the peers' median overhead is 0.97. The higher overhead in the simulations is explained by peers running out of payload during their slot and padding with chaff. High overhead makes the broadcast design unsuitable for bandwidth intensive applications like BitTorrent.

P2P. In the P2P design, each peer keeps constant rate streams with a fixed number of other peers, and inserts payload into these streams; that is, at every time unit, the sum of chaff bytes and real traffic bytes is constant. When forming the P2P overlay paths, flows may be unevenly distributed over streams (*i.e.,* several flows may share the same links), meaning some number of streams generate only chaff traffic (pure overhead) while others generate only payload traffic. As the number of P2P connections n increases the number of chaffed overlay links increases, while the number of payload flows remains constant. Thus, the overhead ratio increases for P2P as k increases.

Aqua. For *Aqua*, the overhead comes from chaff traffic generated by kset members with less payload than the kset transfer rate. This chaff traffic is generated only while a kset is active. Because ksets have relatively short lifetimes, peers spend less time generating chaff traffic than in the other designs, thus contributing to a significantly lower overhead. Furthermore, we observe that the overhead ratio decreases with larger k because the amount of traffic (chaff or payload bytes) sent over each link is constrained by the slowest connection. Larger ksets are likely to include relatively low-bandwidth peers, which reduces the rate at which each host sends traffic. These lower rates lead to relatively lower overhead for flows containing only chaff traffic.

Throttling. We show the throttling results for the constant-rate, broadcast, P2P, and *Aqua* designs in Fig. 4. At a high level, the volumes of chaff traffic required by Broadcast and P2P are quite large; *Aqua* reduces this throttling by grouping peers with similar bandwidth demands into the same ksets, then tearing down ksets as soon as all members' flows have completed. The constant-rate design has no throttling, as expected, at the expense of high overhead.

Constant rate. The constant-rate design has no throttling because all endpoints always send or receive traffic at their capacity rate.

Broadcast. Throttling in the Broadcast design depends on the groups' capacities and sizes k, and the payload bandwidth demand of peers. The impact of groups' capacities on

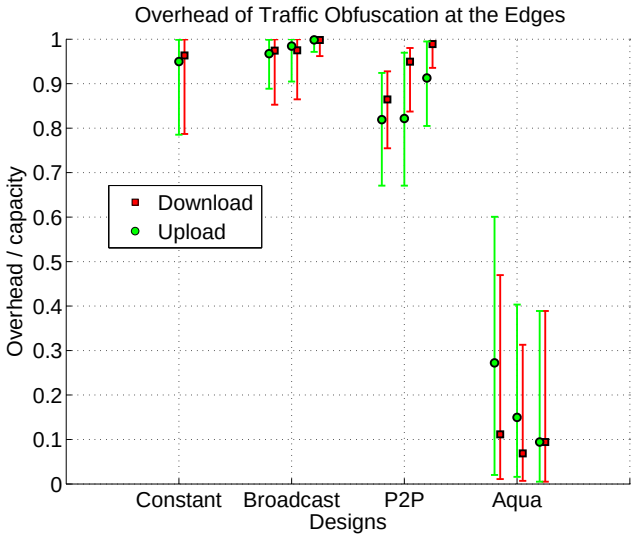

Figure 3: Overhead of traffic obfuscation at the edges for the different designs. Shown on the y axis are the median, 10th and 90th percentile of the overhead. The bars from left to right for Broadcast, P2P, and *Aqua* correspond to k equals 10, 100, and 1,000. Note that Constant-rate is not parameterized by k, so we show its overhead for the number of peer in our traces.

