Journal of Combinatorial Optimization manuscript No.
(will be inserted by the editor)

Combinatorics of TCP Reordering

Anders Hansson[1], **Gabriel Istrate**[⋆⋆2], **Shiva Prasad Kasiviswanathan**[3]

[1] CCS-5, Discrete Simulation Science, P.O. Box 1663, Mail Stop M997, Los Alamos National Laboratory, Los Alamos, NM 87545, e-mail: hansson@lanl.gov
[2] CCS-5, Discrete Simulation Science, P.O. Box 1663, Mail Stop M997, Los Alamos National Laboratory, Los Alamos, NM 87545, e-mail: gabrielistrate@acm.org
[3] Department of Computer Science and Engineering, Pennsylvania State University, 346D IST Building, University Park, PA 16802, e-mail: kasivisw@cse.psu.edu

The date of receipt and acceptance will be inserted by the editor

Abstract We study a combinatorial problem motivated by a receiver-oriented model of TCP traffic from [7], that incorporates information on both arrival times, and the dynamics of packet IDs. An important component of this model is a many-to-one mapping FB from sequences of IDs into a sequence of *buffer sizes*. We show that: i) Given a buffer sequence B, constructing a sequence A of IDs that belongs to the preimage of B is no harder than finding matchings in bipartite graph. ii) Counting the number of sequences A of packet IDs that belong to the preimage of B can be done in linear time in the special case when there exists a constant upper bound on the maximum entry in B. iii) This problem also has a fully polynomial randomized approximation scheme when we have a constant upper bound on the number of repeats in the packet sequences in the preimage. We also provide experimental evidence that the two previous results suffice to efficiently count the number of preimages for buffer sequences observed in real TCP data.

1 Introduction

Consider a sequence of TCP *packets*, identified by an integer IDs, arriving at a *receiver* in the network. The receiver delivers the packets to an *application layer*, respecting *packet sequence integrity*. In other words, at every moment the sequence of packet ID forwarded to the application layer must form a contiguous sequence. Network conditions can make packets arrive out-of-order, therefore out-of-order packets need to be buffered. Several copies of the same packet can arrive, but only one copy of each packet is useful (and will be stored, if needed). We assume that the receiver evicts a given packet from the buffer and passes it to the application

⋆⋆ Corresponding Author

as soon as possible, i.e., as soon as this respects the packet sequence integrity constraint.

For a given sequence of packet IDs $A = (A_1, \ldots, A_n)$, define $FB(A)$ to be the sequence $(FB_{A,1}, \ldots FB_{A,n})$ of sizes of the smallest buffer needed to store all out-of-order packets. We will assume that the we reserve a slot in the buffer for any packet with ID smaller than the largest ID received so far that has not been already uploaded, *including packets not yet received*. For example, sequence of packets $4\ 2\ 3\ 2\ 1$ is mapped onto the sequence of buffer sizes $4\ 4\ 4\ 4\ 0$.

While perhaps less natural from a combinatorial standpoint, this definition has a natural interpretation in terms of the semantics of the TCP protocol: When all packets have the same payload p, FB is linearly related to the size of the advertised window [13], p. 385,

$$\mathrm{AdvertisedWindow} = \mathrm{MaxRcvBuffer} - p \cdot FB.$$

where parameter MaxRcvBuffer is a constant. A second motivation is presented in the next section.

In this paper we address the issue of recovering sequence A from $FB(A)$. Specifically, consider a sequence of positive integers $B = \{B_1, B_2, \ldots, B_n\}$, correspond to a potential buffer sequence. We are interested in fast algorithms for deciding whether there exists a sequence A that is mapped to B, and for reconstructing such a sequence $A = \{A_1, A_2, \ldots, A_n\}$, if it exists. We will also consider the problem of counting/sampling from the set of all possible sequences A that map to B. Our results can be summarized as follows:

1. We show (Theorem 2) that deciding whether there exists a packet ID sequence A satisfying $A = FB^{-1}(B)$ and reconstructing A can be done in polynomial time.
2. We consider the counting/sampling problem under a plausible restriction. We show (Theorem 3) that if the maximum value in B is upper bounded by a fixed constant, counting packet IDs sequences A that map to B (in particular deciding whether any such sequence exists) can be done in linear time. The special case is interesting since because even though a sequence of buffer sizes may have many entries, the congestion control mechanism of TCP is likely to keep each individual entry small.
3. As a second special case of the counting problem, we provide (Theorem 6) a *fully polynomial randomized approximation scheme* (fpras) in the case when the maximum allowed number of repeats is upper-bounded by a constant.
4. We provide empirical evidence that for every buffer sequence $B = FB(A)$ occurring in some real-life TCP traces, the smallest of i) the number of repeats in A, and ii) the maximum entry in B, is upper bounded by a small constant. Thus, assumptions of Theorem 3 and Theorem 6 are realistic.

The problems we study (and the previous definition of buffer size) are motivated by RESTORED, a receiver-oriented model of TCP traffic introduced in [7]. For instance, the algorithm in Theorem 2 is the basis for the encoding/decoding mechanism of RESTORED. Results in Theorems 3 and 6 can be used to generate random samples from the preimage of FB, and for the estimation of the size of

this preimage. In the next section we outline the approach behind RESTORED and present the precise definition of FB, as well as its motivation.

2 Motivation

While most analytical models of TCP (e.g. Jaiswal *et al.* [8]) concentrate on modeling the temporal aspects of the TCP connections, RESTORED models the dynamics of packet IDs, in particular attempts to capture *packet reordering*. This phenomenon has recently received significant attention in the networking literature [1, 11, 14]. Bennett *et al.* [2] have shown that packet reordering is more widespread than originally believed, and is increasingly becoming so, due to technological advances such as link striping and mobility.

