Identifying almost sorted permutations from TCP buffer dynamics

Gabriel ISTRATE[1]

Abstract

Associate to each sequence A of integers (intending to model packet IDs in a TCP/IP stream) a sequence of positive integers of the same length $\mathcal{M}(A)$. The i'th entry of $\mathcal{M}(A)$ is the size (at time i) of the smallest buffer needed to hold out-of-order packets, where space is accounted for unreceived packets as well. Call two sequences A, B *equivalent* (written $A \equiv_{FB} B$) if $\mathcal{M}(A) = \mathcal{M}(B)$.

For a sequence of integers A define $SUS(A)$ to be the *shuffled-up-sequences* reordering measure defined as the smallest possible number of classes in a partition of the original sequence into increasing subsequences. We prove the following result: any two permutations A, B of the same length with $SUS(A)$, $SUS(B) \leq 3$ such that $A \equiv_{FB} B$ are identical. The result is no longer valid if we replace the upper bound 3 by 4.

We also consider a similar problem for permutations with repeats. In this case the uniqueness of the preimage is no longer true, but we obtain a characterization of all the preimages of a given sequence, which in particular allows us to count them in polynomial time.

The results were motivated by explaining the behavior and engineering RESTORED, a receiver-oriented model of traffic we introduced and experimentally validated in earlier work.

Keywords: algorithms, packet reordering, shuffled up sequences.

[1] Department of Computer Science, West University of Timişoara and the e-Austria Research Institute, Bd. V. Pârvan 4, cam 045B, 300223 Timişoara, Romania, Email: `gabrielistrate@acm.org`

1 Introduction

The TCP protocol [16] is the fundamental protocol for computer communications. TCP breaks the information into *packets*, and attempts to maintain a ordered packet sequence to be passed to the application layer. It accomplishes this by *buffering* packets that arrive out-of-order.

Recent work in the area of network traffic modeling has brought to attention the significant impact of packet reordering on the dynamics of this protocol [3, 2, 11]. This has stimulated research (mainly applied) on measuring and modeling reordering [14, 13], and on quantifying the impact of packet reordering on application performance.

In this paper we study a combinatorial problem motivated by modeling packet reordering in large TCP traces: suppose that we map a sequence A of packet IDs into the sequence of integers $\mathcal{M}(A)$ representing the different sizes of the buffer space necessary to store the out-of-order packets; we assume that space in the buffer is reserved (and accounted) for unreceived out-of-order packets as well. What kind of additional information on the sequence A is needed to identify A, given $\mathcal{M}(A)$?

The problem arose in the context of RESTORED [8], a method for receiver-oriented modeling and compression of large TCP traces. In an experimental paper we showed [8] that RESTORED is able to regenerate sequences similar to the original sequences with respect to several reordering metrics. One metric for which this result is true was *reorder density* (RD) from [14, 9, 15]. We found the experimental result for RD paradoxical for the following reason: RESTORED guarantees that the regenerated trace is (locally) similar to the original sequence for a precise notion called $\equiv_{FB}$-equivalent (rigorously explained below). On the other hand $\equiv_{FB}$ equivalence does not uniquely determine the value of measure RD; thus reconstructed sequences have no special reason to have the same RD value as the original sequence: they could get any value compatible with FB equivalence.

Though possible in principle, the scenario we outlined never happened in our experiments with metric RD in [8]. One could attribute this either to the particularities of our reconstruction method or to the existence of "extra structure" in real-life TCP traces that would somehow preclude inconsistency.

The theoretical result in this paper (Theorem 1 below), together with an experimental observation we made in [6] (that over 99% of the traces we previously considered for benchmarking RESTORED have values of the

SUS reordering measure at most 3), makes this result somewhat less paradoxical: indeed Theorem 1 proves that all reordering patterns arising from permutations (i.e. sequences with no repeated or lost packets) satisfying condition $SUS \leq 3$ have at most one preimage with these properties; Therefore all reordering measures are consistent on them.

On the other hand Theorem 2 below provide (as discussed in Section 5.2 below) a way to design a better encoding for the unordered phase of RESTORED (see below); the advantage (compared with the method described in [8]) is an easier method to randomly sample preimage sequences.

