$\mathcal{P}^5$: A Protocol for Scalable Anonymous Communication *

Rob Sherwood Bobby Bhattacharjee Aravind Srinivasan
University of Maryland, College Park, Maryland, USA
{*capveg, bobby, srin*}*@cs.umd.edu*

Abstract

We present a protocol for anonymous communication over the Internet. Our protocol, called $\mathcal{P}^5$ (Peer-to-Peer Personal Privacy Protocol) provides sender-, receiver-, and sender-receiver anonymity. $\mathcal{P}^5$ is designed to be implemented over the current Internet protocols, and does not require any special infrastructure support. A novel feature of $\mathcal{P}^5$ is that it allows individual participants to trade-off degree of anonymity for communication efficiency, and hence can be used to scalably implement large anonymous groups. We present a description of $\mathcal{P}^5$, an analysis of its anonymity and communication efficiency, and evaluate its performance using detailed packet-level simulations.
Keywords: Anonymous Communication, Privacy, Peer-to-Peer

1 Introduction

We present the Peer-to-Peer Personal Privacy Protocol ($\mathcal{P}^5$) which can be used for scalable anonymous communication over the Internet. $\mathcal{P}^5$ provides sender-, receiver-, and sender-receiver anonymity, and can be implemented over the current Internet protocols. $\mathcal{P}^5$ can scale to provide anonymity for hundreds of thousands of users all communicating simultaneously and anonymously.

A system provides *receiver anonymity* if and only if it is not possible to ascertain who the receiver of a particular message is (even though the receiver may be able to identify the sender). Analogously, a system provides *sender anonymity* if and only if it is not possible for the receiver of a message to identify the original sender. A property common to all anonymous systems is their inability to provide *perfect* anonymity. This is because it is usually possible to enumerate all senders or recipients of a particular message. In general, the degree of sender/receiver anonymity is measured by the size of the set of people who *could* have sent/received a particular message. There have been a number of systems designed to provide receiver anonymity [7, 11], and a number of systems that provide sender anonymity [22, 26]. In these systems, individual senders (or receivers) cannot determine the destination (or origin) of messages beyond a certain set of hosts in the network. Note that receiver anonymity is required by applications where the server needs to remain hidden, e.g., censorship resistant publishing, anonymous file distribution, or any peer-to-peer application.

We assume that an adversary in our system may passively *monitor every packet on every link* of a network, and is able to correlate individual packets across links. Thus, the adversary can mount any passive

*This work was supported by grants from the National Science Foundation (ANI0092806 and 0208005) and under ITR Award CNS-0426683. This paper is an extended version of the original IEEE Security and Privacy 2002 conference paper. A list of extensions is available in Appendix A.

attack on the underlying networking infrastructure. However, the adversary is not able to invert encryptions or read encrypted messages. The adversary can also read all signaling messages in the system. A direct implication of this attack model is that the adversary can totally enumerate the network (e.g., IP) addresses of the members of the system. Further, assume that all members of the system have a digital pseudonym (e.g., a public/private key pair), which by assumption, the adversary can also enumerate. The adversary, however, cannot map particular keys to specific addresses. Put another way, the crux of our (or any) anonymity protocol is to conceal the mapping between digital pseudonyms and network addresses from the adversary. Last, assuming digital pseudonyms are implemented using public/private key pairs, we do not assume the existence of a full PKI. Instead, we assume that if two parties wish to communicate, they are able to retrieve the public keys out of band (this is discussed in more detail in Section 2.7).

The motivation for an attacker that can monitor arbitrary links comes from two observations. First, if a protocol is secure against a strong adversary that can monitor arbitrary links, then it is secure against a weaker adversary than can only monitor a subset of links. For example, an attacker may compromise an ISP and monitor all incoming and outgoing traffic. In this case, any clients of the ISP are susceptible to exposure (and thus have no anonymity) unless the anonymous protocol defends against this model. Another weaker, yet more realistic, adversarial model is the *trap and trace* model. Under this model, the attacker subverts the internal auditing equipment of the telephone company or ISP(s) to follow specific links hop by hop iteratively back to the source (from the Wiretapping statues[3], this is known as trap and trace). In general, it is difficult to analyze a protocol's security under a multitude of weaker passive attack models, so we assume a model that is a superset of all passive adversarial models. Second, we believe that this global passive attack is indeed possible as evidenced by the Echelon[4, 24] and Carnivore[1] projects. While the exact details of Echelon are not public, the project's apparent goal is to gather intelligence by monitor multiple diverse sources of information, including global Internet traffic, i.e., exactly the proposed adversarial model. Some sources[18] even claim that Echelon intercepts an estimated 90 percent of global Internet traffic, motivating the existence of a protocol such as $\mathcal{P}^5$. Similarly, the FBI's Carnivore system is installed in a number of ISPs across the country, and can be configured, by court order, to capture packets to a given specification. While the use of Echelon and Carnivore in the United States is strictly bound by constitutional checks and balances, these systems do provide constructive evidence for our attack model. In many parts of the world, the Internet is pervasive enough that it is not possible to completely shut it down; however, online communication is extensively monitored and even used as a basis for political persecution. In these countries, a system like $\mathcal{P}^5$ will likely prove useful for free exchange of ideas online.

Our system provides receiver and sender anonymity under this rather strong adversarial model, and additionally provides the sender-receiver anonymity (or *unlinkability*) property. With sender-receiver anonymity, the adversary cannot determine if (or when) any two parties in the system are communicating. $\mathcal{P}^5$ maintains anonymity even if one party of a communication colludes with the adversary who can identify specific packets sent to or received from the other end of the communication. Unlike previous known solutions, $\mathcal{P}^5$ can be used to implement scalable wide-area systems with many thousand active participants, all of whom may communicate simultaneously.

1.1 A Naive Solution

Consider a global broadcast channel. All participants in the anonymous communication send fixed length packets in to this broadcast channel at a fixed rate. These packets are encrypted such that only the recipient of the message may decrypt the packet, e.g., by using the receiver's published public key. Assume that there is a mechanism to hide, spoof, or re-write sender addresses, e.g., by implementing the broadcast channel using an application layer peer-to-peer ring. Every message is hop-by-hop encrypted, and thus is it not

possible to map a specific incoming message to a particular outgoing message (in essence, each node acts as a *mix*[6]). It is possible that a node may not be actively communicating at a given time, but in order to maintain the fixed communication rate, we require that the node send a packet anyway. Such a packet would be a dummy or *noise* packet, and any packet destined for a particular receiver would be a *signal* packet.

This system provides sender anonymity, since all messages to a given receiver (in the case of a application layer peer-to-peer ring) come from a single upstream node, and the receiver cannot determine the original sender of the message. This system also provides receiver anonymity because the sender does not know *where* in the broadcast channel the receiver is or which host or address the receiver is using. Lastly, this solution provides sender-receiver anonymity from a passive adversary since the adversary is not able to gain any information from monitoring any (or *all*) network links. Because all nodes send to the broadcast channel at a fixed rate, all nodes send and receive at a fixed rate independent of who they are communicating with, or how many signal messages are sent/received. Note that the adversary is not able to *trace* a message from sender to receiver because of the hop-by-hop encryption, and thus even if one end of the communication is colluding with the adversary, the anonymity of the other party is not compromised.

This naive solution does not scale well due to its broadcast nature. As the number of people in the channel increases, the available bandwidth for any useful communication decreases linearly, and the end-to-end reliability decreases exponentially. It is possible to increase the bandwidth utilization and reliability by limiting the number of users in a broadcast channel. One possible solution would be to create a set of multiple, independent channels, but then two parties who want to communicate may be unable to because they end up in different channels.

$\mathcal{P}^5$ is based upon this basic broadcast channel principle; we scale the system by creating a *hierarchy of broadcast channels*. Clearly, any broadcast-based system, including $\mathcal{P}^5$ will not provide high bandwidth efficiency, both in terms of how many bits it takes a sender–receiver pair to exchange a bit of information, and how many extra bits the network carries to carry one bit of useful information. Instead, $\mathcal{P}^5$ allows users to choose how *inefficient* the communication is, and provides a scalable control structure for securely and anonymously connecting users in different logical broadcast groups. We present an overview of $\mathcal{P}^5$ next.

1.2 Solution Overview

$\mathcal{P}^5$ scales the naive solution by creating a broadcast hierarchy. Different levels of the hierarchy provide different levels of anonymity, at the cost of communication bandwidth and reliability. Users of the system locally select a level of anonymity and communication efficiency and can locally map themselves to a level which provides requisite performance. At any time, it is possible for individual users in $\mathcal{P}^5$ to decrease anonymity by choosing a more communication efficient channel. Unfortunately, it is not possible to regain stronger anonymity, as we will show in Section 3.4. Also, it is possible to choose a set of parameters that is not supported by the system (e.g., mutually incompatible levels of bandwidth utilization and anonymity). The $\mathcal{P}^5$ system has been extensively modified from its original conference[25] version, and a summary of these modifications can be found in Appendix A.

1.3 Roadmap

The rest of this paper is structured as follows: we describe the $\mathcal{P}^5$algorithm in Section 2 with specific details in Section 3, and present a set of analytic bounds on performance in Section 4. Section 5, we analyze results from packet-level $\mathcal{P}^5$ simulator. We discuss related work in Section 6. We discuss future work and conclude in Section 7. Appendix [25] contains a summary of updates from the original conference paper.[25]

2 The $\mathcal{P}^5$ Protocol

$\mathcal{P}^5$ is based upon public-key cryptography. $\mathcal{P}^5$ does not require a global public-key infrastructure; however, we do assume that if two parties wish to communicate, they can ascertain each other's public keys using an out-of-band mechanism.

Assume N individuals[1] wish to form an anonymous communication system using $\mathcal{P}^5$. Assume each of these $\mathcal{P}^5$ users have public keys $K_0, \ldots K_{N-1}$. $\mathcal{P}^5$ will use these N public keys, called *communication keys* to create a *logical broadcast hierarchy*.

2.1 The $\mathcal{P}^5$ logical broadcast hierarchy

The $\mathcal{P}^5$ logical broadcast hierarchy is a binary tree ($\mathcal{L}$) which is constructed using the public keys $K_0, \ldots, K_{N-1}$. Each node of $\mathcal{L}$ consists of a bit string of a specified length. We present the algorithm assuming each node of $\mathcal{L}$ contains both a bit string and a bit mask. The bit mask specifies how many of the most significant bits in the bit string are valid. Though not strictly necessary, the addition of the bit mask will significantly ease our exposition. We use the notation $(b\,/\,m)$ to represent a node, i.e., a user, in $\mathcal{L}$ where b is the bit string, and m is the number of matched bits.[2]

The root of $\mathcal{L}$ consists of the null bit string and a zero length mask. We represent the root with the label $(\star\,/\,0)$. The left child of the root is the node $(0\,/\,1)$ and the right child is $(1\,/\,1)$. The rest of the tree is constructed as shown in Figure 1. For example, the node $(0\,/\,1)$ represents the bit string 0 and the node $(00\,/\,2)$ represents the bit string 00.

Each node in $\mathcal{L}$ corresponds to a single user of $\mathcal{P}^5$. Messages are (unreliably) forwarded to a subset of all members of the system. That subset of members, called a broadcast *channel*, is denoted CH$(b\,/\,m)$. The set of nodes in a channel is defined as follows: user A, joined at node $(b'\,/\,m')$, is in *channel* CH$(b\,/\,m)$ if and only if the k most significant bits of b and b' are the same, where k is defined to be $\min\{m, m'\}$. We call this common prefix testing the *min-common-prefix check*.

Thus, a message sent to a channel CH$(b\,/\,m)$ is sent to three distinct regions of the $\mathcal{L}$ tree:

- **Local:** A message sent on CH$(b\,/\,m)$ is sent to the $(b\,/\,m)$ node.
- **Path to root:** For each $m' < m$, this message is also broadcast to all nodes $(b_{m'}\,/\,m')$, where $b_{m'}$ denotes the m'-bit prefix of b.
- **Subtree:** Lastly, for all $m'' > m$, this message is also sent to all nodes $(b|\star\,/\,m'')$, where $b|\star$ is any bit string that begins with the string b.