RESTORED breaks the sequence of packet IDs into multiple *segments*, and maps ID sequences of these segments onto sequences of integers using FB (defined more precisely below). It then stores the histogram of buffer sequences (for the justification of this step see [7]) and uses it, together with the algorithm from Theorem 2 for regenerating segments of packet IDs of the reconstructed trace. FB is precisely defined as follows:

Definition 1 *Let $A = \{A_1, A_2, \ldots, A_n\}$ be a sequence of packet IDs. We define the FB as an operator that after receiving a packet A_i at time index i, outputs the difference between the highest ID (H_i) seen so far and the highest ID (L_i) that could be uploaded.*

$$FB(A_i) = H_i - L_i. \tag{1}$$

In other words, FB is the size of the smallest buffer large enough to store all packets that arrive out of order, where the definition of size accounts for reserving space for unreceived packets with intermediate IDs as well. The buffer sequence *B associated with a sequence of packet IDs is simply a time-series of FB values computed after each packet has been received.*

The precise reason for defining mapping FB lies in the fact that we want to guarantee that regenerated sequence of IDs corresponding to one segment is equivalent to the sequence of IDs of the original segment. The notion of equivalence we use is motivated via the following example: Assume that each received packet is acknowledged, and consider the following two packet ID sequences:[1] 4 2 2 3 1 and 4 2 3 2 1. Both these sequences trigger identical ACK responses, namely 1 1 1 1 5, i.e., we arrive at the following two mappings:

$$\begin{array}{l} 4\ 2\ 2\ 3\ 1 \rightarrow 1\ 1\ 1\ 1\ 5, \\ 4\ 2\ 3\ 2\ 1 \rightarrow 1\ 1\ 1\ 1\ 5. \end{array} \tag{2}$$

Neglecting possible differences in the value of the congestion window, the two ID sequences should be regarded as semantically equivalent.

[1] Assuming that all packets have identical payload, we can employ packet IDs rather than byte IDs.

Definition 2 *Two sequences of packets P and Q are* behaviorally equivalent *if they yield the same sequences of ACKs.*

This is not guaranteed under the ordinary notion of buffer size (see [6] where the same problem is investigated under this notion). For example, returning to example (2), behaviorally equivalent sequences in (2) are mapped onto the same buffer sequence,

$$\begin{array}{l} 4\ 2\ 2\ 3\ 1 \rightarrow 4\ 4\ 4\ 4\ 0, \\ 4\ 2\ 3\ 2\ 1 \rightarrow 4\ 4\ 4\ 4\ 0. \end{array} \tag{3}$$

But under the notion of buffer size they would be mapped to different sequences

$$\begin{array}{l} 4\ 2\ 2\ 3\ 1 \rightarrow 1\ 2\ 2\ 3\ 0, \\ 4\ 2\ 3\ 2\ 1 \rightarrow 1\ 2\ 3\ 3\ 0. \end{array} \tag{4}$$

3 Preliminaries

TCP guarantees to deliver an ordered packet stream to the application layer and needs to buffer packets that arrive out of order. Consequently, the received packets can be classified into two types: those that could be immediately passed to the application layer, and those that have to be temporarily buffered. In (2), packets 4, 3, and 2, are temporarily buffered, and the buffer cannot be flushed until packet 1 is received. A received packet that can flush the buffer is called a *pivot packet*. All packets appearing in order are trivially pivots.

Definition 3 *Let $B = \{B_1, B_2, \ldots, B_n\}$ be a sequence of integers. B is a* valid buffer sequence if there exists at least one packet ID sequence $A = \{A_1, \ldots, A_n\}$ such that

1. A contains exactly one pivot packet, which is the last packet.
2. $B = FB(A)$.

Definition 4 *Given a sequence of packets IDs $(A_1, A_2 \ldots, A_n)$ (displayed in the order they were received), packet A_j is a* repeat *if there exists $i < j$ with $A_i = A_j$.*

We use standard graph theoretic notations throughout. In this paper the graphs are always bipartite $G = (V_1, V_2, E)$ and undirected. Denote by $d(v)$ the degree of vertex v and by $N(v)$ the set of neighbors of v. We will employ a generalized matching problem (GMATCH). The input to GMATCH is a bipartite graph $G = (V_1, V_2, E)$. Solution to GMATCH on input instance G are specified by a set of edges, $E' \subseteq E$, such that E' is incident on all vertices of V_1 and V_2. In addition we require the property that no two edges of E' are incident on a single vertex in V_2. For $A \subseteq V$, $\delta(A)$ denotes the set $\{e \in E : e$ has one end in A and the other end in $V \setminus A\}$. Formally, we can define GMATCH(G) as:

Definition 5 *Consider a bipartite graph $G = (V_1, V_2, E)$.* GMATCH(G) *contains all sets $E' \subseteq E$, such that for all $v \in V_1$, we have $|E' \cap \delta(v)| \geq 1$ and for all $v \in V_2$, we have $|E' \cap \delta(v)| = 1$.*

GMATCH can also be formulated as a variant of the b-matching problem [12]. It is also self-reducible (solving an instance of size n reduces to solving one or more smaller instances) since, for an arbitrary edge $(u, v) \in E, u \in V_1, v \in V_2$, we can represent

$$\text{GMATCH}(G) = \text{GMATCH}(G^-) \cup \{M \cup \{e\} : M \in \text{GMATCH}(G^+)\}$$

where G^- is the graph obtained by deleting e from G, and G^+ the graph obtained by deleting the vertex v with all its incident edges. A consequence of the self-reducibility of GMATCH and the following theorem of Jerrum *et al.* is that uniformly generating a single GMATCH solution is no harder than approximately counting solutions:

Theorem 1 *(Jerrum et al. [10]) Let R be a self-reducible relation over $\sum$ and k_R a constant such that, for all pairs $(x, y) \in R$, $|y| = \mathrm{O}(|x|^{k_R})$. If there exists a polynomial time-bounded deterministic approximate counter for R within ratio $1 + \mathrm{O}(n^{-k_R})$ then there exists a polynomial time-bounded uniform generator for R.*

In Section 5 we will use results on the complexity of problems expressed in *Monadic Second Order Theory* on instances of bounded treewidth. The notions of *tree decomposition* and *treewidth* was introduced by Robertson and Seymour in their work on graph minors [15].

Definition 6 *A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_i, i \in I\}, T = (I, F))$ with $\{X_i, i \in I\}$ a collection of subsets of V, and $T = (I, F)$ a tree, such that:*

1) $\cup_{i \in I} X_i = V$.
2) *For all* $\{v, w\} \in E$, *there is an* $i \in I$ *with* $v, w \in X_i$.
3) *For all* $v \in V, T_v = \{i \in I | v \in X_i\}$ *forms a connected subtree of* T.

The width of a tree decomposition *is* $max_{i \in I}|X_i| - 1$. *The* treewidth of G *is the minimum width of a tree decomposition of* G.

Second order logic is the natural language of graph theory, and most graph theoretic concepts are definable in them. A more detailed description about the expressive power of MS_2 can be found in [5].

Definition 7 *The syntax of the second-order monadic theory* (MS_2) *includes* $\vee, \neg, \wedge$, *variables for vertices, edges, set of vertices, set of edges, the quantifiers* $\forall, \exists$ *that can be applied to these variables, and five binary relations:*

1) $v \in V$, *where* v *is a vertex variable and* V *is a vertex set variable.*
2) $e \in E$, *where* e *is an edge variable and* E *is an edge set variable.*
3) $inc(e, v)$, *where* e *is incident on* v.
4) $adj(u, v)$, *where* u, v *are vertex set variables and they are adjacent.*
5) *Equality for vertices, edges, sets of vertices, and sets of edges.*

In Section 6 we provide a *fully polynomial randomized approximation scheme* for counting ID sequences when the number of repeats is bounded by a constant. The formal definition of fpras is:

Definition 8 *Let f be a function from some set $\sum^*$ of inputs to natural numbers. A fully polynomial randomized approximation scheme for f is a probabilistic algorithm which when given an input $x \in \sum^*$ with an accuracy parameter $\epsilon \in (0, 1]$ outputs a random number X s.t.*

$$\Pr[(1-\epsilon)f(x) \leq X \leq (1+\epsilon)f(x)] \geq 3/4$$

and runs in time polynomial in $|x|$ and ϵ^{-1}. The success probability can be boosted to $1-\delta$ for any $\delta > 0$ by outputting the median of $\mathrm{O}(\log \delta^{-1})$ independent trials.

4 Decoding of Buffer Sequences

In this section we give a polynomial time algorithm for inverting buffer sequences. That is, given a sequence of integers B, we would like to decide whether B is a valid buffer sequence, and if this is the case, find an inverse $A = FB^{-1}(B)$. Our result is actually stronger: we give a parsimonious reduction from our problem to that of finding GMATCH solutions in certain bipartite graphs. A technical assumption we will employ is that repeated packets that have been uploaded to the application layer are discarded. This is a sensible assumption that mirrors TCP behavior. Furthermore, the packets are uploaded as soon as possible. We prove the following theorem:

Theorem 2 *Let $B = (B_1, B_2, \ldots, B_n)$ be a sequence of positive integers. Then there is an algorithm running in time $O(n^{5/2})$ that decides whether B is valid and constructs a ID sequence $A \in FB^{-1}(B)$.*

4.1 Decoding Procedure

To guide the inverse mapping we construct a decoding table, and a new row is appended to the table each time a new element in B is processed. The rows are numbered in order, starting at 1. The columns are also numbered and they represent the packet IDs, which we try to decode. Entries in the table are marked in one of the following four ways:

- **R** if the packet has been **R**eceived but not uploaded.
- **N** if the packet has **N**ot yet been received.
- **U** if the packet has already been **U**ploaded.
- **P** if the packet could **P**ossibly have been received in this stage or earlier.

We also construct a bipartite graph $G(B) = (V_1, V_2, E)$. Nodes in V_1 are labeled by the IDs,[2] whereas the nodes in V_2 are labeled by the time stamps (rows). At

[2] Throughout this paper node i represents node labeled i. Comparison between nodes is actually comparison between their labels.

Assume $B_0 = 0$
1) For each index i in the buffer sequence
a) If $B_i > B_{i-1}$ then $V_1 = V_1 \cup \{H_i\}$,$V_2 = V_2 \cup \{i\}$,$E = E \cup \{(H_i, i)\}$.
b) If $B_i = B_{i-1}$ then $V_2 = V_2 \cup \{i\}$.
c) If $B_i < B_{i-1}$ then $V_1 = V_1 \cup \{L_{i-1} + 1\}$,$V_2 = V_2 \cup \{i\}$,$E = E \cup$ $\cup\{(L_{i-1} + 1, i)\}$.
2) For every $v \in I \setminus V_1$, do $V_1 = V_1 \cup \{v\}$.
3) For every $v \in V_1$ and every $v_1 \in M(v)$, do $E = E \cup \{v, v_1\}$.