To sum up: the two theoretical results in this paper (Theorems 1 and 2) serve to explain some paradoxical behavior of a software system we had previously designed for inference of large network traces and help to better engineer it.

2 Preliminaries

We first give a brief primer on the relevant aspect of the TCP protocol, Restored and the concepts used in the sequel.

2.1 A brief introduction to networking

The TCP protocol [16] attempts to maintain an ordered stream of data bytes, identified by an integer called *byte ID*, that is effectively communicated through the network by breaking it down into *packets*. The ordering is maintained by *buffering* out-of-order packets. Buffer dynamics can be described in part using several parameters:

1. The first parameter is *NextByteExpected*, and is the smallest index of a data byte that has still not been received by the receiver.

2. A second, related, parameter is *LastByteRead*, the index of the last byte processed by the receiver-side application that communicates through the network via the TCP protocol. Throughout this paper we will make the simplifying assumption that data is read by the application as soon as it is ready. In other words NextByteExpected = LastByteRead+1.

3. Another parameter is *LastByteRcvd*, the index of the last byte that has arrived at the receiver, awaiting processing.

4. *RcvWindow*, the size of the *receiver window*, is a parameter that is meant to provide the sender with an estimate of the available buffer space at the receiver.

5. Finally, *RcvBuffer* is a implementation-dependent system constant, the size of the receiving buffer.

The functioning of the TCP protocol ensures that these four parameters are related through the relation ([10] section 3.5):

$$\text{RcvWindow} = \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead}). \qquad (1)$$

The term in parentheses on the right-hand side is the actual size of the TCP receiver-buffer. The measurement takes into account space reserved (but not necessarily used) for all packets from the first expected to the last arrived. This is, of course, proportional to the buffer size measured in packets rather than bytes if all packets have the same size.

TCP is *receiver-driven*: that is, the receiver attempts to maintain control on the sender flow stream by directing the sender speed, and *acknowledging* the received packets. An acknowledgment (shortly, ACK) generally consists of the ID of the *first packet that has not yet been received.* Acknowledgment mechanisms vary across different implementations, and can entail *delayed* or *selective* acknowledgments, urgent retransmission requests, etc. From our standpoint, what is important that we can associate a sequence of integer ACKs to every sequence of packet IDs, the sequence of ACKs that would be sent if the receiver would immediately ACK every packet received.

Example 1 *Consider the following hypothetical sequence of packet IDs:* $A = (4 \quad 3 \quad 2 \quad 1)$. *Then the sequence of ACKS is* $ACK(A) = (1 \quad 1 \quad 1 \quad 5)$.

2.2 A primer on RESTORED

Restored [8] is a Markovian model of large TCP traces that incorporates information on the dynamics of packet reordering. It can be used to provide estimates of various measures of quality of service without making these measurements online, or storing the entire sequence. Rather, it first "compresses" the trace into a small "sketch" that allows regeneration of a TCP trace with (hopefully) similar characteristics. If needed, we can then perform a large number of measurements on the regenerated trace. Thus, the way Restored is envisioned to work involves the following steps:

1. “Learn” Markovian models (and their associated parameters) instead of storing large TCP traces.

2. Use these models to reconstruct similar sequences.

3. Use these sequences to estimate measures of quality of service for the original sequences.

Of course, one could hope to further add features to the model, beyond the details provided by Restored by, for instance, finding efficient ways of comparing and clustering connections with similar traffic properties. We will not discuss these engineering aspects further, but concentrate on some mathematical aspects relevant to engineering Restored .