Figures 2-5 show examples of broadcast channels.

Note that these broadcast channels should be implemented as peer-to-peer unicast trees in the underlying network (and not multi-cast trees). These channels may lose messages and require no particular consistency, reliability, or quality-of-service guarantees. We describe the precise networking and systems requirements of $\mathcal{P}^5$ and underlying protocols in Section 3.5.

In general, communication efficiency increases as the channel's size decreases, corresponding to the channel's mask size increasing. However, as we shall see, the anonymity of a node relates to the size of the channel which is communicates with, so this increase in efficiency comes at an expense of reduced anonymity. The depth of the $\mathcal{L}$ tree grows depending on the number of people in the system (N).

[1] It is entirely possible that the N keys belong to n different individuals, such that $n < N$. We discuss this issue in Section 2.6.

[2] This is similar to the notation used to name CIDR[13] address blocks

2.2 Mapping users to $\mathcal{L}$

We use a secure public hash function ($H(\cdot)$) to map users to $\mathcal{L}$ nodes. Consider user A, with public-key K_A. Assume $b_A = H(K_A)$. User A will join as a node in some channel of the form CH(b_A / m). The length of the mask m, i.e., which channel, is chosen independently by user A according to a local security policy, as described in Section 2.5. Given a chosen channel to join, the exact node that A has joined to in that channel should be kept secret, and it should not be possible to determine which *precise* node a user is joined to. The node choice is only limited to the set of nodes that pass the min-common-prefix check with the chosen channel. Thus, given a public key, it is public knowledge which set of nodes a user *may* be in, but it is difficult to determine which *specific* node in this set the user has chosen. This is the key to the anonymity of the system.

We say a channel c is common between A and B if and only if messages sent to c are forwarded to both A and B. Suppose A and B join channels CH(b_A / m_A) and CH(b_B / m_B) respectively, and assume both know each other's public key, K_A and K_B, respectively. Since A knows K_B, it can determine b_B (b_B=$H(K_B)$); however, A does not know the value of m_B. Even without any knowledge of the masks, A and B can begin to communicate by broadcasting to the entire system, i.e., by using the CH($\star$ / 0) channel. Unfortunately, this communication channel can be quite lossy since messages have a higher probability of getting lost in the channels "higher" up in the $\mathcal{L}$ tree and CH($\star$ / 0) is the highest/most lossy communication channel of all. Also, if channels are chosen uniformly at random as a function of the public keys (as above), then there is approximately 50% probability that CH($\star$ / 0) will be the *only* channel that A and B would have in common!

Our solution to this inefficient messaging is *each user joins multiple channels on the logical tree*. For each joined channel, users generate another public-private key pair, called *routing* key, which is kept distinct from its main public key: its *communications* key. These routing keys are generated locally, and do not require any global coordination. In fact, it should not be possible to map a user's routing key to their communication key, otherwise, the user's anonymity can be compromised (using an *Intersection* attack, Section 2.6).

When user A joins a channel c, it periodically sends a message to the channel listing other channels that it is joined to. These messages serve as a routing advertisement, and alert nodes about more efficient *lateral* routes along the lower levels of the $\mathcal{L}$ tree. In general, the advertisements from a node contains the set of channels it can directly reach, the set of channels it can reach using one other node, and so on. In effect, these t routing keys generate lateral edges in the tree. In Section 4, we show that typically, each user needs to join only a few channels (≤ 3) for any two users in $\mathcal{P}^5$ to have short paths (≤ 2 channel crossings) between them with high probability.

With each user joining multiple channels, the communication proceeds as follows. Instead of sending messages through CH($\star$ / 0), A discovers multi-hop lateral routing paths from any of the channels it has joined to one of the set of channels specified by B's communication key. By acquiring sufficient routing discovery messages, A then tries to send messages to some channel CH(b_B / m). As above, while A can calculate b_B, the problem is it does not know which m B has chosen. In other words, A knows a set of channels that B may be in, i.e., CH(b_B / 0), CH(b_B / 1), CH(b_B / 2), ..., but not which specific one. Let A guess a value m, and send a message to CH(b_B / m). If $m \leq m_B$, i.e., if the channel guessed is a superset of B's chosen channel, then B can respond with the correct m_B, and communication can proceed more efficiently. In this manner, A can probe for the correct channel that B has joined. However, if A has guessed an m that is strictly smaller than m_B, then B must *ignore*[3] the message. Failure to do so results

[3]In practice this translates to not passing the message up to the application layer

in a *difference* attack (Section 2.6). Lastly, note that the communication efficiency is upper bounded by the largest of the two channels chosen by either of the participants in a communication.

We note that a user joins a set of channels only when it enters the system, and should not change the set of channels it is part of. Otherwise, once again, yet another intersection attack becomes feasible that can compromise their anonymity.

2.3 Signal and Noise

Assuming packet sources cannot be traced from the broadcast messages (See Section 3.1 for the precise packet format), the protocol as described provides sender and receiver anonymity. We assume that each messages is of the same size and is encrypted *per-hop*, and thus it is not possible to map an outgoing message (packet) to a specific packet that the node received in the past. However, a passive observer can still mount an easy statistical attack and trace a communication by *correlating* a packet stream from a communicating source to a sink.

Thus, we add the notion of *noise* to the system. The noise packets should be added such that a passive correlation attack becomes infeasible. There are many possible good noise-generation algorithms, and we use the following simple scheme.

Each $\mathcal{P}^5$ user, at all times, generates fixed amount of traffic destined to their advertised channel. A packet transmitted from a node is one of the following:

- A packet (noise or signal) that was received from some incoming interface that this node is forwarding onto some other channel(s). (The precise forwarding rule for $\mathcal{P}^5$ is described in Section 3.2).
- A signal packet that has been locally generated.
- A noise packet that has been locally generated.

Note that a critical property of this system is that to an external observer, there is no discernible difference between these three scenarios. In general, only the source and destination of a communication can distinguish between noise and signal packets. They are treated with equal disdain at all other nodes.

Message Discard Algorithms In any communication system without explicit feedback, e.g. our channel broadcasts, message queues may build at slow nodes or at nodes with high degree. In $\mathcal{P}^5$, members may simply drop any message they do not have the bandwidth or processing capacity to handle. The end-to-end properties of the system depend upon how messages are dropped. We have considered two different dropping algorithms:

- Uniform drop: This is the simplest scheme in which messages from the input queue are dropped with equal probability until the input queue size is below the maximum threshold.
- Non-uniform drop: In this scheme, messages which are destined to larger channels, i.e., higher up in $\mathcal{L}$ are dropped preferentially.

  We have experimented with several variations of this scheme; the specific scheme which we use for our simulations drops packets destined for higher nodes with an exponentially higher probability.

  If most of the end-to-end paths in a $\mathcal{P}^5$ network can use lateral edges, i.e. between channels at the same logical height, then this scheme provides lower drop rate. However any communication that must use "higher" channels have proportionately high drops.

2.4 *SYN* Vs. Data Packets

One main draw back of the naive solution described in Section 1.1 is the need for at least one asymmetric decryption per packet. We can reduce this requirement to exactly one asymmetric decrypt per packet by encrypting a symmetric key in each packet, and then encrypting the packet data with the faster symmetric key, but this is still not sufficient. Modern asymmetric key algorithms are capable of maintaining on the order of 100 decrypts per second in software [2]. So, in order to saturate a 10Mb connection, this would require the fixed packet size to be 100Kb, which is inefficient for most applications. The general $\mathcal{P}^5$ system uses the asymmetric communications key to negotiate a per flow symmetric session key, so that further packets in the flow could be decrypted efficiently. We call the packets encrypted with the asymmetric key *SYN* packets, and correspondingly, the packets encrypted with a symmetric session key *data* packets. Each packet contains a plain-text bit denoting SYN versus data packet, so that the receiving node knows how to process it. However, since the SYN/data bit is in plain-text, adversaries can track connection initiations, which could be used to infer usage patterns, and ultimately to reduce the anonymity of the sender. The solution to this problem, as follows from the naive solution, is to send SYN packets at constant, fixed intervals, whether the sender is initiating a connection or not. So, like with packet transmission before, if no SYN packet is available to be sent, a *noise* SYN packet is sent in its place. This also implies that connection initiation has to wait until the next SYN packet time interval, introducing a connection latency to channel size linear trade off, as described in Table 1. Separate SYN and data packet queues should be kept, and the dropping rules as mentioned in 2.3 apply equally to both queues.

2.5 Anonymity Analysis

Assume node a has joined the channel CH(b/m).

Claim 2.1 *The anonymity of a node communicating using channel CH(b/m) is equivalent to the size of the set of members who are part the channel CH(b/m).*

Proof.

We consider the sender-, receiver-, and sender-receiver anonymity cases separately.

- Sender Anonymity

  Sender anonymity is the size of the set of the nodes that *could* have sent a particular packet to a given host (say B). A receiver who can only monitor their own links cannot determine the source of

Channel Size	Connection Latency (s)
25	1/4
50	1/2
100	1
200	2
400	4
800	8
1600	16

Table 1: Broadcast Channel Sizes relative to Connection Initiation Latency: assumes 100 decrypts/s

a packet since this information is never included in the packet. A receiver, in collusion with a all-powerful passive adversary, however, can enumerate the set of nodes who *could* have sent the packet by computing the transitive closure of the set of nodes who have a causal relationship with B. (In this case, nodes X and Y are causally related if X sent a packet to Y). This closure would be computed over some finite time window on the order of the end-to-end latency in the system.

However, in $\mathcal{P}^5$, a user connected to CH(b/m) sends packets at a constant rate (signal or noise), and these packets are received by all users in CH(b/m). Thus, there is a causal relationship between every user in a broadcast channel. Additionally, via lateral paths, nodes transmit packets between channels, and over time every node in the system is causally related to every other node in the system.

Suppose a malicious receiver tries to expose a sender that it is communicating with. Assume the communication stream is one way from the sender to the receiver. Given our assumption that the source IP address is obscured by the hop-by-hop retransmission, and that the packet contains no return addressing information, the malicious receiver is not able to distinguish the sender from *any* node in the system. It cannot even assert that a given set of nodes are not the sender.

- Receiver Anonymity

  When A sends a packet to B at channel CH((b_B/m_B), every member of CH(b_B/m_B) receives the packet. From the perspective of an external observer, the behavior of the system is exactly the same whether B receives the packet or not. Thus, B is indistinguishable from any other node in the same broadcast channel. Thus, B's anonymity is exactly equivalent to the set of all users who receive the packet, namely the channel CH(b_B/m_B).

- Sender-receiver anonymity

  Since all nodes in the system send at a constant rate, and all packets are pair-wise encrypted between each hop, we claim that it is impossible for a passive observer to distinguish noise from signal packets. Since the observer cannot distinguish signal packets, it cannot discern if or when A communicates, and thus, it cannot determine when A is communicating with any other node B.

□

Assume that the rate at which some user A sends packets does not change when it is sending signal versus noise packets. In this case, the distribution of packets, whether they are signal packets or noise, does not affect the security of the node. Thus, a nice property of our system is that the anonymity of any node A depends only upon the length of the mask that A is willing to respond to, not the rate at which it transmits.

2.6 Attacks

In this section, we outline a number of attacks common to all anonymity systems,and show why $\mathcal{P}^5$ resists them.

- **Correlation Attack:**

  As we have already alluded, a passive observer who is able to detect when signal packets are sent from the sender, and received by the receiver (independent of the content) is able to statistically correlate these events over time to discern that the two parties are communicating. If this adversary colludes with the sender, who knows the pseudonym of the receiver, it is possible to map the node's address to the pseudonym, breaking the anonymity. In $\mathcal{P}^5$ this attack is thwarted because the adversary cannot discern signal packets from noise.