Table 1 Function *Transform*(B)

the end of procedure, the nodes in V_2 will thus be $\{1, 2, \ldots, n\}$. In the decoding table we use row $i-1$ as a template for row i with some changes. After each row is appended, we make a decoding decision by evaluating its entries. We call the set of rows for which decisions can be made C. Sometimes we cannot make any decoding decision and the set of such rows is $\overline{C}$.

Both H_i and L_i are nondecreasing sequences. Our algorithm keeps track of them. In the cases when B_i either increases or decreases we can infer the ID of the received packet. $B_i > B_{i-1}$ implies that H_i increases and $B_i < B_{i-1}$ implies that L_i increases. In general, the pair (H_i, L_i) is

$$(H_i, L_i) = \begin{cases} (L_{i-1} + B_i, L_{i-1}) & \text{if } B_i > B_{i-1}, \\ (H_{i-1}, L_{i-1}) & \text{if } B_i = B_{i-1}, \\ (H_{i-1}, L_{i-1} + B_i - B_{i-1}) & \text{if } B_i < B_{i-1}. \end{cases}$$

We define a function *Transform* (Table 1) that produces $G(B)$ from B. The idea behind each step in function *Transform* is:

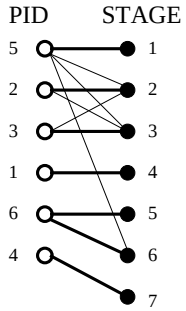
1a) For row i the newly arrived packet ID H_i is marked **R**. All entries between H_{i-1} and $H_i - 1$, are marked **N**. We add i to the set C.

1b) No decision is made. We mark all candidate IDs (L_i+2 to H_i) in row i with **P**. We add a node i to V_2. We add i to the set $\overline{C}$.

1c) The newly arrived packet $L_{i-1} + 1$ causes all packets with IDs in the interval $[L_{i-1} + 2, L_i]$ to be uploaded to the application layer. All IDs in $[L_{i-1} + 1, L_i]$ are marked **U**. We mark $L_i + 1$ as **N**. We update the previous rows by changing any $L_i + 1$ marked **P** to **N**. If the packet $L_i + 1$ has already been received we have a contradiction, and the sequence B is invalid. We add i to the set C.

2) Without loss of generality we assume that packet IDs start from 1. Define I to be the set of all integers in $[1, H_n]$. For all IDs $v \in I \setminus V_1$ we add a new node v to V_1.

3) We finish the construction of $G(B)$ by defining for all nodes $v \in I$, the set of vertices $M(v) \subseteq V_2$ (could be $\emptyset$) that satisfies the following conditions: i) nodes in $M(v)$ have **P** in the vth column of the decoding table. ii) nodes in $M(v)$ have their corresponding rows in $\overline{C}$. For nodes $v \in V_1$ we add edges between v and $M(v)$. The idea is that set $M(v)$ represents the stages at which a packet labeled v could possibly arrive.



Stage / PID	1	2	3	4	5	6
1	**N**	**N**	**N**	**N**	**R**	
2	**N**	**P**	**P**	**N**	**P**	
3	**N**	**P**	**P**	**N**	**P**	
4	**U**	**U**	**U**	**N**	**U**	
5	**U**	**U**	**U**	**N**	**U**	**R**
6	**U**	**U**	**U**	**N**	**P**	**P**
7	**U**	**U**	**U**	**U**	**U**	**U**

Fig. 1 The bipartite graph G between packet IDs and row stages for $B = \{5, 5, 5, 2, 3, 3, 0\}$ with a single GMATCH solution shown in bold. The decoding table constructed during *Transform* is on the right.

Figure 1 illustrates the described procedure. Consider a valid buffer sequence B. Let T be the function from the set of all $A = \{A_1, \ldots, A_n\}$ mapping to B to the set GMATCH$(G(B))$ defined as follows: T maps ID sequence A onto the set of edges, $T(A) = \{(A_i, i)\}$, $i = 1, \ldots, n$.

Lemma 1 *Function T is well-defined and bijective.*

Proof. To show that T is well-defined it is enough to show that $T(A)$ belongs to GMATCH$(G(B))$. Every $v \in V_2$ is incident to exactly one edge in $T(A)$ by the construction of this set. On the other hand, since last packet of A is a pivot all IDs in V_1 appear in A. This shows that $T(A) \in$ GMATCH$(G(B))$.
T is injective because any two distinct Packet ID sequences yield different elements in GMATCH$(G(B))$. Also, every element Γ in GMATCH$G(B)$ has as preimage a Packet ID sequence $A = \{A_1, \ldots, A_n\}$, where A_i is the node in V_1 matched to node $i \in V_2$ in Γ. ❑

Proof of Theorem 2. By Lemma 1 the existence of a packet sequence A is equivalent to the GMATCH$(G(B)) \neq \emptyset$. If $|V_1| > |V_2|$ then GMATCH$(G(B)) = \emptyset$ and B is an invalid sequence. Therefore, assume that $|V_1| \leq |V_2|$. Compute a maximum matching W for $G(B)$ w.r.t. to V_1. If GMATCH$(G(B)) \neq \emptyset$ then W covers all the vertices in V_1. On the other hand, given such a maximum matching W that covers all the vertices in V_1, complete W to an element of GMATCH$(G(B))$ by matching any unmatched node in V_2 with one of its neighbor. Using Hopcroft-Karp maximum cardinality matching algorithm for bipartite graphs gives the required running time.