For the purposes of the present paper, a *connection* is simply a sequence of integers (packet IDs). Suppose that the receiver observes the following (hypothetical) packet stream

$$1 \quad 2 \quad 3 \quad 6 \quad 5 \quad 7 \quad 4 \quad 8 \quad 9 \quad 10 \quad 12 \quad 13 \quad 14 \quad 11.$$

In this example packets with IDs $4, 5, 6, 7, 12, 13, 14$ and 11 arrive out of order. One can, consequently, classify the received packets into two categories: those that can be immediately passed to the application layer, and those that have to be temporarily stored before delivery. In the example, packets 5, 6, and 7 are temporarily buffered, and the buffer is only flushed when packet 4 is received. Similarly, packets 12, 13, and 14 are temporarily buffered, and the buffer is flushed when packet 11 arrives. We will call a packet that marks the end of a sequence of consecutively buffered packets a *pivot packet.* Packets that are immediately delivered to the application layer are also trivially pivots. In our example this is the case for packets 1, 2, 3, 4, 8, 9, 10 and 11.

The distinction we introduced effectively defines a partition of the stream of packet IDs into *segments.* A segment of packets is bounded by pivot packets. There are two possibilities for describing packet dynamics

- packets arrive in order. The largest consecutive subsequence of packets for which this property holds will be represented in Restored as the occurrence of an *ordered state.*

- there is reordering and buffering. Such a segment bounded by consecutive pivot packets form an occurrence of an *unordered state.*

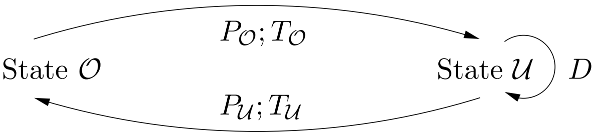
$P_{\mathcal{O}}; T_{\mathcal{O}}$
State $\mathcal{O}$
State $\mathcal{U}$
D
$P_{\mathcal{U}}; T_{\mathcal{U}}$

Figure 1: Markovian model of packet dynamics in RESTORED.

Each occurrence of the ordered state is followed by one or more occurrences of the unordered state. It was the contribution of [8] to show that one can use a Markov Chain with the two states (see Figure 1) to represent the dynamics of packet IDs of real traces.

The sequence of packet IDs in the ordered state is trivial by definition: they arrive in order, starting with the first expected packet. For the unordered state we proposed in [8] to deal with packet IDs in the unordered states by encoding reordering patterns into "sketches" via a many-to-one mapping $\mathcal{M}$ defined on sequences of packet IDs. The mapping reported in [8] was the following:

Definition 1 *Let $A = \{A_1, A_2, \ldots, A_n\}$ be a sequence of packet IDs (considered on the receiver side). First define:*

- *H_i is the highest ID in the sequence $A_1, A_2, \ldots, A_i$.*
- *L_i is the largest ID of a packet among $A_1, A_2, \ldots, A_i$ that can be uploaded, zero if none of them can. Equivalently $L_i = ACK_i - 1$.*

Further, define buffer sequence *$\mathcal{M}(A)$ associated with sequence A as follows. $M(A) = (M_1, \ldots, M_n)$ with*

$$\mathcal{M}_i = H_i - L_i. \tag{2}$$

In other words, $\mathcal{M}$ 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.

Two sequences of packet IDs P and Q are full buffer (FB) equivalent *(written $P \equiv_{\mathrm{FB}} Q$) if $\mathcal{M}(P) = \mathcal{M}(Q)$.*

Example 2 *Let $A = (4 \quad 3 \quad 2 \quad 1)$. Then $\mathcal{M}(A) = (4 \quad 4 \quad 4 \quad 0)$.*

The mapping $\mathcal{M}$ is many-to-one, but a preimage (when it exists) can be computed in polynomial time [7]. This was used in the regeneration algorithm, where first we use the Markov chain to sample a sequence of ordered and unordered states. In any occurrence of the ordered state we simply sample from the distribution of possible lengths of such sequences. On the other hand, in each occurrence of the unordered state we first sample a sketch S from the distribution of such sketches and then reconstruct a preimage (via $\mathcal{M}$) of S.

Furthermore, mapping $\mathcal{M}$ defined this way provides a formal way to guarantee that the reconstructed sequence is locally "similar" to the original one. The formal notion of similarity has implication for the dynamics of the TCP protocol:

Definition 2 *Two packet sequences A, B are* behaviorally equivalent *if they yield the same sequence of ACKs.*

Suppose now that a TCP implementation uses simple (as opposed to selective or cumulative) ACKs , and acknowledges every single packet. Then *two traces that map (via $\mathcal{M}$) to the same sequence are behaviorally equivalent* [5]. As the dynamics of the congestion window is receiver-driven, assuming identical network conditions for the ACK sequences, such traces can be regarded as "equivalent," from a receiver-oriented standpoint.