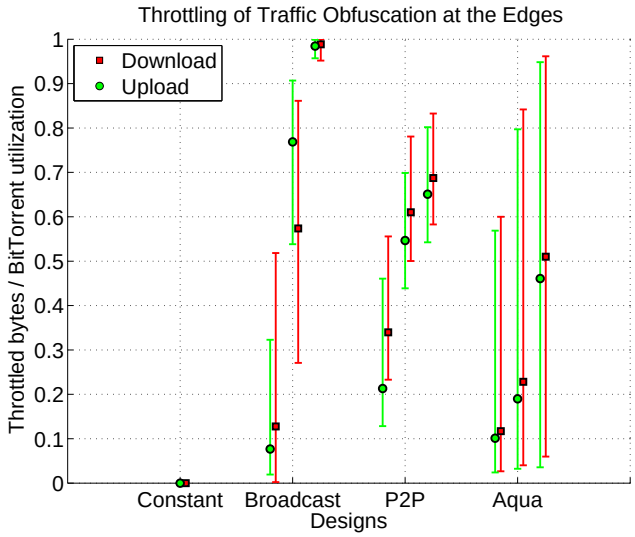

Figure 4: Throttling of the different designs due to traffic obfuscation at the edges. Shown on the y axis are the median, 10th and 90th percentile of the throttling. The bars from left to right for Broadcast, P2P, and *Aqua* correspond to k equals 10, 100, and 1,000.

throttling is minimal in our simulations because we assign peers to broadcast groups based on their bandwidth capacities. The main factor affecting the degree of throttling is k because the capacity available for payload in a broadcast group is $1/k$. We see in Fig. 4 that there is little throttling for $k = 10$ because there is enough available bandwidth to satisfy peers' demand rate (5% in the median case). However, throttling increases considerably for larger k because the bandwidth available for payload is insufficient to meet peers' demands.

P2P. In the P2P model, the number of overlay links increases with n and so the capacity of individual P2P links decreases. As a result, the throttling increases with the size of the anonymity set.

Aqua. With *Aqua*, the median download throttling remains between 0.02 and 0.21 for all values of k. When all peers in a kset have sufficient capacity, flow rates ramp up without throttling. When a fraction of peers finish their flows before others, there is throttling for idle peers until all peers can ramp down simultaneously. The impact of this throttling is limited because flows in BitTorrent are short-lived.

Throttling increases with $k = 1,000$ because it is more likely that a high-capacity peer joins a kset with a lower-capacity peer, forcing the high-capacity peer to throttle its throughput to that of the lowest-capacity peer. A potential optimization is to prevent such ksets from forming unless there is no alternative option, or even delaying a transfer to wait for a sufficient number of peers with similar capacity to appear.

4.4.2 Mix performance

Next, we present the results for network overheads incurred by routing traffic over multiple hops, where traffic is mixed and chaffed. Note that these results apply only to designs that use mixes, namely, Constant-rate and *Aqua*. We find that, in general, *Aqua* has low overhead in the core, because multipath routing evenly balances payload traffic across mixes, reducing the need for chaff.

Methodology. For this evaluation, we use a twenty-mix full mesh topology, and simulate single-path routing, multi-path routing, and perfect routing. Here, perfect routing assumes that flows can be split and routed perfectly among all the mixes and links, i.e., distributed evenly so as to minimize the necessary chaff traffic. For multi-path routing, the number of paths for each flow is the number of mixes minus two, as described in Section 3. We provision the mix network with aggregated bandwidth required by the worst case, Constant-rate with single path routing.

In Constant-rate simulations, each pair of mixes exchange traffic at the same constant rate, which is the maximum payload rate on all links over all the simulation time. In *Aqua* simulations, we allow link rates to vary uniformly every hour, so at any time, the rates on all links are identical, and equal to the maximum link payload rate in this hour.

Overhead. We show the overhead of Constant-rate and Aqua in Fig. 5, each with three routing schemes, single-path routing, multi-path routing, and perfect routing, respectively. Although Constant-rate with single-path routing has a median overhead above 0.49, the overhead is reduced to 0.1 when multi-path routing is used.

Compared to Constant-rate, *Aqua* mixes can dynamically change rates. As a result, overheads are significantly lower—less than 0.01 in all cases. Taking *Aqua* receive rates as an example, we find that overhead is 0.0093 for single-path routing, and less than an order of magnitude smaller when using multi-path (0.00088). While the absolute difference is small in this case, it is relative to a large capacity required for Constant-rate traffic. Thus, in practice multipath can