5 Counting Packet ID Sequences with Bounded FB

The complexity of the algorithm in the previous section is dominated by the complexity of solving a GMATCH problem in the bipartite graph. The running time can be reduced when the FB size is upper-bounded by a constant, and in fact in

this case one can also *count* the number of sequences mapping to a given buffer sequence. We use a result from Courcelle, Makowsky and Rotics [4], on the complexity of counting solutions of a MS_2 definable problem for graphs with bounded treewidth. We prove the following theorem:

Theorem 3 *If the maximum entry in buffer sequence $B = \{B_1, B_2, \ldots, B_n\}$ is bounded by c, counting the number of sequences A satisfying $A = FB^{-1}(B)$ can be done in $\mathrm{O}(n)$.*

We first show that if the maximum entry is bounded by a constant c, then the bipartite graph has a treewidth at most $c + 1$. This bound of c implies that the number of **P** in each decoding row is also at most c, which implies that for all $v \in V_2$, $d(v) \leq c$. A tree decomposition $T = (I, F)$ with a treewidth at most $c+1$ is constructed. Figure 2 represents tree decomposition for $G(B)$ from Figure 1. We start by proving a simple property about $G(B)$ required for proving our results.

Lemma 2 *Let $G(B) = (V_1, V_2, E)$ be the graph produced by Transform for buffer sequence B. Let l_v and r_v denote the neighbors of $v \in V_1$ with smallest and largest labels, respectively. Then for every $k \in V_2$ such that $l_v < k \leq r_v$ either $k \in \overline{C}$ and $v \in N(k)$, or $k \in C$ and $v \notin N(k)$.*

Proof. If a packet is marked **P** in some stage, it remains marked **P** in the following stages until we are sure that we have uploaded it. Also, once the packet has been uploaded it is never marked **P** again. The packet ID v is marked **P** between stages $l_v + 1$ and r_v. If $k \in \overline{C}$, then $k \in M(v)$ which implies that there exists an edge between v and $M(v)$. However, if $k \in C$, then it should correspond to a row where a decision was made and $d(k) = 1$. Therefore, $v \notin N(k)$. ❑

Lemma 3 *Let the maximum entry in B be bounded by a constant c. Then the corresponding graph $G(B) = (V_1, V_2, E)$ has treewidth at most $c + 1$.*

Proof. We create a tree T in the following manner. For every node $i \in V_2$ we create a node in I_i. Let $I = \{I_1, I_2, \ldots, I_n\}$. Define sets X_i of Definition 6 as $X_i = \{i\} \cup N(i)$. The node set I fulfills the first two conditions for a tree decomposition. We assume elements in $\overline{C}$ are sorted. We add tree edges of the following types:
Type I: For any consecutive $i, j \in \overline{C}, i < j$, add an edge between I_i and I_j.
Type II: For every $i \in C, j \in \overline{C}$ add an edge between I_i and I_j if: a) $N(i) \cap N(j) \neq \emptyset$ and b) For no $k' \in \overline{C}, i < k' < j$.
To show the constraint 3 in Definition 6 holds, let us concentrate on a node $v \in V_1$. Each node in $N(v)$ is assigned to a different node in T. We now show that nodes in $N(v)$ form a connected component. From Lemma 2, for every k, $l_v < k \leq r_v$, either $k \in \overline{C}$ implying $v \in N(k)$ or $k \in C$ implying $v \notin N(k)$. The case $v \notin N(k)$ is not relevant for ensuring the connectivity of the set $T_v = \{i \in I | v \in X_i\}$. So we assume $k \in \overline{C}$. We use induction over k to show that I_{l_v} and I_k are connected. For the base case $k = l_v + 1$ we have either a Type I or Type II edge between I_{l_v} and I_k, depending on whether l_v belongs to $\overline{C}$ or C. In the inductive hypothesis let us assume that I_{l_v} and I_k with $k \in N(v)$ form a connected component. In the inductive step we go from k to $k + 1 \leq r_v$ and we again have a Type I edge

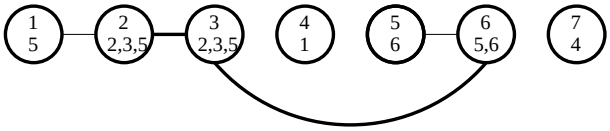


Fig. 2 The tree decomposition of $G(B)$ from Figure 1. The set X_i is displayed inside each node with the top line representing nodes in V_2 and bottom line representing nodes in V_1. The bold edges are Type I edges.

$\phi(E') \equiv \exists V_1 \exists V_2 [part(V, V_1, V_2) \wedge bipar(E', V_1, V_2) \wedge match(E', V_1)$
$\wedge match(E', V_2) \wedge match1(E', V_2)]$ where:
$part(X, Y, Z) \equiv \forall v \in X[((v \in Y) \vee (v \in Z)) \wedge (\neg(v \in Y) \vee \neg(v \in Z))]$
$bipar(F, Y, Z) \equiv \forall e \in F,\ \exists v_1 \in Y,\ \exists v_2 \in Z[inc(e, v_1) \wedge inc(e, v_2)]$
$match(F, Y) \equiv \forall v_1 \in Y,\ \exists e \in F[inc(e, v_1)]$
$match1(F, Z) \equiv \forall v_2 \in Z,\ \forall e_1, e_2 \in F[(e_1 = e_2) \vee (\neg inc(e_1, v_2) \vee \neg inc(e_2, v_2))]$

Table 2 Formulation of GMATCH as MS_2 formula ϕ

between I_k and I_{k+1}. Therefore, we ensure that the constraints are satisfied for all $v \in V_1$. ❑

We now formulate GMATCH problem as an MS_2 formula ϕ (Table 2). We use lowercase letters for vertices or edge variables and uppercase letters for variables representing a set of edges or vertices. We then use the following theorem from Courcelle, Makowsky and Rotics [4] showing that counting of an MS_2 definable problem can be done in polynomial time for graphs with bounded treewidth.