We will also need a standard measure of disorder [4]. This measure is denoted by shuffled up-sequences (SUS) and is defined as follows:

Definition 3 *Given sequence of integers A denote by $SUS(A)$* the minimum number of ascending subsequences into which we can partition A.

For example, a sequence $A = \langle 6, 5, 8, 7, 10, 9, 12, 11, 4, 3, 2\rangle$ has

$$\mathrm{SUS}(A) = \|\{\langle 6, 8, 10, 12\rangle, \langle 5, 7, 9, 11\rangle, \langle 4\rangle, \langle 3\rangle, \langle 2\rangle\}\| = 5, \tag{3}$$

where $\|S\|$ denotes the cardinality of a set S.

3 Main result

In this section we will prove our main result:

Theorem 1 *Let A, B be permutations of length n with $SUS(A), SUS(B) \leq 3$ such that $A \equiv_{FB} B$. Then $A = B$.*

Observation 1 *The theorem is no longer true if we replace the condition with $SUS(A)$, $SUS(B) \leq 4$. This is witnessed by sequences* $(4 \quad 3 \quad 2 \quad 1)$ *and* $(4 \quad 2 \quad 3 \quad 1)$*. Indeed $A \equiv_{FB} B$, since they both map to* $(4 \quad 4 \quad 4 \quad 0)$*. In fact $SUS(A) = 4$, $SUS(B) = 3$.*

SUSGreedy(W)

```
INPUT: $W = (p_1, p_2, \ldots, p_n)$
a list of non-negative integers.

let $i = 1$;
let $j = 1$;
let $L_1$ be the empty list;
while $(i \leq n)\{$
add $p_i$ to the first list $L_t$, $t \leq j$
where it can be added while
keeping it sorted;
if this is not possible
    {
        j++;
        create new list $L_j = \{p_i\}$;
    }
i++;
}
let $u$ be the number of lists
created by the algorithm;

OUTPUT
$u = LDS(W) = SUS(W)$.
```

RECONSTRUCT

```
INPUT: list $W = (w_1, w_2, \ldots, w_n)$
of positive integers in the range $1 \ldots n$

let PACKET and ACK be integer vectors
of size $n$, with all fields $-1$;
let $LARGEST = 0$, $ACK[0] = 0$;
also let $w_0 = 0$.
for $(i = 1$ to $n)$
    if $(w_i < w_{i-1})\{$
        $PACKET[i] = ACK[i-1]$;
        ACK[i]:=ACK[i-1]+$(w_{i-1} - w_i)$;
} else {
    ACK[i]=ACK[i-1];
    if $(w_i > w_{i-1})$
        LARGEST:= PACKET[i]:=
            := LARGEST+$(w_i - w_{i-1})$;
}
for $(i = 1$ to $n)$
    if $(w_i = w_{i-1})\{$
        let $PACKET[i]$ be the smallest
        integer not in $\{PACKET[j]\}_{j=1\ldots i-1}$
}
if (PACKET is a permutation of $\{1, \ldots, n\}$)
return PACKET;
else return NO PERMUTATION EXISTS;
```

Figure 2: (a) Greedy Algorithm for computing SUS (b).Algorithm for reconstructing permutations from buffer sizes

Proof:

We consider the greedy algorithm in Figure 2(a). The algorithm is related to patience sorting [1] and has been implicitly shown to compute measure SUS in [12]; the reason is that SUS coincides [12] with another reordering measure denoted by LDS, defined as follows:

Definition 4 *Let $A = (a_1, a_2, \ldots, a_n)$ be a sequence of nonnegative integers. $LDS(A)$ is defined as the longest length of a decreasing subsequence $a_{i_1} > a_{i_2} > \ldots a_{i_j}$ $(1 \leq i_1 < i_2 < \ldots < i_j \leq n)$ of A.*

But it is well-known [1] that patience sorting (and indeed, algorithm SUSGreedy) computes parameter LDS (to make the paper self-contained we will reprove this result below).