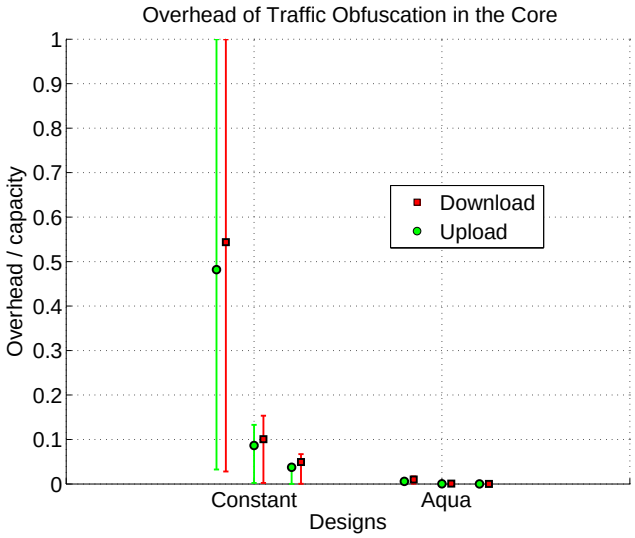

Figure 5: Overhead for traffic obfuscation in the core for different designs. Shown on the y axis are the median, 10th and 90th percentile of the overhead. The bars from left to right for Constant-rate and *Aqua* correspond to single-path, multi-path, and perfect routing.

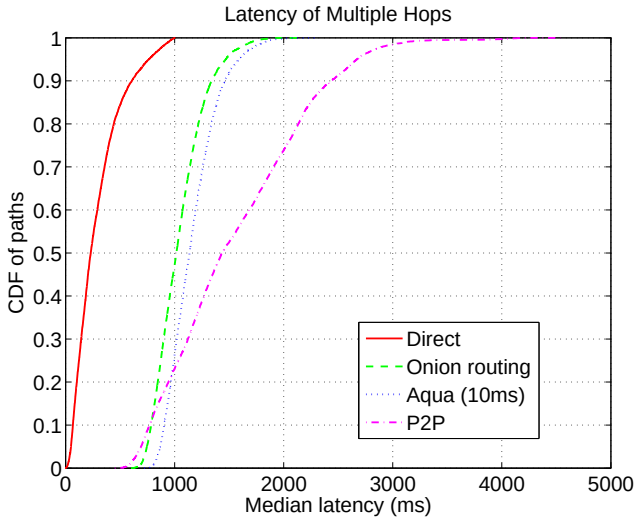

Figure 6: Median latency for paths using the different designs, based on latency data gathered from end users. The cumulative distribution function shows that *Aqua* with 10 ms delay has only 12% higher latency than onion routing and 20% lower latency than an approach that routes exclusively over end users (labeled *P2P*).

lead to substantial overhead savings relative to single-path routing.

4.4.3 Multi-hop Latency

Next, we consider the question of how much additional latency is imposed by each anonymity system and determine its impact on transfer rates. Our key finding is that latency for *Aqua* is comparable to that of onion routing, and the added latency will not significantly impact the rates for the vast majority of BitTorrent flows in our dataset.

Methodology. Our deployment model for *Aqua* includes mixes that are located in hosting providers in the core of the network, *e.g.,* in well-connected points of presence (PoPs). To model the latency overhead in *Aqua*, we would like to use empirical delays from peers in edge networks to hosts in popular PoPs, and delays between hosts in popular PoPs. Furthermore, we want to include a set of delays exclusively between peers in edge networks to compare with an approach such as Tarzan.

To address these needs, we use latencies gathered from 200 million traceroutes between Ono users between March 1, 2010 and June 1, 2010. These measurements provide end-to-end latency measurements between end-users, as well as hop-by-hop delays to intermediate routers.

Estimating the delay through an *Aqua* deployment requires a set of mix locations and the latencies between them. We further leverage the traceroute dataset by assuming mixes will be placed in networks that are traversed by a large fraction of paths between end users. Specifically, we count the number of distinct source/destination pairs that traverse each AS boundary (using IP-to-AS translation), then use the 100 most popular networks as mix locations. We obtain the latency between a client and mix using the traceroute-based delay measured when the client probed the mix location. Because most pairs of mix locations are traversed by at least one end-to-end traceroute measurement, we obtain a nearly complete matrix of delays between our selected mix locations. Our simulation does not use paths for which our dataset does not contain a delay meaurement.

To avoid being biased by paths through large tier-1 networks, we pick at most two mix locations per AS. The resulting set of locations spans 70 ASNs and 25 countries in North America, Europe, Asia and South Africa.