- **Intersection Attack:** If an adversary knows that a user is two different sets U and V, then the anonymity of the user is reduced to $U \cap V$. If users are uniformly distributed across such sets that can be intersected, then the anonymity for any user in these reduces exponentially with the number of intersecting sets. For example, suppose users communicate using both their routing keys and their communication key. With each key, there is a corresponding set of users who may own that key. This leads to an intersection attack. This is the reason a user communicates with only one key in $\mathcal{P}^5$, and routing keys cannot be mapped back to the communication keys. Note that this is also the reason users cannot increase their anonymity beyond the smallest set they have ever been mapped to.

- **Difference Attack:** If an adversary can map the user to some set U and can assert that the user is not in some other set V, then it can map the user to the difference between these two sets, $U - V$.

  For example, suppose user A has joined CH(b / m). In this case, it should ignore packets sent to any channel CH(b / m') where $m' > m$. If A does respond/react/acknowledge such packets, then A is divulging where in the $\mathcal{L}$ it is *not*. That is because the receiver set of CH(b / m') is smaller than the receiver set CH(b / m), and now an adversary knows that A is not in the difference between the two channels.

- **DoS flooding attack:** Suppose a malicious user wants to reduce the efficiency of the system by sending a large number of useless packets. $\mathcal{P}^5$ can withstand this type of an attack since we impose a per-link queue limit, and all the extra packets from the malicious user will be dropped at the very first hop. Note that even the local broadcast channel is not affected by a DoS attack as long as the first non-colluding hop correctly implements its queue limits.

- **Forwarding DoS attack:** In $\mathcal{P}^5$ a selfish node may chose drop traffic that is not its own. If the source and destination of the traffic are in different channels, one potential method to mitigate this attack is to re-route traffic through another, potentially longer, path. While section 4 provides bounds on the minimum length of the path between two channels, there are multiple paths between any two channels. In the case where the source and the destination of the traffic are in the same channel, it is still possible to re-route the traffic around a malicious node by forwarding the traffic to another channel, and then back to the destination channel at a different point.

- **Drop Rate Partitioning:** Assume that an attacker is in an extended conversation with the victim, long enough to measure the average rate at which packets are dropped. Further assume that the attacker is colluding with a number of other hosts in the system, and can obtain average drop rate information for these hosts, as well as their distances in hop counts. In the case of the naive solution with the broadcast ring, two attackers, A_1 and A_2, can use this method to reduce the anonymity of the victim, V, as follows. Since there is a single path through the system, showing that A_1 has a lower drop rate when talking to V than it does when talking to A_2 implies that V is *between* A_1 and A_2, otherwise V is not between them. In either case, some amount of of anonymity is lost: this is a difference attack, as defined above.

  However, this attack breaks down in the general $\mathcal{P}^5$ case because there is not a single path through the system, and it is difficult to make assertions about the path from A_i to V relative to the path from A_i to A_j. For example, assume that A_1 is communicating with V, and with A_2, and has measured their respective drop rates, as above. Because there are in general many multi-hop lateral paths in the system between any two channel, and the sender can change between paths without notifying the receiver, it is not possible to make assertions about path characteristics over time.

2.7 Obtaining Public Keys Out of Band

$\mathcal{P}^5$ assumes that if Alice wishes to communicate with Bob, that she can obtain Bob's public key out of band from the system. However, Alice must be careful how she does this, as the act of fetching the key, for example querying a public server for Bob's key, could potentially break the anonymity of the communication before even using $\mathcal{P}^5$. One possibility is Alice could obtain the key from a trusted party. This is not as useless a technique as it would appear at first pass, because Alice needs someone to tell her about the existence of Bob, and to give Alice some reason for wanting to contact him. Another possible solution is a global directory service which publishes public keys, like a phone book. The downfall of this solution is that Alice has to keep state for all public keys in the *entire* system. A much more elegant solution, however, is to run a public key server as a service in the $\mathcal{P}^5$ system itself. The public key for the server could be well known in the system, and any communications to the server would be, by definition, as anonymous as any other communication in $\mathcal{P}^5$.

3 Details

We have implemented $\mathcal{P}^5$ in a packet level simulator, but the details from our simulation would be useful in a "real" implementation as well.

3.1 Packet Format

We use fixed length packets of size 1 KB. The fixed packet length is used to eliminate any information an adversary can gain by monitoring packet lengths. The 1 KB size was chosen arbitrarily as a trade off between packet fragmentation and communications efficiency, and is not integral to the system.

The $\mathcal{P}^5$ header only contains the identifier for the first hop destination channel (a (b/m) pair), and a flag denoting a SYN versus data packet, as described in Section 2.4. In our simulation, b is a 32 bit unsigned integer, and m is a 6 bit integer. Since packets may need to be encapsulated, and each packet is the same size, each packet also contains a padding field which is the size of the $\mathcal{P}^5$ header.

The payload of a $\mathcal{P}^5$ SYN packet contains a set of fixed size "chunks", each of which is encrypted with the receiver's public key. These chunks are formed naturally by many public-key encryptions. The decrypted data part of the $\mathcal{P}^5$ packet contains a checksum which the receiver uses to determine whether a packet is destined for itself or not. Each chunk can be decrypted independently; thus, a receiver does not need to decrypt an entire noise packet, it can discard a packet as soon as the first chunk fails its checksum. For efficiency, the first chunk of a signal packet may also include a symmetric cipher key for use in decrypting the other chunks, as symmetric cipher decryptions tend to be faster than asymmetric ones.

Regardless of whether a packet decrypts properly, the receiver schedules each packet for further delivery within the local channel using the forwarding rule described below.

The first chunk of a signal packet contains an encrypted bit which determines whether a packet should be forwarded onto some other channel, or whether the packet is destined for the current node. It also contains a channel identifier for the ultimate destination for the packet, which the current node uses to choose an outgoing channel.

When a node receives a packet with the "forward" bit set, it interprets the rest of the data as another $\mathcal{P}^5$ packet, and if possible, forwards it onto the specified channel. In the forwarding step, the process at a channel router is different depending on whether the packet is at its ultimate channel or not:

- If the packet is not at its final channel, the current node replaces the first chunk with a new chunk in which the "forward" bit is set, sets the proper ultimate destination channel, and encrypts this chunk with the public key of the next hop.

  If the packet is already is the destination channel, then the data part of the packet is already formatted with the proper address and has a valid first chunk encrypted by the public key of the intended recipient. The current node adds a last chunk at end of the packet with random bits to increment the packet length to the fixed system size.

If the "forward" bit is not set, then this signal packet is delivered to the local channel.

3.2 Forwarding within a channel

Since each broadcast channel is a tree, each node can use the following simple forwarding algorithm to forward a packet p along CH(b/m):

> Forward p to a node A if and only if only if p did not come in on A and if A passes the min-common-prefix check with respect to CH(b/m).

Note that it is important that the output order of the packets not be determined by the input order, else it becomes possible to correlate packets across successive nodes and trace communication between two parties. In other words, a *mix* [6].

3.3 Channel Selection and Join Algorithms

Analogous to the definition in [22], we define anonymity for a user A as the set S of users who are *indistinguishable* from that A, i.e., no other user or a passive adversary can resolve messages from A to a granularity finer than S. From Section 2.5, we know that in $\mathcal{P}^5$ the set S is equivalent to the size of the channel A joined. In this Section, we discuss how A selects a channel, and how to actually join.

We assume that each user u requires a minimal acceptable level of anonymity, i.e. each user requires their corresponding S set to be of a minimum size. We call this minimum set size the *security parameter*, and denote it with ψ_u. Each user u may also define a maximum required level of security (i.e. a maximum size of the corresponding S set) since this provides a bound on the communication inefficiency. We call this the *efficiency parameter*, and denote it with η_u. Thus, a node wants to find a channel that is "large enough" to fit their security parameter, but "small enough" to still be efficient. In other words, user A chooses a channel CH(b/m), such that $\psi_A \leq |\text{CH}(b/m)| \leq \eta_A$ with the following algorithm:

- A initially picks channel CH($\star/0$). If $\psi_A \leq |\text{CH}(\star/0)| \leq \eta_A$, then both security and efficiency requirements are met, and the node is done.

  If $\psi_A > |\text{CH}(\star/0))|$, A the entire system does not have enough members to provide the requisite anonymity. Thus, A forwards packets for other nodes and sends noise packets, but does not directly communicate with other nodes until such time as more nodes have joined the system to meet the security requirements.

- If $|\text{CH}(\star/0)| > \eta_A$, then this channel has too many nodes, and A picks the appropriate channel of the form CH($b/1$), i.e., one level lower on the tree. A repeats this procedure by incrementing m until it finds a CH(b/m) such that $\psi_A \leq |\text{CH}(b/m)| \leq \eta_A$.

Once the user has picked the appropriate broadcast channel CH(b/m) to join, they must physically join the tree to start receiving messages. The difference between the joined node and the joined channel is subtle but important. Joining channel CH(b/m) means that the user is one of the nodes (b'/n') that receives broadcast messages sent to that channel. Specifically, this implies that (b'/n') and CH(b/m) pass the min-common-prefix check. It should not be possible to discover a the specific node a user has joined at, only the channel they have chosen.

Given that the user decides to join channel CH(b/m) where $b = b_0 \dots b_{m-1}$, they pick which node to physically join using the following rules:

- Starting at the root node ($\star/0$), i.e. at the $i = 0$th level of $\mathcal{L}$ traverse down the tree following the left child if $b_i = 0$, or the right child if $b_i = 1$, i.e. a binary search.
- Once the node corresponding to (b/m) has been found, continue descending down the tree, picking the left or right child *uniformly at random*, until the bottom of the tree is reached.
- Physically join by connecting as a leaf to the last node visited.

More generally, a new user joins as a random leaf in the subtree rooted at (b/m), as shown in figures 6-8.

Clearly, it is possible to choose incompatible values ψ and η such that $\eta > |\text{CH}(b/m)|$ and $\psi > |\text{CH}(b/m+1)|$. In this case, the user can either change their security or efficiency parameter or wait in channel CH($b/m+1$), until $\psi \leq |\text{CH}(b/m+1)|$.

In our simulations, each user can determine the number of people in a channel by consulting an oracle which maintains an up to date list of channel memberships. In implementation, this information can be maintained in a secure distributed manner, either by the underlying application-layer multi-cast primitive, or at a well-known centralized topology server.

The topology server construct is needed if it is not possible to infer approximate channel sizes. The topology server keeps pairs of the form $\langle CH(\text{b} / \text{m}) : \text{IP address}\rangle$, where CH($b/m$) is the communication channel of the node. It is possible for the topology server to expose a user by providing false information (reporting a channel is large when it is in fact not). For extra security, the topology information can be replicated at l different topology servers. A user would only consider the minimum channel size reported by all topology servers; this way, a user can withstand up to $l-1$ colluding malicious topology servers. Similar techniques can be used to handle malicious topology servers who return a value smaller than the actual channel size (to make the communication inefficient). Also, note that the topology servers can also be used to find users on a specified channel, which is needed when a new user joins a channel. Further note that a node can keep track of the depth of the tree when it physically joins. Assuming that the tree is balanced because nodes join as random leaves, then the size of the channel can be estimated from the depth of the tree. This estimate provides an additional "sanity check" on the results returned by the topology server.

3.3.1 An Example of $\mathcal{P}^5$ Communication

Consider the following example: Alice has K_A such that $H(K_A) = 0000\dots$. Alice has locally determined that $\psi_A = 8$ and $\eta_A = 10$, so Alice wants to find a channel to join that is larger than 8 but smaller than 10. Assume the logical tree $\mathcal{L}$ looks like Figure 9 before she attempts to join. Node Alice can choose to any channel that passes the min-common prefix test with $H(K_A)$, specifically: CH($\star/0$) (28 nodes), CH($0/1$) (14 nodes), CH($00/2$) (8 nodes), CH($000/3$) (5 nodes), and CH($0000/4$) (5 nodes). Alice chooses to join CH($00/2$), because it is the only channel that fits both her security and efficiency parameters (Section 3.3).