Theorem 4 *(Courcelle* et al. *[4]) Let C be the class of graphs which is of bounded treewidth k'. Then each MS_2 definable counting problem (given by ϕ) can be solved in time $c_{k'} \cdot \mathrm{O}(|V| + |E|)$, where $c_{k'}$ is a constant which depends only on ϕ and k'.*

Proof of Theorem 3. From Lemma 3 we know that $G(B)$ has a tree decomposition with treewidth at most $c + 1$. From Table 2 we know that GMATCH problem can be formulated as an MS_2 formula. Using Theorem 4 we complete the proof.

6 Counting Packet ID Sequences with Bounded Repetitions

In the previous section we have seen that in certain instances one can actually count sequences A satisfying $A = FB^{-1}(B)$ in linear time. Results of Jerrum *et al.* [9] and the following recent improvement of Bezáková *et al.* [3] provide an fpras for the number of perfect matchings in a bipartite graph:

Theorem 5 *(Bezáková* et al. *[3]) For all $\epsilon > 0$, there exists a randomized approximation scheme algorithm to approximate, within a factor $(1 \pm \epsilon)$, the number of perfect matchings in $n + n$-vertex bipartite graph in time $\mathrm{O}(n^7 \log^4 n + n^6 \log^3(n)\epsilon^{-2})$.*

1) For every solution $s_i = \{a_1, \ldots, a_{\lvert V_1 \rvert}\}$ of $\mathcal{X}$ do: a) Construct $G^{(i)}$. b) Repeat O($\log(\lvert 4p(n) \rvert)$) independent trials of the fpras for perfect matchings (Theorem 5) on $G^{(i)}$. c) Let Z_k be a random variable denoting the number of perfect matchings outputted for trial k. Find the median Z_m over all trials. d) Define $W^{(i)} = Z_m/(\prod_{j=1}^{\lvert V_1 \rvert} a_j!)$. 2) Compute the estimator $W = \sum_{i=1}^{p(n)} W^{(i)}$.

Table 3 Algorithm for Obtaining an FPRAS:

Given this result, and the connection with matching problems outlined in Section 4, we expect that our problem of counting all ID sequences also has an fpras. We only prove this in a special case (and leave the general case as an open problem). By the construction in Section 4, the difference $|V_2| - |V_1|$ corresponds precisely to the number of repeats in any sequence A with $A = FB^{-1}(B)$. In this section we obtain an fpras for counting the number of GMATCH solutions in any bipartite graph $G = (V_1, V_2, E)$ assuming that $|V_2| - |V_1|$ is positive and upper bounded by some constant. From Lemma 1, this implies an fpras for counting sequences A satisfying $A = FB^{-1}(B)$. We prove the following theorem in this section.

Theorem 6 *Let $G = (V_1, V_2, E)$ be a bipartite graph with $0 \leq |V_2| - |V_1| \leq c$ for a constant c. There exists a fully polynomial randomized approximation scheme for computing the size of* GMATCH(G). *Consequently, there exists an fully polynomial randomized approximation scheme for counting ID sequences A satisfying $A = FB^{-1}(B)$ if the number of repeats is bounded by some constant.*

We first decompose the set GMATCH(G) into a polynomial number of subsets $S_1, \ldots, S_{p(n)}$. Then for each such subset S_k we construct a bipartite graph $G^{(k)}$ such that the cardinality of set S_k is a constant multiple of the number of perfect matchings in bipartite graph $G^{(k)}$ (with easily computable multiplicity constant c_k). For each graph $G^{(k)}$ we use Theorem 5 to approximate the cardinality of the set S_k with error ϵ. Finally, we will add up all the estimates, obtaining an approximation of the size of GMATCH(G) with error at most ϵ. The algorithm for obtaining an fpras is described in Table 3. We solve a few technical problems to show we have an fpras:

- Each estimator is only guaranteed to be correct with probability at least 3/4. To make sure that the sum of the estimators is correct with probability at least 3/4, we amplify the correctness probability from 3/4 to $1 - (4p(n))^{-1}$. A union bound argument then shows that the probability that none of the amplified estimators is incorrect is at least $3/4$. The desired amplification can easily be accomplished by repeating the procedure for each estimator $\log(4p(n))$ times and taking the median.
- We want to make sure that the total running time of the algorithm is upper bounded by a fixed polynomial $q_0(n, 1/\epsilon)$. To be able to guarantee this, it is enough to show that the running times of all the procedures for approximatively computing the cardinality of all sets S_i is uniformly bounded by a fixed

polynomial $q_1(n, \epsilon^{-1})$. One can then take $q_0(n, \epsilon^{-1}) = p(n) \cdot q_1(n, \epsilon^{-1}) + q_2(n)$, where $q_2(n)$ is the overhead for constructing the bipartite graphs $G^{(1)}, \ldots, G^{p(n)}$, calling the corresponding $p(n)$ estimator procedures, and adding the results.