For each set of latencies measured directly between endpoints (*i.e.,* the last hop of a traceroute), we require there be at least three sample values and take the median latency. We further filter out latencies that are unreasonably large (>1000 ms), as they are indicative of severe buffer bloat or other transient performance problems that we do not intend to capture with this analysis. Specifically, we wish to capture the additional delay from mix hops that are located in the core of the network and not subject to last-mile delays.

Results. We obtained simulation results for latencies according to the following models:

- ***Aqua.*** There are two concatenated mix circuits, meaning 6 mix hops in each direction (12 hops total). Each mix is located in a well-connected network as described above, endpoints are located in access networks, and we show results for a batch period of 10 ms.
- **Circuit-switched (Onion).** There are two concatenated onion circuits, resulting in six distinct hops between endpoints, traversed once in each direction (12 hops total). Each onion hop is located in a PoP, the endpoints are located in access networks.
- **P2P (Tarzan).** There are six onion hops, each of which is located in an access network.

We sample 100,000 pairs of hosts, simulate latency over 1000 randomly selected paths, then report results for the median latencies[2]. Figure 6 shows CDFs, where each point (x, y) represents the latency value (x) for a single source-destination pair (cumulative fraction, y). There is one curve for the direct path between a source and destination (labeled "direct"), and one curve each for the median latencies in *Aqua*, onion routing and P2P routing (labeled as "[*Aqua*,Onion routing,P2P]").

While it is not surprising that there are higher delays in *Aqua* from six additional overlay hops in each direction compared to direct-path routing, the median latencies stay within a constant factor of the direct-path delays. For example, the median delay for *Aqua* latency is approximately five times the median delay for direct path latency when using a delay of 10 ms at each hop. Note that when compared to onion routing, *Aqua* imposes only an additional 120 ms of delay due to buffering for chaffing at mixes. Thus, *Aqua* offers resilience to timing attacks, at the cost of a modest additional delay (12%) over Tor.

We observe that P2P routing has nearly 20% larger the median latency when compared to *Aqua* with 10 ms delays at each hop. The reason is the latency of traversing last-mile links at each hop in P2P routing, when compared to latencies between mixes in *Aqua* located in the network core.

In the worst case, both *Aqua* and onion routing may suffer latencies of one or two seconds. Focusing at the top of Fig. 6, *Aqua*'s latency is a full second faster than P2P routing. While both approaches suffer large delays in the worst case, the impact on end-to-end performance differs. For onion routing, which is circuit-based, a poor circuit choice lasts until the circuit is torn down—potentially affecting many flows. Importantly, *Aqua* picks paths on a per-packet basis, limiting the impact of such poor choices.

Impact of Latency. We now consider the impact of additional delays on the maximum rate for a TCP flow over a path and evaluate whether it impacts end-to-end performance for an application like BitTorrent. We use the traditional formula

$$Rate = (1.2 * MSS)/(RTT * \sqrt{loss})$$

and estimate the steady-state transfer rate for a flow experiencing 1% packet loss, using a maximum segment size (MSS) of 536 bytes and the RTT of the 90th percentile of the distribution in Fig. 6. We then compare this steady-state rate with the distribution of maximum transfer rates for peers in our dataset (these are considered the peer capacity). For each of the routing models considered in the previous section, we estimate the end-to-end rate as the minimum of the per-hop TCP connection in the multi hop overlay. Further, we focus on upload rates because asymmetric bandwidth in access networks implies that senders' transfer rates are the main bottleneck in BitTorrent. Last, we assume there are 10 simultaneous flows for each user, based on the average number of parallel flows per peer in our dataset (9.6).

We find that the steady-state transfer rates for *Aqua*, Onion and P2P routing are at least 10, 12.3 and 6.1 KB/s, respectively, for 90% of peers. By comparison, 90% of flows in our BitTorrent traces have a send rate less than 13.4 KB/s. We then compare these per-flow rates with empirically measured peer capacities. When there are 10 parallel flows, we find that only 14.6% of paths experience throttling in the upload direction for *Aqua* (10% for Onion and 27% for P2P). Thus, we believe that the delays incurred by overlay routing should not significantly reduce available capacity in the system for the vast majority of peers and paths.

[2]We found the average and median distributions to be nearly identical and thus omit the average for clarity in the figures.

4.5 Caveats

Simulation limitations. Our simulation approach captures realistic session times, throughput capacities and connection patterns for BitTorrent. However, our simulation does not account for the following factors.