To join CH(00 / 2), Alice joins as a random leaf of the tree rooted at node (00 / 2), specifically, Alice becomes node (0001 / 4). Additionally, Alice joins two other channels (Figure 10), CH(01 / 2) as node (0101 / 4) and CH(11 / 2) as node (1001 / 4), chosen uniformly at random to create lateral links across the tree. Alice generates random routing keys, R_1, R_2, and R_3, for each channel joined, and starts broadcasting routing availability messages, e.g., "Channel CH(00 / 2) is reachable from CH(01 / 2) using key R_2". Also, Alice starts to cache other node's routing availability messages in a local routing table.

Next, Alice decides she wants to talk to Bob, but she does not know what channel he is in. Assuming that Alice obtains Bob's private key, K_B, out of band (Section 2.7), she now knows which set of channels Bob must be in, but not which specific one. Using, the channel CH($\star$ / 0), i.e., the entire system, Alice sends a message to Bob to find out which specific channel he is in. Note that Alice is a member of CH($\star$ / 0), so she can send the message directly, as opposed to routing through an intermediary channel. Since the channel is quite lossy, Alice needs to resend the message a number of times, but eventually gets a response from Bob that he is in CH(11 / 2). This implies that Bob is one of the nodes that passes the min-common-prefix check with CH(11 / 2), but Alice does not know which specific node, leaving Bob anonymous to the size of CH(11 / 2). However, Alice is not a member of CH(11 / 2), so she is forced to route the message through a third party. After consulting her local cache of routing advertisements, Alice discovers that there exists a node C that routes from CH(10 / 2), which Alice is a member, to CH(11 / 2), where Bob is. As shown in Figure 11, Alice then broadcasts the message out on CH(10 / 2) to C's routing key. Node C reads the message, decrypts it, then rebroadcasts the message on CH(11 / 2) (Figure 12), where Bob receives it.

Note that despite the fact that Alice and Bob have both joined CH(10 / 2), they cannot communicate on that channel, or even know that they have both joined the same channel! Recall that both joined the channel to create lateral links across the tree, not to communicate. If, for example, Bob told Alice that he was also in CH(10 / 2), then Alice could intersect the set membership of channels CH(10 / 2) and CH(11 / 2), and reduce Bob's anonymity. In other words, Alice could mount an intersection attack on Bob.

3.4 Handling Node Churn

Suppose A is in channel CH(b / m), and the A's security and efficiency parameters are met, i.e., $\psi_A \leq |CH(\text{b} / \text{m})| \leq \eta_A$. As users join and leave the system, it is possible for the channel CH(b / m) to violate A's security or efficiency parameter. If too many nodes join A's channel and its efficiency parameter is violated, A can migrate to channel CH(b / $m+1$), as long as the security requirement is maintained.

In contrast, if too many nodes leave the system, A's security parameter can be violated. For example, suppose A with $\psi_A = 75$ and $\eta_A = 150$, joins a channel that contains 100 users. But, over time due to node leaves and failures, 40 nodes leave and the size of the channel is reduced to 60. As noted by Wright[28], this is an inherent problem in all known anonymity systems. A key property of $\mathcal{P}^5$ is that when a single node leaves, a channel's anonymity is diminished by *exactly one*. In other words, in the example above, A's security parameter is violated only once 25 nodes leave A's channel.

When a node's security parameter is violated, unfortunately A cannot *re-gain* any security by migrating "up" $\mathcal{L}$ (i.e. by decreasing m). This is because an intersection attack fixes S_A to the size of the smallest channel that A was part of since it initially joined. Thus, A's only available action is to leave the system, and rejoin under a different key.

We mitigate this costly operation as follows. Assume that the joining node has some knowledge of the average rate over time that nodes leave the system, α. The value α could be inferred from querying the topology servers over time, or maintained by the topology servers themselves. Given α, the newly joining node A picks a k such that the time $\frac{k}{\alpha}$ is sufficiently long for A to finish all transactions in the system, or long enough to amortize the cost of leaving the system and rejoining with a new key. Then, if ψ'_a is A's

actual security requirement, A joins with an effective security parameter of $\psi_a = \psi'_a + k$. In other words, A picks a channel to withstand up to k node leaves before being forced to rejoin the system with a new key.

Also, note that users who do leave the system open themselves to an intersection attack. An adversary can enumerate all nodes that have left the system, by our passive monitoring assumption, and assuming it can assert that a victim node has left the system (i.e., because the node is no longer responding to messages), can intersect that set with a victim's last known channel. Thus, there is incentive for nodes to persist in the system. This attack is not unique to $\mathcal{P}^5$ and is common to all systems with similar adversarial models.

3.5 The Network Abstraction and Processing Requirements

In this section we describe the precise networking requirements of $\mathcal{P}^5$. It was our design goal for $\mathcal{P}^5$ to be easily implementable using the current Internet protocols, as such our networking requirements are meager.

$\mathcal{P}^5$ requires the implementation of broadcast channels in which the source address cannot easily be determined. This can be efficiently implemented using a application-layer multi-cast protocol. The transport level requirements of $\mathcal{P}^5$ are minimal, and UDP would suffice as the transport protocol for $\mathcal{P}^5$ edges on the underlying physical topology. Lastly, note that during normal operation, $\mathcal{P}^5$ members only remain joined to the same set of channels; they only change channels if the overall security policy changes or if the channel dynamic changes drastically. Thus, the signaling load due to $\mathcal{P}^5$ is low, and since the topologies within each channel is relatively static, the $\mathcal{P}^5$ tree can be optimized to map efficiently on to the underlying physical topology.

Host Requirements $\mathcal{P}^5$ requires state, processing, and link bandwidth at each host. We discuss these requirements in turn:

- Suppose member A communicates using channel CH(b/m) at depth m in $\mathcal{L}$. In order to communicate within the channel, A may have to maintain next hop information about 2^m channels. For small m (e.g. $m < 16$), the state requirements are minimal. Note that unlike an IP router, A does not have to search its "routing table" for every packet; it only searches this table when it initiates a new data connection. Thus, this table can be maintained in secondary storage.

  If m is large (e.g. $m \geq 24$), it may not be feasible for a node to maintain information about all 2^m peer channels. In this case, A could maintain a small cache of advertisements, and if a new communication requires a channel that is not in the cache, A would have to wait until it hears another advertisement for that channel.

- There is a single public-key decryption for every connection initiation, or SYN packet, that member A receives . Further, A is required to encrypt every signal packet during communication. However, in general, A does not *have* to encrypt noise packets; it is only necessary that the adversary not be able to distinguish noise packets from signal packets. Thus, it is feasible for A to generate noise packets using a good local random number source.

- The broadcast nature of $\mathcal{P}^5$ requires individual channel members to (potentially) devote more bandwidth for communication than pure unicast (or systems such as Crowds [22]). However, the extra bandwidth directly results in enhanced anonymity, and, obviously, individual users may choose a more bandwidth efficient channel if they are willing to sacrifice some anonymity. We analyze the actual bandwidth usage and how it scales with number of channel members and different security parameters in Section 5.

3.6 $\mathcal{P}^5$ Nodes as IP Proxies

As described above, if two nodes want to communicate, they are required both to be members of the $\mathcal{P}^5$ system. However, orthogonal to the actual protocol, it is possible to communicate with parties outside of $\mathcal{P}^5$ if nodes implement IP proxies. Similar to existing anonymous communication systems[14, 22, 26], if node A wants to anonymously communicate with B that is not in the $\mathcal{P}^5$ system, A can contact an arbitrary node C anonymously through $\mathcal{P}^5$, and C can in turn contact B, acting as an IP layer proxy for A to B. Note that in doing so, B has no receiver anonymity because A must know B's IP address to initiate the session. Also, the sender-receiver anonymity of A to B is lost, because C knows that someone is talking to B. However, the sender anonymity of A is still preserved.

It is worth noting that any anonymous protocol that allows traffic to leave the system must be designed with care. Specifically, a naive implementation of IP proxies could lead to a trivial DoS attack. An attacker simply uses one or many proxies to flood a victim and remains anonymous in the process. Rate limiting and protocol verification are two techniques that could be used to mitigate this attack.

4 The Random Channels Model

In this section, we present an analysis of the paths in a $\mathcal{P}^5$ network. Let n be the number of users in the system, and suppose there are k channels available in total. For some integer t (which is typically small—at most 5), each user u independently chooses t random channels without replacement: we will denote this random set of t channels by $S(u)$. Two central parameters for us will be: (i) d, the maximum "hop-count" between any 2 users, which is the maximum communication distance between any two users, and (ii) L_{min} and L_{max}, the minimum and maximum load (number of users) on any channel. Note that L_{min} and L_{max} are different than the security parameters as they count only the set of users local to a channel, while the security parameters count the set of users in all channels that provide anonymity for a given user. We next discuss these two parameters.

The parameter d. We say that there is a path $u = u_0, u_1, u_2, \ldots, u_\ell = v$ between two users u and v, if for each i, users u_i and u_{i+1} choose a channel in common. The *length* of such a path is defined to be ℓ, and the minimum length of any path between u and v is the distance or hop-count between u and v; if there is no such path, then this hop-count is defined to be ∞. We are interested in bounding d, the maximum hop-count between any two users.

The load parameters L_{min} and L_{max}. Define the load on a channel to be the number of users who chose it among their t random channels; let L_{min} and L_{max} respectively be the minimum and maximum loads on any channel. A lower bound on the former is needed to guarantee the security of the system, and an upper bound on the latter is required to show that the bandwidth overhead is not significant.

Given the above discussion, our basic goals will be as follows. Clearly, we simultaneously want "small" d, a value of L_{min} that is "not too small", and L_{max} being "not too high". It is easy to see that the expected load on any given channel is exactly $t \cdot n/k$. Thus, n/k is a natural parameter to study our system with. So, we will consider scenarios where k is constrained to be at most some given value K, and n/k is required to be at least some given value λ. So, the primary parameters are K, t, and λ. Given these, and for any choice of (n, k) for which $k \leq K$ and $n/k \geq \lambda$, we aim to show that the system has the above-sketched satisfactory properties w.r.t. d, L_{min}, and L_{max}.

More concretely, we will proceed as follows. Fix $\epsilon = 0.3$, say. Let $\mathcal{A}_s$ denote the desirable event that "(i) all the channel loads are within $1 \pm \epsilon$ of the expected value $t\lambda$, and (ii) $d \leq s$". We derive the following sufficient conditions for $\mathcal{A}_s$ to hold (for $s = 2, 3$) with a probability of at least $1 - 10^{-4}$:

1. $s = 2$: two sufficient conditions are

   (P1) $t = 3$, $\lambda \geq 100$, and $K \leq 100$; or

   (P2) $t = 4$, $\lambda \geq 80$, and $K \leq 300$.

2. $s = 3$: two sufficient conditions are

   (P3) $t = 3$, $\lambda \geq 100$, and $K \leq 150$; or

   (P4) $t = 4$, $\lambda \geq 80$, and $K \leq 700$.

We now prove that these conditions are indeed sufficient, in the rest of this section.

4.1 Analysis Approach

In our analysis, we will frequently use the union bound or Boole's inequality: $\Pr[E_1 \vee E_2 \vee \ldots \vee E_s] \leq \sum_{i=1}^{s} \Pr[E_i]$.