The decomposition into sets $S_1, \ldots, S_{p(n)}$ is accomplished in an intuitive way: every element in GMATCH(G) is guaranteed to contain exactly one edge incident on all vertices in V_2, and at least one edge incident on all vertices in V_1. For every vertex in $j \in V_1$ we therefore associate a variable $x_j \in \mathbb{Z}^+$. Assume w.l.o.g. that $\{1, \ldots, |V_1|\}$ is the set of nodes in V_1. We define a set of linear equations $\mathcal{X}$:

$$x_1 + x_2 + \ldots + x_{|V_1|} = |V_2| = n \text{ with } 1 \leq x_j \leq d(v_j), \forall v_j \in V_1 \qquad (5)$$

The number of solutions to $\mathcal{X}$ is polynomial in n if $|V_2| - |V_1|$ is bounded by a constant. Let $\{s_1, \ldots, s_{p(n)}\}$ denote the set of solutions to $\mathcal{X}$. Given any $N \in$ GMATCH(G), let E_N denote the edges selected by N. For every $s_i = \{a_1, \ldots, a_{|V_1|}\}$, define S_i to be the subset of GMATCH(G), s.t. for every $N \in S_i$ we have $|\delta(v_j) \cap E_N| = a_j$, $\forall v_j \in V_1$. The size of GMATCH(G) is just the sum of the cardinalities of sets S_i, where the sum ranges over all solutions of $\mathcal{X}$. For every value of $i = 1, \ldots, p(n)$ we construct a new bipartite graph $G^{(i)}$ as:

Construction of $G^{(i)}$ for solution instance $s_i = \{a_1, \ldots, a_{|V_1|}\}$ of $\mathcal{X}$:
$G^{(i)} = (V_1^{(i)}, V_2^{(i)}, E^{(i)})$. Set $V_2^{(i)} = V_2$. For all $j \in V_1$ add a_j number of nodes $\{j_1, \ldots, j_{a_j}\}$ to $V_1^{(i)}$ and to each of them add the same edges that were incident on vertex j in G.

Since for $G^{(i)}$ we have $|V_1^{(i)}| = |V_2^{(i)}|$, the problem GMATCH reduces to finding perfect matchings (PMATCH).

Lemma 4 *Let $s_i = \{a_1, a_2, \ldots, a_{|V_1|}\}$ be a solution instance to $\mathcal{X}$ as defined above for G, with $G^{(i)}$ as its corresponding graph instance. Then*

$$|S_i| = \frac{|\text{PMATCH}(G^{(i)})|}{(\prod_{j=1}^{|V_1|} a_j!)}$$

where PMATCH$(G^{(i)})$ *is the set of all perfect matchings in $G^{(i)}$.*

Proof. Let P be any element from the set PMATCH$(G^{(i)})$. In $G^{(i)}$, $V_2^{(i)}$ equals V_2. Also, by ensuring $1 \leq x_i \leq d(v_i), \forall v_i \in V_1$ we guarantee that $V_1 \subseteq V_1^{(i)}$ and $E \subseteq E^{(i)}$. For every edge of type $(j, k) \in E$ we have a_j number of edges in $E^{(i)}$ from $\{j_1, \ldots, j_{a_j}\}$ to the same $k \in V_2^{(i)}$. As P is a perfect matching we know that only one among these a_j edges could be selected. Therefore, for every edge of the form (j_l, k), $1 \leq l \leq a_j$ we can select (j, k) for constructing a solution in GMATCH(G). Hence, P satisfies that for all $v \in V_1$, $|P \cap \delta(v)| \geq 1$ and for all $v \in V_2$, $|P \cap \delta(v)| = 1$ and therefore $P \in$ GMATCH(G). Also, $P \in S_i$. Therefore, every P has a pre-image in S_i.

Assume $|\text{PMATCH}(G^{(i)})| > 0$. Fix a matching of nodes $V_1^{(i)} \setminus \{j_1, \ldots, j_{a_j}\}$ in $G^{(i)}$. Now consider the $a_j!$ perfect matchings that are created by matching

$\{j_1, \ldots, j_{a_j}\}$ to the remaining unmatched nodes. All these $a_j!$ perfect matching solutions result in the same solution in S_i. Taking the product over all $j \in V_1$ proves the result. ❑

Lemma 5 *The estimator W (Table 3) is an $(1 \pm \epsilon)$ approximation for the size of* GMATCH(G) *with probability at least $3/4$.*

Proof. The size of GMATCH(G) is just the sum of cardinalities of sets S_i, where the sum ranges over all $p(n)$ solutions. If for fixed ϵ and for all $i = 1, \ldots, p(n)$, $W^{(i)}$ is an $(1 \pm \epsilon)$ approximation for $|S_i|$, then W gives an $(1 \pm \epsilon)$ approximation for the sum. By union bound the probability of this happening is at least $3/4$. ❑

Proof of Theorem 6. From Lemma 5 we know that W gives an $(1 \pm \epsilon)$ approximation for the size of GMATCH(G) with desired probability. We have a construction algorithm with time complexity uniformly bounded in polynomial in n. We apply $\log(4p(n))$ times the estimator from Lemma 5 and take the median. The running time of Step 1 of the algorithm for every $i = 1, \ldots, p(n)$ can be uniformly upper bounded by a polynomial $q_1(n, \epsilon^{-1})$. Let the polynomial $q_2(n)$ be the overhead for constructing the bipartite graphs $G^{(1)}, \ldots, G^{p(n)}$, calling the corresponding $p(n)$ estimator procedures, and adding the results. The total running time of the algorithm to obtain an fpras is $p(n) \cdot q_1(n, \epsilon^{-1}) + q_2(n)$.