First, our trace data contains transfer rates along direct paths between hosts, but the anonymous communication designs induce additional latency from multiple hops. We do not model these delays in our simulation but we showed that they do not significantly impact the steady-state transfer rates of existing direct-path flows.

Next, we do not simulate delays or bandwidth constraints at mixes, except for those required for batching. We believe this is reasonable because we expect mixes to be deployed in hosted data centers where bandwidth and processing power are sufficient to support large numbers of users.

Lastly, throttling and kset formation alter the empirical transfer rates and session durations recorded in the trace. This, in turn, would change how BitTorrent establishes future connections in a real implementation. We do not model these second-order effects.

Dataset limitations. There are several limitations of the Ono dataset. We use an extensive dataset of empirical latency and flow samples from end users that is representative of paths between users in a P2P network. This does not necessarily reflect the paths between end-users and content providers such as CDNs and Web servers. Likewise, we do not consider the performance impact of load at each mix, though we expect the mix locations to be well provisioned. Finally, there is bias in our empirical dataset in that it represents locations where BitTorrent usage is high. We believe this is also where *Aqua* is most likely to be popular if it were deployed today.

4.6 Preliminary implementation results

At the time of this writing, we have implemented *Aqua*'s multipath routing component in Tor v0.2.2.37, which adds approximately 3,000 lines of C code. The implementation of traffic obfuscation as described in Section 3 is still in progress. To quantify the CPU and memory usage of multipath routing, we performed a simple experiment that consists of a client downloading a 100MB file from an Apache server in a well provisioned network with nine *Aqua* prototype mixes. The client's download rate (averaged over 5 runs) is 9.12Mb/s for a direct connection, 7.52Mb/s with single-path Tor, and 7.68Mb/s with the *Aqua* prototype. On average, Tor proxies use 2% of CPU and 45MB or memory and the *Aqua* prototype mixes use 2.1% of CPU and 46MB of memory. As we can see, the *Aqua* prototype with multipath routing introduces negligible overhead relative to Tor.

5. CONCLUSION

We have introduced *Aqua*, an efficient traffic-analysis resistant anonymity network for BitTorrent applications. *Aqua* derives its efficiency from using different traffic obfuscation mechanisms in the core and at the edges of the network. In the core, *Aqua* employs uniform rate chaffing

to take advantage of infrequent changes in aggregate traffic. Furthermore, multipath routing disperses traffic hot spots in the core to minimize chaff overhead. At the edges, *Aqua* dynamically groups peers with correlated payload traffic patterns and couples their rate changes to efficiently provide k-anonymity. We showed that these mechanisms scale to much larger anonymity sets than existing work while achieving latency low enough to have minimal impact on TCP bulk performance in BitTorrent workloads. These properties allow *Aqua* to anonymize BitTorrent traffic with high bandwidth efficiency. Our future work aims at providing strong anonymity to a broader range of applications.

Acknowledgements. We thank the anonymous reviewers and our shepherd, Katerina Argyraki, for their helpful feedback.

6. REFERENCES

[1] HideMyAss.com doesn't hide logs from the FBI. `http://blog.hidemyass.com/2011/09/23/lulzsec-fiasco/`.

[2] Private communication with a large European ISP, 2012.

[3] Bamford, J. The NSA Is Building the Country's Biggest Spy Center (Watch What You Say), 2012. `http://www.wired.com/threatlevel/2012/03/ff_nsadatacenter/all/1`.

[4] Berthold, O., Federrath, H., and Köpsell, S. Web MIXes: A system for anonymous and unobservable Internet access. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (July 2000), H. Federrath, Ed., Springer-Verlag, LNCS 2009, pp. 115–129.

[5] Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (February 1981).

[6] Chaum, D. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology 1* (1988), 65–75.

[7] Choffnes, D. R., and Bustamante, F. E. Taming the torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In *Proceedings of SIGCOMM* (August 2008).

[8] Dai, W. Pipenet 1.1. Post to Cypherpunks mailing list, November 1998.

[9] Danezis, G. The traffic analysis of continuous-time mixes. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)* (May 2004), vol. 3424 of *LNCS*, pp. 35–50.

[10] Danezis, G., Dingledine, R., and Mathewson, N. Mixminion: Design of a Type III Anonymous Remailer Protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy* (May 2003), pp. 2–15.