The parameters L_{min} and L_{max} are much more tractable than d, so we handle them first. Let $\exp(x)$ denote e^x. It is an easy consequence of large-deviations bounds such as the Chernoff-Hoeffding bounds [8, 15] that for any given channel c and any parameter $\epsilon \in [0, 1]$, its load $L(c)$ satisfies:

$$\begin{aligned} & \Pr[L(c) \notin [(1-\epsilon) \cdot tn/k,\ (1+\epsilon) \cdot tn/k]] \\ \leq\ & \exp(-t\epsilon^2/2 \cdot (n/k)) + \\ & \exp(-t(\epsilon^2/2 - \epsilon^3/6) \cdot (n/k)) \\ \leq\ & \exp(-t\lambda\epsilon^2/2) + \exp(-t\lambda(\epsilon^2/2 - \epsilon^3/6)). \end{aligned}$$

Now, a simple application of the union bound yields

$$\begin{aligned} & \Pr[(L_{min} < (1-\epsilon) \cdot tn/k) \bigvee (L_{max} > (1+\epsilon) \cdot tn/k)] \\ \leq\ & K \cdot \exp(-t\lambda\frac{\epsilon^2}{2}) + K \cdot \exp(-t\lambda(\frac{\epsilon^2}{2} - \frac{\epsilon^3}{6}))] \end{aligned} \tag{1}$$

We next turn to bounding d. We cannot directly draw on the rich random graphs literature, since we are working here with a certain model of random hyper graphs with possibly repeated hyperedges. We next study the two requirements of most interest: $d \leq 2$ and $d \leq 3$. As can be expected, the second case involves more work than the first. Our basic plan is as follows. Lemma 4.1 gives an upper-bound on $\Pr[d > 2]$, and Lemma 4.2 gives an upper-bound on $\Pr[d > 3]$. Then, letting $\overline{\mathcal{E}}$ denote the complement of event $\mathcal{E}$, we see by the union bound that $\Pr[\overline{\mathcal{A}_2}]$ is upper-bounded by the sum of (1) and the probability bound given by Lemma 4.1. Similarly, $\Pr[\overline{\mathcal{A}_3}]$ is upper-bounded by the sum of (1) and the bound given by Lemma 4.2. We shall do this "putting together" in Section 4.4.

4.2 The Requirement $d \leq 2$

Here, we want sufficient conditions for "$d \leq 2$" to hold with high probability. In other words, we want to show that for any two users, there is a path of length at most 2 between them. To do so, we fix distinct users u and v, and upper-bound the probability p that there is no path of length 2 between them; then, by the union

bound, the probability of $d > 2$ is at most $\binom{n}{2} \cdot p$, since there are only $\binom{n}{2}$ choices for the unordered pair (u, v). Thus, we need to show that p is negligible in comparison with $(\binom{n}{2})^{-1}$, which we proceed to do now.

Our plan is to condition on the values of $S(u)$ and $S(v)$. For each such choice, we will upper-bound the probability that there is no user (among the remaining $(n-2)$) who chose a channel that intersects both $S(u)$ and $S(v)$. Then, the maximum such probability is an upper-bound on p. Fix $S(u)$ and $S(v)$. If $S(u) \cap S(v) \neq \emptyset$, then u and v are at distance 1; so suppose $S(u) \cap S(v) = \emptyset$. In particular, we may assume that $k \geq 2t$.

Consider any other user w. What is the probability of w's random choice $S(w)$ intersecting $S(u)$ and $S(v)$? The total number of possible choices for w is $\binom{k}{t}$. The number of possible intersection patterns can be counted as follows. Suppose $|S(w) \cap S(u)| = i$ and $|S(w) \cap S(v)| = j$, where $i, j \geq 1$ and $i + j \leq t$. The remaining $t - (i + j)$ elements of $S(w)$ are selected at random from outside $S(u) \cup S(v)$. Thus,

$$\Pr[(S(w) \cap S(u) = \emptyset) \bigvee (S(w) \cap S(v) = \emptyset)] = 1 - f(k, t),$$

where

$$f(k, t) = \frac{\sum_{i,j \geq 1;\ i+j \leq t} \binom{t}{i}\binom{t}{j}\binom{k-2t}{t-i-j}}{\binom{k}{t}}.$$

Thus, since different users w make their random choices independently, we get that $p \leq (1 - f(k, t))^{n-2} \leq \exp(-(n-2)f(k, t))$. Thus, as discussed above, a union bound yields

$$\Pr[d > 2] \leq (n^2/2) \cdot \exp(-(n-2)f(k, t)). \tag{2}$$

In order to see what this bound says for various concrete values of our parameters K, λ, t, we develop:

Lemma 4.1 *Suppose $\lambda \geq 2K/(t^3(t-1))$ and $K \geq 2^{t+2}$. Then,* $\Pr[d > 2] \leq ((Ke\lambda)^2/2) \cdot \exp(-0.8t^3(t-1)\lambda/K)$.

Proof. (Sketch.) Since $f(k, t) \geq t^2\binom{k-2t}{t-2}/\binom{k}{t}$, bound (2) shows that

$$\Pr[d > 2] \leq (n^2e^2/2) \cdot \exp\left(-nt^2\binom{k-2t}{t-2}/\binom{k}{t}\right).$$

We can now do a calculation to show that subject to our constraints (which includes the constraint that $k \geq 2t$), this bound is maximized when $k = K$ and $n = \lambda K$. Further simplification then leads to the bound of the lemma.

□

4.3 The Requirement $d \leq 3$

We now adopt a different approach to get an upper-bound on $\Pr[d > 4]$. Let $\mathcal{C}$ denote our set of k channels. For a set $A \subseteq \mathcal{C}$ with $|A| = t$, define $Y(A)$ to be the set of all $a \in (\mathcal{C} - A)$ for which the following holds:

$$\exists x :\ (a \in S(x)) \bigwedge (S(x) \cap A \neq \emptyset).$$

In other words, $Y(A)$ is the set of channels that lie outside of A, but which lie in some set $S(x)$ that intersects A. Thus, we need to show that with high probability, at least one of the following four conditions holds for each pair of users u and v: (i) $S(u) \cap S(v) \neq \emptyset$; (ii) $S(u) \cap Y(S(v)) \neq \emptyset$; (iii) $Y(S(u)) \cap S(v) \neq \emptyset$; or (iv)

$Y(S(u)) \cap Y(S(v)) \neq \emptyset$. To do so, we will instead show that with high probability, *all* $A \subseteq \mathcal{C}$ with $|A| = t$ will have $|Y(A)| > s$, where $s = \lfloor (k-2t)/2 \rfloor$. It can be verified that this implies that at least one of the conditions (i), (ii), (iii) and (iv) will hold for all u, v.

Fix $A \subseteq \mathcal{C}$ such that $|A| = t$. Let us bound $\Pr[|Y(A)| \leq s]$. Now, for any fixed $B \subseteq (\mathcal{C} - A)$ such that $|B| = k - t - s = \lceil k/2 \rceil$, a calculation can be used to show that

$$\Pr[Y(A) \cap B = \emptyset] \leq (\exp(-t^2/k) + 2^{-t})^{n-1}. \tag{3}$$

We summarize with

Lemma 4.2 *Suppose* $\lambda \geq 2K/(t^3(t-1))$ *and* $K \geq 2^{t+2}$. *Then,* $\Pr[d > 3] \leq 2^{K+2} \cdot K^t \cdot \exp(-t^2\lambda/2)$.

Proof. (Sketch.) A union bound using (3) yields

$$\begin{aligned} \Pr[d > 3] &\leq \Pr[\exists A : |Y(A)| \leq s] \\ &\leq \binom{k}{t} \cdot \binom{k-t}{\lceil k/2 \rceil} \cdot (\exp(-t^2/k) + 2^{-t})^{n-1} \\ &\leq 2^k \cdot k^t \cdot (\exp(-t^2/k) + 2^{-t})^{n-1}. \end{aligned}$$

A calculation shows that subject to our constraints, this bound is maximized when $k = K$ and $n = \lambda K$. Further simplification completes the proof. □

4.4 Putting It Together

Now, as described at the end of Section 4.1, we just do routine calculations to verify the following. First, if $s = 2$ and any one of (P1) and (P2) holds, then the sum of (1) and the probability bound given by Lemma 4.1 is at most 10^{-4}. Similarly, if $s = 3$ and any one of (P3) and (P4) holds, then the sum of (1) and the probability bound given by Lemma 4.2 is at most 10^{-4}. This concludes our proof sketch about these sufficient conditions for $\mathcal{A}_2$ and $\mathcal{A}_3$ respectively to hold with high probability.

5 Simulation Results

In this section, we present results from a packet-level $\mathcal{P}^5$ simulator. Our simulator is written in C, and can simulate the entire $\mathcal{P}^5$ protocol with thousands of participants. We designed and implemented five basic experiments:

- Measure system performance as the number of participants increase; specifically, we measure the end-to-end bandwidth, latency, and packet drop rates as the number of users in the system is increased
- Measure the effect of the security and efficiency parameter on communication efficiency
- Estimate the amount of time it takes $\mathcal{P}^5$ systems of a given number of participants to converge (i.e. how long does it take a user to find a channel that satisfies their security and efficiency constraints)
- Measure the effects of different noise generation rates and queuing disciplines
- Measure how the system behaves when increasing numbers of nodes engage in end-to-end communication

Simulation Methodology For each experiment, we generated a random physical topology. We did not model different propagation delays between pairs of nodes; instead, we assumed unit propagation delay between any two nodes. Since all inter-node latencies are the same, our simulation proceeds using a synchronous clock. At every tick, all packets sent from every node is received at their destination.

We assume an unbounded input queue length and a bounded output queue. All packets received at a node at a given time step are processed. Some of these packets may be queued at appropriate output queues, and a subset of them may be delivered locally. Next, each node generates a set of outgoing packets and enqueues these on the output queues. We then impose the output queue limit and according to the queuing discipline, discard packets if any output queue is larger than its maximum specified size. Note that during the discard phase, the node does not discriminate whether it is dropping its own packets or packets from some other node. All remaining packets at an output queue are delivered in the next time step to the next hop node.

Since all packets queued at a node are delivered at the next time step, the output queue size serves as a measure of both the processing and bandwidth requirements at a node. We instrumented the simulator to record the end-to-end latencies (number of simulator ticks), drop rates and bandwidth, end-to-end hop counts, number of channel crossings, and convergence times. In the rest of this section, we report results from individual experiments.

5.1 Scalability

In Figure 13, we plot the end-to-end loss rate as the number of users in the system is increased. In each case, there is only one pair of communicating nodes; all other nodes only send noise packets. For each group size, we chose ten different random seeds and created ten different topologies. For each topology, we chose three different sender-receiver pairs. Each point on Figure 13 is an average of all these runs (24 for each topology size) with the error bars denoting 95% confidence intervals. Unless otherwise noted, we choose the values of security parameters as $\psi = 100$, and $\eta = 300$, and implement non-uniform queuing. All nodes in the system were connected to two channels, i.e. they had one communication key and one routing key.

There are two different curves, each corresponding to two different sending rates. In the "Sending Rate=1/S" case, each node generates a packet at each time stop with probability $1/S$, where S is the size of its current broadcast group. Analogously, in the "Sending Rate=$\log S/S$" case, each node generates a packet at each time stop with probability $\log S/S$. For both cases, the queue sizes at each node were very small: $\max\{10, \log S\}$. From the plot, it is clear that the $1/S$ sending rate can be sustained in the system, and almost no signal (or noise) packets are lost. However, as the sending rate is increased, the queues in the system are saturated, and drop rates increase with group size.

In Figure 14, we plot the average number of channels that the signal packets have to cross in order to reach their destination. Note that in this case, each user only connects to 2 channels. As predicted by the analysis in Section 4, the average channel level hop count is very small, and is less than 1 for all our runs. The average end-to-end hop count in these runs were ~ 13. The worst case inter-channel distance that we encountered in these runs was 2. This occurred in 4 out of 6000 signal packets sent.

Analysis In our simulations, the average size of each channel in these experiments is approximately 150, and the average queue size is around the minimum (10). In the worst case, each queue is always full, and the nodes have to handle two full queues worth (20 packets) per tick. Each tick in our system corresponds to an end-to-end propagation delay. Assume that this delay is on average on the order of 100ms. Thus, these nodes would have to handle up to 200 packets per second. At 1000 bytes per second, this translates to 1.6

Mbps of bandwidth and the ability to handle two hundred symmetric key decryptions per second, assuming that most packets are data and not SYN packets, is trivial with modern computers. The bandwidth required is somewhat more of a concern; however, note that individual users can always reduce their bandwidth requirement by migrating "lower" in the tree. The "Sending Rate=1" corresponds roughly to each node sending at 16Kbps (with essentially no packet loss). If nodes are willing to incur higher packet loss, then the sending rate can be much higher, e.g. for the 8192 user case, users can send at up to 200 Kbps if they are willing to handle up to 40% packet losses. Note that these losses accumulate over about 13 hops, which is the average end-to-end path length in these simulations.