7 Experimental Validation

In this section we provide experimental evidence that results in Sections 5 and 6 suffice to efficiently count the number of preimages for most buffer sequences observed in real TCP data. To test this, we used five sets of TCP traces (Trace 5, 7, 8, 9, 10) collected during August 2001 at the border router of the Computer Science Department, University of California, Los Angeles (UCLA). The set was obtained by the UCLA Network Research Lab. Each trace, containing over 15 million packets, was analyzed and buffer sequences were computed.

For every buffer sequence B in the given trace, we find the maximum buffer size, i.e. the largest entry in B ($MFB(B)$). Figure 3a) shows the distribution of $MFB(B)$ for Trace 9. It can be seen for more than 93.10% of buffer sequences the $MFB(B)$ is less than 10 and for 99.97% of the buffer sequences it is less than 100. For Trace 9, Figure 3b) displays $MFB(B)$ against the total number of repeats ($TNR(B)$) for every buffer sequence B. In Table 4, we summarize our results for different traces. The results provide evidence for our claim that $MFB(B)$ is a small constant for almost all sequences. They also show that the largest of min($MFB(B)$,$TNR(B)$) over all buffer sequences B is upper bounded by a small constant.

8 Conclusions

We have studied the complexity of a combinatorial problem motivated by a model of the TCP protocol that incorporates information on packet dynamics. The problem amounts to finding preimages of a many-to-one mapping FB that transforms

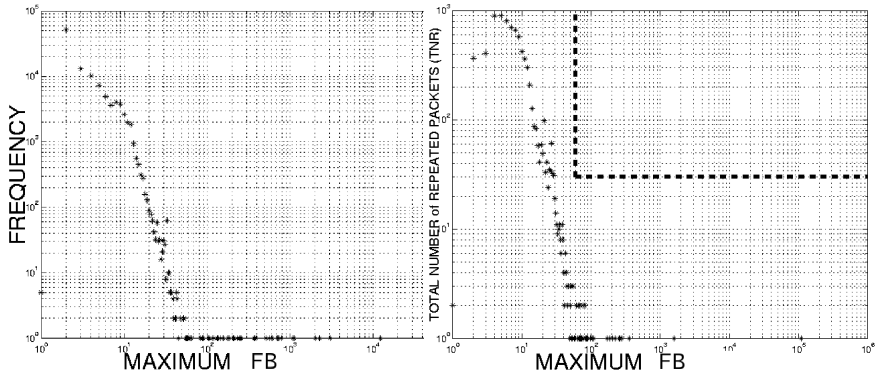


Fig. 3 a) Distribution of the $MFB(B)$ in Trace 9. b) $MFB(B)$ against $TNR(B)$ for each buffer sequence B in Trace 9.

Trace	*% of B with $MFB(B) < 100$*	*% of B with $MFB(B) < 10$*	*Max. over all B of min($MFB(B)$,$TNR(B)$)*
Trace 5	99.97	95.95	20
Trace 7	99.95	95.76	10
Trace 8	99.97	95.67	30
Trace 9	99.97	93.10	30
Trace 10	99.99	98.72	10

Table 4 Results on Maximum FB and Total Number of Repeats. The last column indicates that the maximum over min($MFB(B)$,$TNR(B)$) is practically a small constant.

sequences of packet IDs into buffer sequences. Our results show that deciding whether the preimage of FB is empty, constructing an element of this preimage, or counting the number of such elements can be done in polynomial time under realistic assumptions. Deciding the complexity of the counting version in the general case remains an open problem. We believe that this problem is $\#P$-complete.

9 Acknowledgements

Work of the third author was performed while visiting the CCS-5 group at Los Alamos National Laboratory. This work has been supported by the U.S. Department of Energy under contract W-705-ENG-36.

References

1. J. Bellardo and S. Savage. Measuring packet reordering. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, pages 97–105, Marseille, France, November 2002.

2. J. C. R. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networking*, 7(6):789–798, December 1999.
3. I. Bezákova, D. Štefankovič, E. Vigoda, and V. V. Vazirani. Accelerating simulated annealing algorithm for the permanent and combinatorial counting problems. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA'06)*, pages 900–907, 2006.
4. B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1-2):23–52, 2001.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
6. G. Istrate and A. Hansson. Counting preimages of TCP reordering patterns. submitted to Discrete Applied Mathematics, October 2005.
7. G. Istrate, A. Hansson, S. Thulasidasan, M. Marathe, and C. Barrett. Semantic compression of TCP traces. In *Proceedings of the IFIP NETWORKING Conference, F. Boavida (editor)*, volume 3976 of *Lecture Notes in Computer Science*, pages 123–135. Springer Verlag, 2006.
8. S. Jaiswal, G. Iannacone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP connection characteristics through passive measurements. In *Proceedings of INFOCOM'04*, volume 3, pages 1582–1592, Hong Kong, China, 2004.
9. M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of the ACM*, 51(4):671–697, July 2004.
10. M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.
11. M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, pages 28–36, 2002.
12. L. Lovász and M. Plummer. *Matching Theory*. Number 29 in Annals of Discrete Mathematics. North Holland, 1986.
13. L. Peterson and B. S. Davie. *Computer Networks. A Systems Approach*. Morgan Kauffman, San Francisco, CA, 2nd edition, 2000.
14. N. M. Piratla, A. P. Jayasumana, and A. A. Bare. RD: A formal, comprehensive metric for packet reordering. In *Proc. IFIP Networking Conference 2005*, volume 3462 of *Lecture Notes in Computer Science*, pages 78–89. Springer Verlag, 2005.
15. N. Robertson and P. D. Seymour. Graph minors II, algorithmic aspects of tree width. *Journal of Algorithms*, 7:309–322, 1986.