We next give a simple algorithm called RECONSTRUCT, displayed in Figure 2 (b), that, given a sequence W of positive integers constructs (if possible) a permutation A of size n with $SUS(A) \leq 3$ such that $\mathcal{M}(A) = W$. The proof that the algorithm is correct will imply the uniqueness of sequence A.

We prove the correctness of algorithm RECONSTRUCT in a couple of intermediate steps. The first two apply to a general sequence A (rather than one with $SUS(A) \leq 3$).

Lemma 1 *Suppose there exists a permutation π with $\mathcal{M}(\pi) = w$. Then the following are true for any $i \geq 1$:*

1. *The last element added to list L_i is the maximum element in lists L_k, $k \geq i$. In particular the largest element of L_1 is the maximum element seen so far.*
2. *If element x is the largest element seen up to step i then $x = ACK_i + \mathcal{M}_i - 1$.*

Proof: Let $i = 1$. Statement 1. is clearly true. For the second statement, note that $ACK_1 = 2$ and $\mathcal{M}_1 = 0$ if $x = 1$ (in-order packet) otherwise $ACK_1 = 1$, $\mathcal{M}_1 = x$.

Consider now the case $i > 1$. By the induction statement, the largest element seen so far (call it y) is the last element of L_1 and $y = ACK_{i-1} + \mathcal{M}_{i-1}$.

Case 1: x is added to L_1. By the definition $x \geq y$ so x is the largest element seen so far. Moreover, since x is an out-of-order element we have $ACK_i = ACK_{i-1}$ and $\mathcal{M}_i = \mathcal{M}_{i-1} + x - y$.

Case 2: x is added to some other list L_j. If x is the first element of the new list then statement 1 follows immediately. Otherwise let z be the largest element of list L_j *before* adding x. Applying the induction hypothesis it follows that z is the largest element in lists L_k, $k \geq j$. But $z \leq x$ (since we add x to list L_j). Thus x becomes the new largest element of lists L_k, $k \geq j$. As for the second statement, from the algorithm it follows that $x < y$ so y is still the largest element seen so far. If the buffer size does not modify then the desired relation follows from $y = ACK_{i-1} + \mathcal{M}_{i-1}$ (which holds by induction) and relations $ACK_i = ACK_{i-1}$ and $\mathcal{M}_i = \mathcal{M}_{i-1}$. Otherwise the buffer shrinks with size $ACK_i - ACK_{i-1}$, so $\mathcal{M}_{i-1} - \mathcal{M}_i = ACK_i - ACK_{i-1}$. We infer the fact that

$$\begin{aligned} y &= ACK_{i-1} + \mathcal{M}_{i-1} - 1 = ACK_i - (ACK_i - ACK_{i-1}) + \mathcal{M}_{i-1} - 1 = \\ &= ACK_i + (\mathcal{M}_i - \mathcal{M}_{i-1}) + \mathcal{M}_{i-1} - 1 = ACK_i + \mathcal{M}_i - 1. \end{aligned}$$

□ □

Corollary 1 *Algorithm SUSGreedy correctly computes* $u = LDS(A)$ *(which is equal to* $SUS(A)$ *[12]).*

Proof: Let $B = a_{i_1} > a_{i_2} > \ldots > a_{i_{LDS(A)}}$ be a decreasing subsequence of W of maximum length, and let $L_1, L_2, \ldots, L_j$ be the lists created by the algorithm on input sequence A. Each list L_k is increasing, so it contains at most one element from B. Therefore $u \geq LDS(A)$. On the other hand, each element a_m set by the algorithm to a list L_k, $k \geq 2$ is smaller than some element a_n, $n < m$, set by the algorithm to list L_{k-1} (otherwise a_m would be set to a list L_j, $j < k$). Applying this observation starting with the last element of list L_u we create a decreasing sequence of length u. It follows that $u \leq LDS(A)$, thus $u = LDS(A)$. □