Thus, in a 8192 node $\mathcal{P}^5$ network, any subset users can *anonymously* communicate at hundreds of kilobits per second (with relatively high packet losses), if they invest approximately 2 Mbps of bandwidth. Clearly, the loss rate is high compared to the communication media we are used to, but we have to remember that in $\mathcal{P}^5$, the user is gaining anonymity of at least 100 other users. In a pure broadcast system with 8192 users and these same parameters, the average end-to-end loss rate would be about $1-10^{-39}$; communication in this system would, essentially, be impossible. (In such a system, to achieve a 50% average end-to-end loss rate, the sending rate per user would have to be $\sim$ 1 packet/8 seconds. Of course, each user is also gaining anonymity from *all* other users in the system).

Additionally, modern forward error correction techniques can be used to mitigate the effects of the high loss rate. With packet loss rate of δ and a bandwidth of B, then the effective rate of the channel is $B*(1-\delta)$ with Reed-Solomon[21] codes or $B*(1-\delta(1+\epsilon))$ with tornado[19] codes. Recall that Reed-Solomon encoding requires O($n \cdot log_n$) computation where n is the number of blocks in an encoding, where as tornado codes require O(n) computation. There is even work[20] that works with variable loss rates.

In Section 5.3, we discuss the effects of varying the sending rate while the bandwidth and group sizes are fixed.

5.2 Convergence Times

In Table 2, we present the convergence times for $\mathcal{P}^5$ in our simulator. There are results from two different experiments in the Table; in the experiment we fixed the security parameters (at $\psi = 100,\ \eta = 300$), and varied the number of users. In the second experiment, we fixed the number of users at 1024, and varied the security parameter (ψ). In all cases, we used $\eta = 3\psi$.

In all experiments, all the users join simultaneously at time 0. Each user migrates down the $\mathcal{L}$ tree in rounds. Each round consists of 10 simulator ticks, and each user only makes a single migration decision

Channel Size	No. of rounds	Security Parameter	No. of rounds
64	0	16	6
128	0	32	5
256	1	64	4
512	2	128	3
1024	3	256	2
2048	4	512	1
4096	5		
8192	6		

Table 2: Convergence times

in any one round. As expected, the convergence times increase as the number of users increase (or the ψ parameter is decreased) since each user *settles* "lower" down in $\mathcal{L}$. However, in all cases the number of rounds to converge is given by $\max\{0,\ \log N - \log \psi\}$.

5.3 Noise and Signal Generation

In Figure 15, we vary the sending rate while keeping all other parameter fixed. We monitor a single sender–receiver pair, and report the observed packet loss. We use the two base sending rates from Section 5.1, and linearly increase these rates by the rate multipliers plotted on the x-axis, while keeping the link bandwidths constant. As expected, the drop rates increase linearly with increases in sending rate. Interestingly, the non-uniform drop rates perform slightly better as the sending rates increase.

In Figure 16, we repeat the same experiment and vary the number of sender-receiver pairs in the system. The rest of the users still generate noise at the same rate ($\log S/S$). We plot the average drop rate across all of the sender-receiver pairs, and as above, the error bars denote 95% confidence intervals. As expected, the drop rate is not affected by the number of sender–receiver pairs, and thus, no extra information is divulged to a passive observer when more people in the system communicate. We have also experimented with different values of ψ and η. As expected, the drop rate increases as users choose higher values of the security parameters, since they are mapped to larger broadcast channels.

6 Related Work

We discuss prior work related to $\mathcal{P}^5$. Previous work can be broken down into work relating to the Dining Cryptographers problem, Mixes, and other systems that provide anonymity over the Internet. We then summarize existing work in Table 3.

6.1 Dining Cryptographers

The Dining Cryptographers (DC-net) protocol [7] provides sender anonymity under an adversary model similar to $\mathcal{P}^5$. DC-Net assumes a public key infrastructure, and users send encrypted broadcasts to the entire group, thus achieving receiver anonymity. However, unlike $\mathcal{P}^5$, all members of the group are made aware of *when* a message is sent, so Dining Cryptographers does not have the same level of sender-receiver anonymity. Also, in DC-net, only one user can send at a time, so it takes additional bandwidth to handle reservations, collisions, and contention [27]. Lastly, a DC-net participant fixes its anonymity versus bandwidth trade off when joining the system, and there are no provisions to rescale that trade off when others join the system.

6.2 Mixes

A *mix* is a process that provides anonymity via packet re-shuffling. Encrypted messages are sent to a machine, the mix, are batched, then decrypted and resent to their destinations in concert, thereby obscuring the originator of the message. Mixes were introduced by Chaum in [6], but were improved upon in [5, 16]. Mixes work best in series, and need a constant amount of traffic to avoid delay while preserving anonymity. $\mathcal{P}^5$ does both by creating a hierarchy of mixes, and the constant stream of signal and noise packets serve to keep the mixes operational.

Of particular note is Mixminion[10] in which a user chooses a route through a set of mixes to a destination. Receiver anonymity is provided by registering a pseudonym at a *nymserver*, and publishing the

location of the nymserver. Messages are queued at the nymserver until such time as the recipient makes an anonymous connection to the nymserver to retrieve the messages. Further, Mixminion uses key rotation, 2 legged paths, and timed dynamic-pool batching to mitigate replay attacks, tagging attacks, and blending attacks respectively. They use dummy traffic between mixes for further mitigate the blending attack, but do not use end to end dummy traffic and cede that Mixminion is vulnerable to a long term correlation attack.

In contrast to Mixminion, $\mathcal{P}^5$ uses the same adversarial model, but is *not* vulnerable to the long term correlation attack. By broadcasting dummy/noise traffic end to end at a fixed rate, all nodes send and receive a constant rate of traffic, defeating the correlation. Additionally, by using broadcast channels, $\mathcal{P}^5$is not vulnerable to replay, tagging, or blending attacks. Of course, $\mathcal{P}^5$'s broadcast channels are less efficient than Mixminion's multi-hop unicast traffic, but users of $\mathcal{P}^5$ can locally configure the level of inefficiency. Also, Mixminion targets high latency tolerant applications such as e-mail, where $\mathcal{P}^5$ is a general purpose anonymous network layer suitable for all applications.

6.3 Recent Internet-based Anonymous Communications Work

We describe four anonymity protocols that can be implemented over the Internet.

Xor-Trees Like $\mathcal{P}^5$, Xor-Trees [11] provides sender-, receiver-, and sender-receiver anonymity. However, unlike $\mathcal{P}^5$, Xor-Trees do not admit a per user anonymity versus communications efficiency trade off. Also, like in DC-net, only a single user may send at any one time in an Xor-Tree. Thus, in an Xor-Tree, performance degrades due to collisions as the number of users increase.

Sender Anonymity Protocols Both Crowds [22] and the more recent Hordes [26] provide strictly sender anonymity. The basic idea in both these systems is similar to Onion Routing [14] and Tarzan [12], in which messages between communicating users are routed on an application-layer overlay using paths that are either randomly or sender chosen to obscure the origin. The receiver cannot resolve the sender of a particular message since messages take different, potentially randomly chosen, routes through the network. However, none of these systems provide anonymity when confronted by a passive observer who can mount statistical attacks by tracing and correlating packets throughout the network over time. Additionally, none of these systems provide receiver anonymity. However, any of these protocols can be adapted to provide receiver anonymity by using an anonymous proxy [23]. Aside from the scalability issues of having all of the users send their communications through a single server, users of an anonymous proxy system explicitly give up sender-receiver anonymity to the connecting proxy.

FreeNet FreeNet [9] provides an anonymous publish-subscribe system over the Internet using an application-layer overlay, much like $\mathcal{P}^5$. However, FreeNet is designed for anonymous storage and retrieval, and the anonymity issues for such a system are different than a system like $\mathcal{P}^5$ that provides anonymity when communicating parties are on-line. There is no notion of noise or signal, etc., and the major issues in FreeNet are decoupling/hiding authorship from a particular document, and providing fault-tolerant anonymous availability for a set of static documents.

6.4 Summary

The capabilities of the above systems are summarized in Table 3. The columns "Sender", "Receiver", and "Sender-Receiver" refer to the systems ability to provide sender, receiver, and sender-receiver anonymity,

as described above. Under receiver anonymity, "In System" denotes that the receiver can have anonymity if they are a participant in the system, as opposed to an external receiver connected via an IP proxy. Note that any system that does not provide receiver anonymity can be modified to do so via an anonymous proxy. Further note that all protocols that are incapable of receiver anonymity are still able to achieve two way communication. The "Scalable" metric refers generically to whether the system can reasonably scale to thousands of nodes, and still make progress. The "correlation" column refers to whether the system is vulnerable to a long term traffic correlation attack. The last column, "Locally Configurable" refers to the ability of an individual node to locally configure their anonymity levels, independent of other nodes.

Viewed broadly, previous systems forced users to choose between security, i.e., all three types of anonymity with resistance to long term correlation attacks, or efficiency and scalability. To the best of our knowledge, $\mathcal{P}^5$ is the only system that locally lets users trade off between these two points.

7 Conclusions

$\mathcal{P}^5$ is a protocol for anonymous communication over the Internet. It allows secure anonymous connections between a hierarchy of progressively smaller broadcast channels, and allows individual users to trade off anonymity for communication efficiency.

In developing $\mathcal{P}^5$, we found an interesting property relating communication latency, bandwidth usage, and anonymity. In general, we found it was easy to construct protocols that provided two out of these three properties, e.g., consider plain unicast communication: it provides low latency and high bandwidth usage, but does not provide anonymity. Now consider multi-casting to a set (using per-source shortest path trees) in which the message is intended for only one member of the channel. This solution provides low latency; however the bandwidth utility decreases as the anonymity and unlinkability increases. $\mathcal{P}^5$ has the interesting property that it allows individual users to trade-off these three properties on-line.

We designed $\mathcal{P}^5$ to be scalable and compatible with current Internet protocols. Our simulations show that $\mathcal{P}^5$ can scale to large groups, and our analysis shows that $\mathcal{P}^5$ will maintain its "short paths" property with very little extra overhead for extremely large groups. Our current work is to adapt lower overhead noise generation algorithms to further improve scalability; provide better reliability by considering more connected structures within individual groups; and to build a prototype for deployment around the Internet.

Acknowledgments. We thank the anonymous referees of the IEEE Security and Privacy 2002 program committee and the Journal of Computer Security editorial committees for their helpful comments.

Protocol	Sender	Receiver	Sender-Receiver	Scalable	Correlation	Locally Configurable
Xor Trees	Yes	In System	Yes	No	No	No
DC Net	Yes	In System	Yes	No	No	No
Crowds	Yes	No	No	Yes	Yes	Yes
Onion Routing	Yes	No	No	Yes	Yes	Yes
Tarzan	Yes	No	No	Yes	Yes	Yes
Mixminion	Yes	In System	Yes	Yes	Yes	Yes
FreeNet	Yes	No	No	Yes	Yes	Yes
$\mathcal{P}^5$	Yes	In System	Yes	Yes	No	Yes

Table 3: Summary of Capabilities of Various Anonymity Protocols

References

[1] http://www.epic.org/privacy/carnivore.

[2] NTRU Performance Numbers. http://www.ntru.com/cryptolab/faqs.htm.

[3] Pen Registers and Trap and Trace Devices. http://www.usdoj.gov/criminal/cybercrime/pentrap3121_3127.htm.

[4] www.echelonwatch.org.

[5] O. Berthold, A. Pfitzmann, and R. Standke. The disadvantage of Free MIX Routes and How to Overcome Them. pages 30–45. Designing Privacy Enchancing Technologies: Workshop on Design Issue in Anonymity and Unobservability, 2000.

[6] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2), 1981.

[7] D. Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[8] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Annals of Mathematical Statistics, 1952.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009*, 2001.