[11] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium* (August 2004).

[12] Freedman, M. J., and Morris, R. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS 2002)* (Washington, DC, November 2002).

[13] Furukawa, J., and Sako, K. An efficient scheme for proving a shuffle. In *In Proc. of CRYPTO'01* (2001), Springer-Verlag, pp. 368–387.

[14] Goldschlag, D. M., Reed, M. G., and Syverson, P. F. Hiding Routing Information. In *Proceedings of Information Hiding: First International Workshop* (May 1996), R. Anderson, Ed., Springer-Verlag, LNCS 1174, pp. 137–150.

[15] Gülcü, C., and Tsudik, G. Mixing E-mail with Babel. In *Proceedings of the Network and Distributed Security Symposium - NDSS '96* (February 1996), IEEE, pp. 2–16.

[16] Juels, A. Dining cryptographers revisited. In *In Advances in Cryptology (EUROCRYPT 2004), Springer LNCS 3027* (2004), pp. 456–473.

[17] Katti, S., Jeff, J. C., and Katabi, D. Information slicing: anonymity using unreliable overlays. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation* (Berkeley, CA, USA, 2007), NSDI'07, USENIX Association, pp. 4–4.

[18] Landsiedel, O., Pimenidis, L., Wehrle, K., Niedermayer, H., and Carle, G. Dynamic multipath onion routing in anonymous peer-to-peer overlay networks. In *GLOBECOM* (2007), pp. 64–69.

[19] Larsson, S., Svensson, M., de Kaminski, M., RÃűnkkÃű, K., and Olsson, J. A. Law, Norms, Piracy and Online Anonymity: Practices of De-identification in the Global File Sharing Community. *Proceedings of Journal of Research in Interactive Marketing 6*, 4 (2012).

[20] Levine, B. N., Reiter, M. K., Wang, C., and Wright, M. K. Timing attacks in low-latency mix-based systems. In *Proceedings of Financial Cryptography (FC '04)* (February 2004), A. Juels, Ed., Springer-Verlag, LNCS 3110, pp. 251–265.

[21] Möller, U., Cottrell, L., Palfrader, P., and Sassaman, L. Mixmaster Protocol — Version 2. IETF Internet Draft, July 2003.

[22] Neff, C. A. A verifiable secret shuffle and its application to e-voting. ACM Press, pp. 116–125.

[23] Øverlier, L., and Syverson, P. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy* (May 2006), IEEE CS.

[24] Rennhard, M., and Plattner, B. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)* (Washington, DC, USA, November 2002).

[25] Shane, S., and Burns, J. F. U.S. Subpoenas Twitter Over WikiLeaks Supporters, 2011. `http://www.nytimes.com/2011/01/09/world/09wiki.html`.

[26] Sherwood, R., Bhattacharjee, B., and Srinivasan, A. P5: A protocol for scalable anonymous communication. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy* (May 2002).

[27] Shostack, A., and Goldberg, I. Freedom systems 1.0 security issues and analysis. White paper, Zero Knowledge Systems, Inc., October 2001.

[28] Syverson, P., Tsudik, G., Reed, M., and Landwehr, C. Towards an Analysis of Onion Routing Security. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (July 2000), H. Federrath, Ed., Springer-Verlag, LNCS 2009, pp. 96–114.

[29] Wang, X., Chen, S., and Jajodia, S. Tracking anonymous peer-to-peer voip calls on the internet. In *Proceedings of the ACM Conference on Computer and Communications Security* (November 2005), pp. 81–91.

[30] Wolinsky, D. I., Corrigan-Gibbs, H., and Ford, B. Dissent in numbers: Making strong anonymity scale. In *Proc. OSDI* (2012).

[31] Yang, Z., Zhong, S., and Wright, R. N. Anonymity-preserving data collection. In *In KDD'05: Proc. of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining* (2005), pp. 334–343.

[32] Zhu, Y., and Bettati, R. Unmixing mix traffic. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2005)* (May 2005), pp. 110–127.

[33] Zhu, Y., Fu, X., Graham, B., Bettati, R., and Zhao, W. On flow correlation attacks and countermeasures in mix networks. In *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)* (May 2004), vol. 3424 of *LNCS*, pp. 207–225.