[10] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a Type III Anonymous Remailer Protocol. IEEE - Security and Privacy Symposium, 2003.

[11] S. Dolev and R. Ostrovsky. *Xor-Trees for Efficient Anonymous Multicast Receiption*. Advances in Cryptography - CRYPTO 97, 1997.

[12] M. J. Freedman, E. Sit, J. Cates, and R. Morris. Introducing Tarzan, A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of 1st Intl. Workshop on Peer-to-Peer Systems*, March 2003.

[13] V. Fuller, T. Li, J. Yu, and K. Varadhan. Rfc 1519 - classless inter-domain routing (cidr): an address assignment and aggregation strategy, September 1993.

[14] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2), February 1999.

[15] W. Hoeffding. Probability inequalities for sums of bounded random variables. American Statistical Association Journal, 58, 1963.

[16] A. Jerichow, J. Müller, A. Pfitzmann, B. Pfitzmann, and M. Waidner. Real-Time MIXes: A Bandwidth-Efficient Anonymity Protocol. *IEEE Journal on Selected Areas of Communication*, 1998.

[17] P. Karger. Non-discretionary access control for decentralized computing. Master's thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1977.

[18] G. Lindsay. The government is reading your e-mail. TIME DIGITAL DAILY, June 1999.

[19] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47:569–584, February 2001.

[20] P. Maymounkov. Online Codes. Technical report, New York University, 2002.

[21] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math*, 8:300–304, 1960.

[22] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[23] V. Scarlata, B. Levine, and C. Shields. Responder Anonymity and Anonymous Peer-to-Peer File Sharing. In *Proceedings of IEEE International Conference on Network Protocols (ICNP) 2001.*, 2001.

[24] B. Schneier. http://www.schneier.com/crypto-gram-9912.html.

[25] R. Sherwood, B. Bhattacharjee, and A. Srinivasan. P5: A Protocol for Anonymous Communications. In *In the Proceedings of IEEE Security and Privacy Symposium*, 2002.

[26] C. Shields and B. N. Levine. A Protocol for Anonymous Communication Over the Internet. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (CCS-00)*, pages 33–42, N.Y., Nov. 1–4 2000. ACM Press.

[27] M. Waidner and B. Pfitzmann. The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology—EUROCRYPT 89*, volume 434 of *Lecture Notes in Computer Science*, page 690, April 1989.

[28] M. Wright, M. Adler, B. Levine, and C. Shields. An analysis of the degradation of anonymous protocols. 2002.

A Updates to Security and Privacy 2002 Paper

There are significant updates to the $\mathcal{P}^5$ protocol from the original conference paper[25]. One of the most common critiques of the original protocol was the need for one public key decryption per packet. The solution, as described in Section 2.4, is to use the public key cryptography to negotiate a symmetric session key. This technique can be used to arbitrarily reduce the CPU load on a node, at the cost of increased connection initiation latency.

Another problem with the original protocol was its failure to account for the fact that new nodes in the system do not increase the security of previously communicating nodes. For example, Alice joins the system at a certain channel in $\mathcal{L}$ that meets her security and efficiency requirements, and she starts communicating with Bob. Then, over time, enough new nodes join the same channel, and this causes the channel to exceed Alice's efficiency parameter. Alice decides to go down in the tree (Section 3.4) to preserve her efficiency. However, if we assume that Bob is colluding with the global passive attacker (as per our adversarial model), then as soon as Alice starts communicating, Bob tells the passive adversary which channel she is in, and then the passive adversary can enumerate all of the nodes of Alice's old channel. From there, the adversary knows that any new nodes that join the same channel are *not* Alice. But, when Alice goes down one level in the logical tree to a new, smaller channel, her anonymity is not the size of the new channel, but the size of

the set formed by the intersection of the new channel with the original members of the old channel. In other words, Alice is vulnerable to an intersection attack. Note that the original protocol had multiple users in a single node of $\mathcal{L}$, where the revised protocol has exactly one use per node in $\mathcal{L}$.

The updated protocol fixes this by slightly modifying the join procedure, as shown in Section 3.3 and Figures 6-8. By making each node of $\mathcal{L}$ correspond to exactly one user, and by making the users fill in the nodes of $\mathcal{L}$ as they join, Alice knows that the people above her in the tree are people who joined before her, and people below her are people who have joined after her. Also, since the hashes of public keys in the system are randomly distributed, and the exact leaf joined in the tree is randomly chosen (Figure 8), then we can assume that new leaves to $\mathcal{L}$ are on average evenly distributed. Because the $\mathcal{L}$ is a binary tree, there are 2^m nodes at level m. Thus, if Alice joins the system, and advertises channel (b/m), a new node ends up in the same channel with probability 2^{-m}.

Thus, the intersection attack is mitigated two ways. First, by understanding that nodes in $\mathcal{L}$ are filled in sequentially as they join, Alice can decide whether going down in $\mathcal{L}$ will violate her security parameter, or if she needs to rejoin with a new key. This information was not available in the original conference version of the protocol[25]. Second, by evenly distributing where in $\mathcal{L}$ new nodes join the system, Alice will not need to go down in her channel unless there is a significant increase in the system size. For example, assume Alice has parameters $\psi = 100$, $\eta = 300$, and $H(PK_{Alice}) = 010101\ldots$, and joins when there are 8000 nodes in the system. Channel CH($*$ / 0) will have all 8000 nodes, and on average, channels CH(0 / 1) will have roughly 4000 nodes, CH(01 / 2) will have 2000 nodes, and so on. So, statistically, a channel on the 6th level will have approximately 125 nodes, so Alice chooses channel CH(010101 / 6) to advertise because it fits her ψ and η parameters. A new node ends up joining as a leaf in channel CH(010101 / 6) with probability $2^{-6} = 1/64$. Further, if 175 more nodes join Alice's channel, her efficiency parameter will be violated (125 nodes+ 175 nodes = 300 nodes), and she will have to go down in the $\mathcal{L}$. So, on average $64 * 175$ or 11200 new nodes would have to join the system before Alice has to go down in the $\mathcal{L}$.

The last major criticism of $\mathcal{P}^5$ was the restriction that once a user starts communicating in the $\mathcal{P}^5$ system, he cannot leave the system. In the updated version of the protocol, we relax this restriction, as described in Section 3.4.

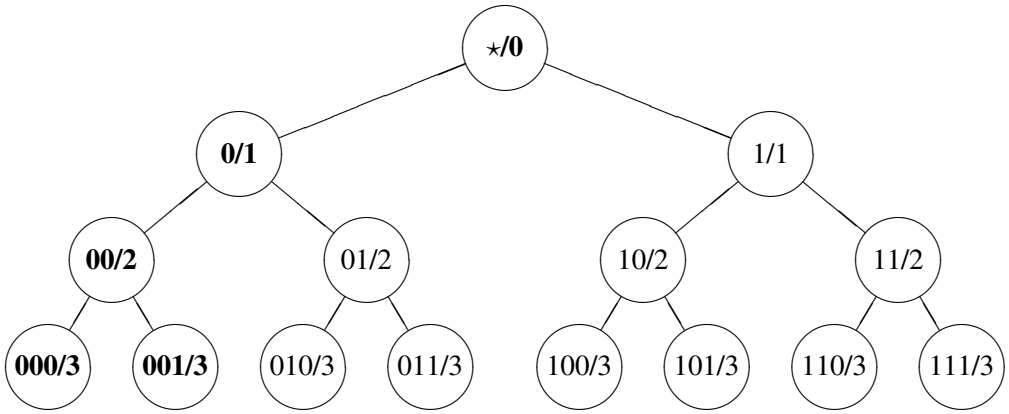

Figure 1: Form of the $\mathcal{P}^5$ logical broadcast tree ($\mathcal{L}$). The effective broadcast channel of a user in node $(00/2)$ is shown in boldface.

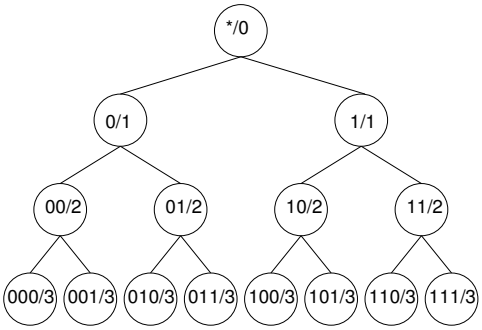

Figure 2: Channel CH($\star$ / 0)

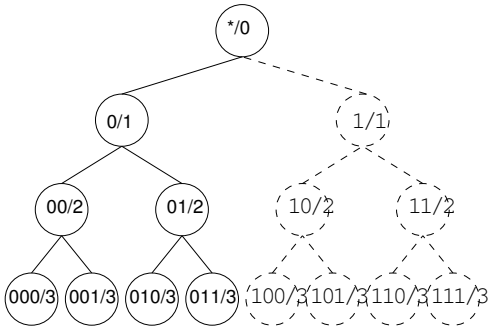

Figure 3: Channel CH(0 / 1)

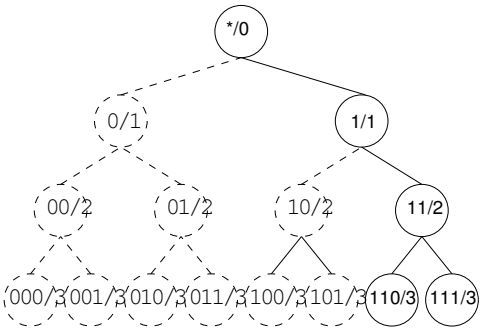

Figure 4: Channel CH(11 / 2)

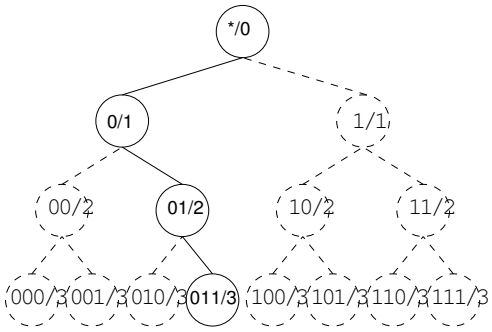

Figure 5: Channel CH(011 / 3)

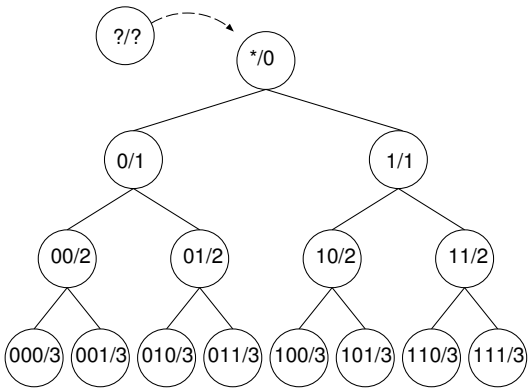

Figure 6: Node joins with $\eta = 14, \psi = 8$ and $H(Pk) = 10$....

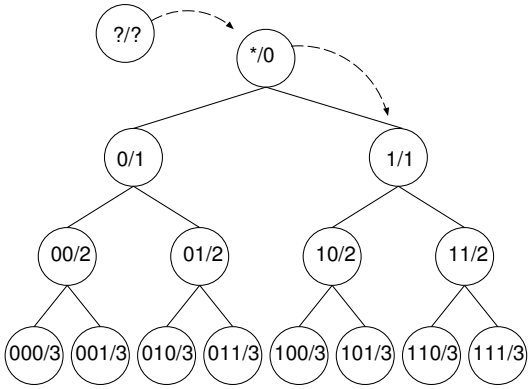

Figure 7: Node binary searches to find $(1/1)$, $|E\{(1/1)\}| = 8$

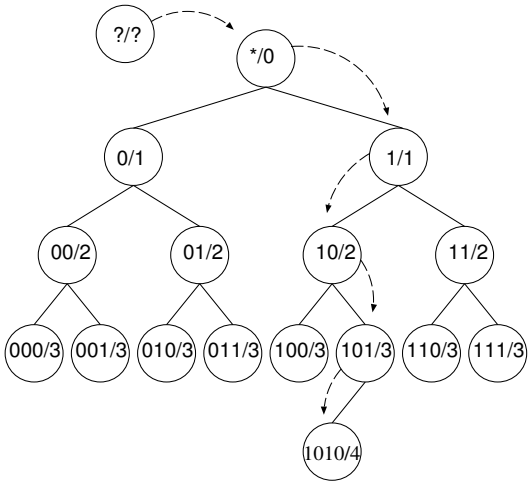

Figure 8: Node then randomly searches down the tree, and joins as a leaf

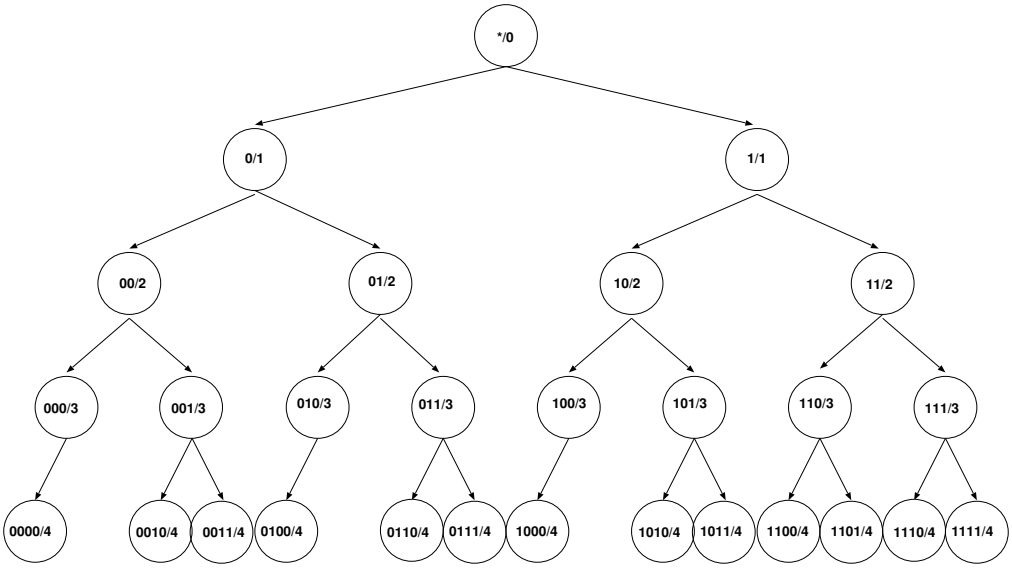

Figure 9: $\mathcal{P}^5$ before Alice joins

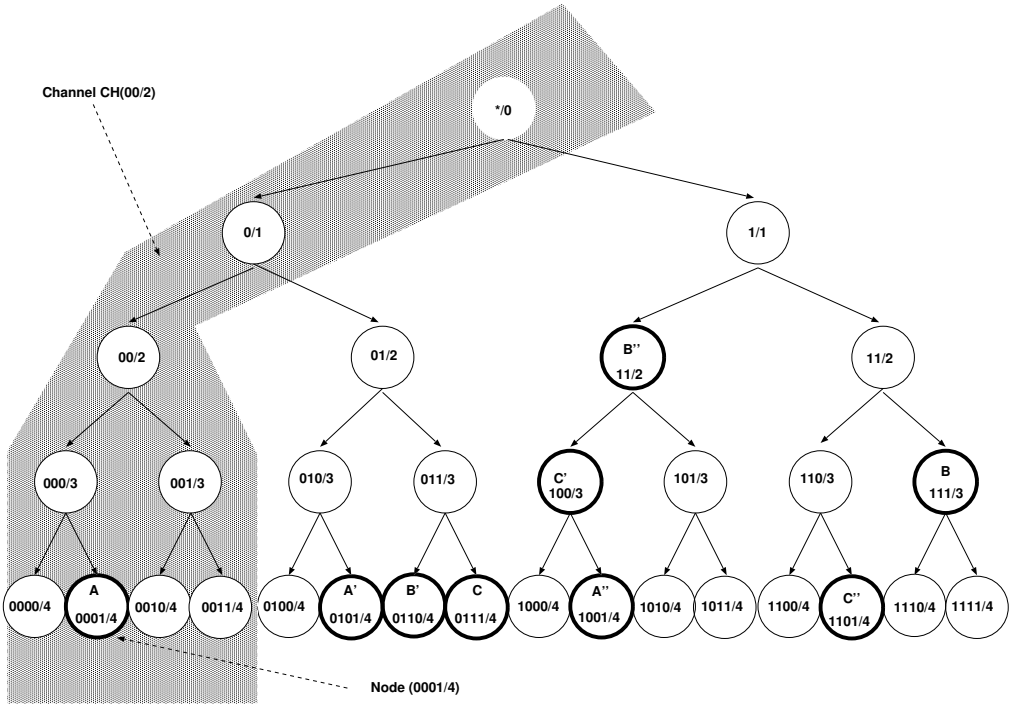

Figure 10: $\mathcal{P}^5$ after Alice joins

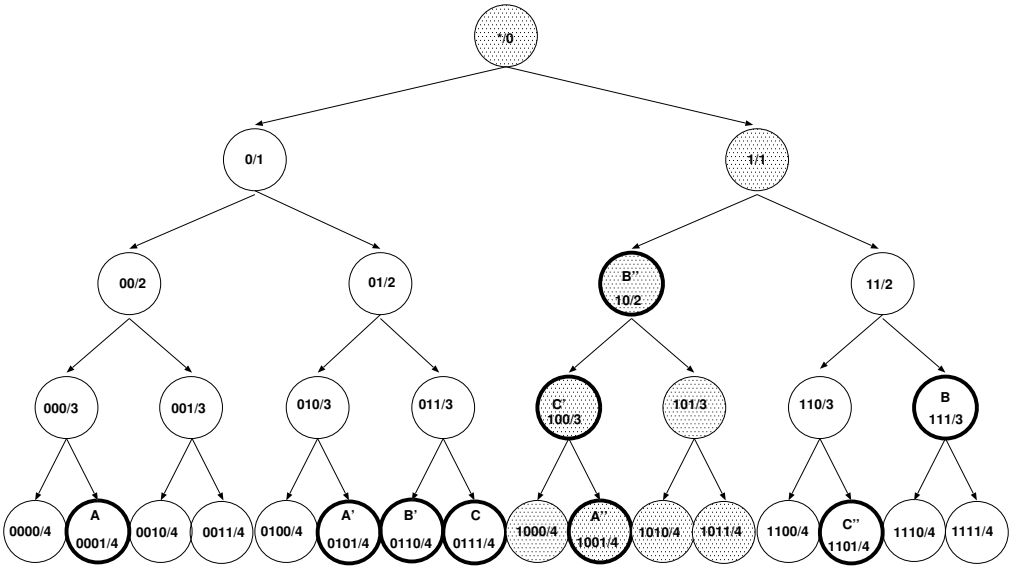

Figure 11: First hop of Alice → Bob message

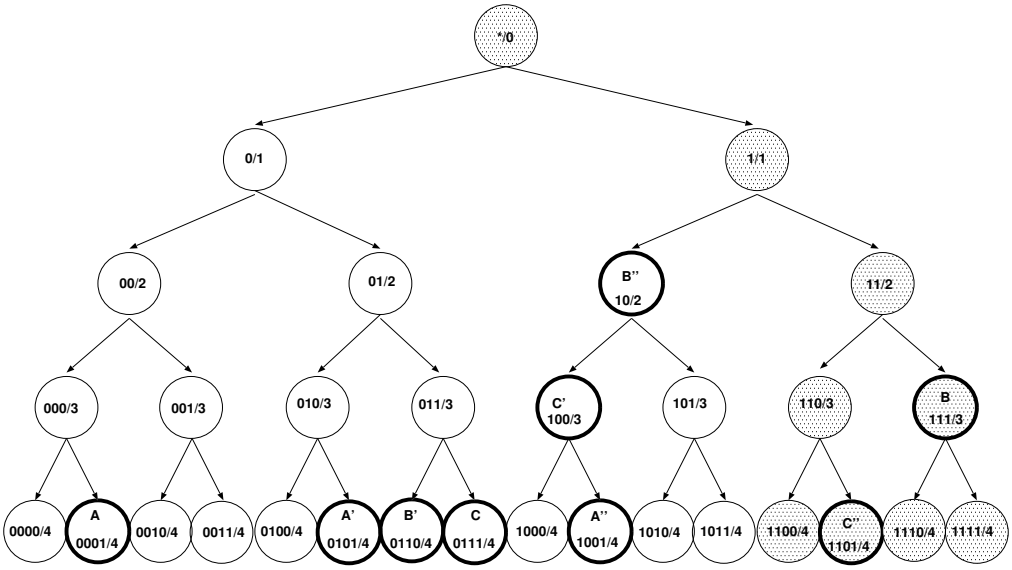

Figure 12: Second/Final hop of Alice → Bob message

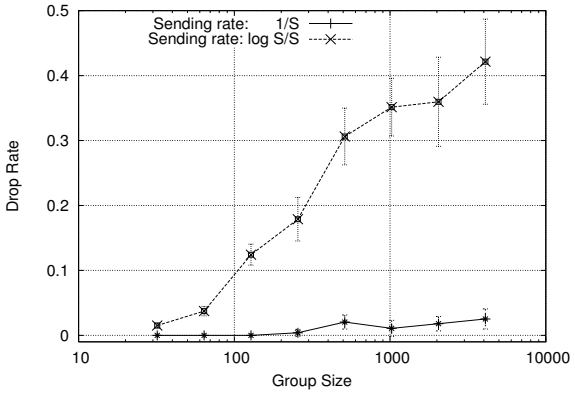

Figure 13: Loss rate versus number of users

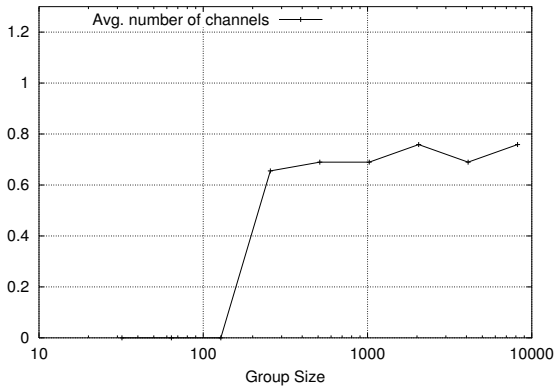

Figure 14: Average number of channels

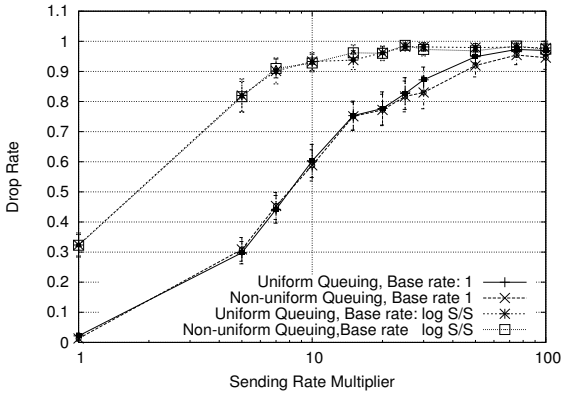

Figure 15: Loss rate versus sending rate

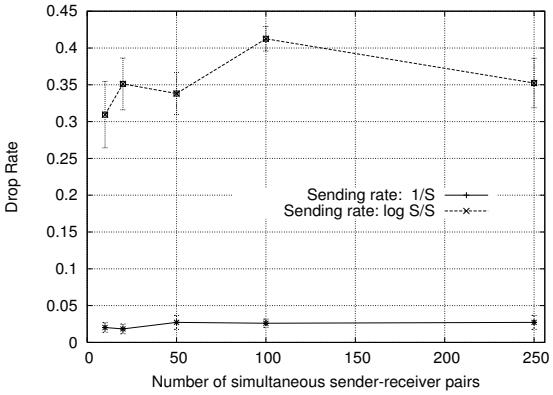

Figure 16: Loss rate versus number of senders