From now on we assume that there exists a permutation A such that $SUS(A) \leq 3$ such that $\mathcal{M}(A) = w$. We will run the algorithm SUSGreedy along algorithm RECONSTRUCT. First we give a simple corollary of Lemma 1:

Corollary 2 *Suppose that* $w_i > w_{i-1}$*. Let* y *be the largest ID of a packet received in stages 1 to* $i-1$ *and* x *be the ID of the new packet. Then* $x = y + (w_i - w_{i-1})$ *and* x *is added by SUSGreedy to list* L_1*.*

Next we deal with another possible case, the one when the buffer size shrinks:

Lemma 2 *(a). Let packet ID x be added at stage i, and assume that $w_i < w_{i-1}$ Then $x = ACK_{i-1}$ and all packets with indices at most $ACK_{i-1} + (w_{i-1} - w_i - 1)$ have been received in the first i stages.*

(b). Suppose packet ID x is added by algorithm SUSGreedy to list L_3. Then packet x falls into case (a) of this lemma.

Proof:

(a). The fact that $x = ACK_{i-1}$ follows from the definition of parameter ACK and the fact that the buffer shrinks. The second relation follows from the fact that the buffer shrinks by exactly $w_{i-1} - w_i$.

(b). Since x goes in list L_3, at the time when it is added x is smaller than the last element in lists L_1 and L_2. If x were larger than ACK_{i-1} then the packet with index ACK_{i-1} (which arrives sometimes after x does) could not be placed in lists L_1, L_2 or L_3, making the sequence A require $SUS(A) \geq 4$, a contradiction.

The other two relations follow from the definition of parameter ACK_i.

□

Lemma 3 *In the conditions of Theorem 1 there exists at most one permutation π with $\mathcal{M}(\pi) = w$, and it is the one found by algorithm RECONSTRUCT.*

Proof:

This follows easily: if $w_i \neq w_{i-1}$ then by Corollary 2 and Lemma 2 the ID of the packet is uniquely determined. To prove the Lemma we have to show that this is true for the case $w_i = w_{i-1}$ as well. We claim that if a packet ID x is set at stage i in the second for loop then *it must correspond to adding x to list L_2*. Indeed, if x were added to L_1 then it would be the largest element seen so far (hence we would have $w_i > w_{i-1}$). The fact that x cannot be added to L_3 follows from Lemma 2.

Since list L_2 is sorted, x **is the smallest element that has not been set up to this stage**. This constraint uniquely determines the value of x, thus proving Lemma 3. □

4 Extension to permutations with repeats

In [7] we have also considered extending encodings of permutations with repeats, defined as follows:

Definition 5 *A* permutation with repeats *is a sequence of integers* $A = A_1, \ldots, A_m$ *such that,*

1. *for some* $n \geq 1$,
$$\{A_1, A_2, \ldots, A_m\} = \{1, 2, \ldots, n\}$$
(as sets).

2. *For any repeat packet* A_i*, we have*
$$A_i > ACK_i.$$

The second condition is a reasonable assumption from the standpoint of TCP behavior: namely, repeats of a packet that has already been uploaded to the application layer are removed from consideration.

In that paper the encoding of permutations was *not* $\mathcal{M}$, but a different map B. In the sequel we define a similar extension of map $\mathcal{M}$:

Definition 6 *Let* A *be a permutation with repeats. Let* $\overline{A}$ *the subsequence of* A *obtained by eliminating all repeats. Define* $\overline{M}$ *as follows:* $\overline{M}(A)$ *is obtained by inserting* -1*'s into* $M(\overline{A})$ *in positions corresponding to repeat packets.*

Also, for two packet sequences define $A \equiv_{\overline{FB}} B$ *iff* $\overline{M}(A) = \overline{M}(B)$.

Example 3 *If*
$$A = (4 \quad 5 \quad 6 \quad 2 \quad 4 \quad 1 \quad 3)$$
then
$$\overline{M}(A) = (4 \quad 5 \quad 6 \quad 6 \quad -1 \quad 4 \quad 0).$$

Theorem 1 does *not* extend to the case of permutations with repeats:

Example 4 *Let* A*,* B *be the following two sequences.*
$$A = (4 \quad 5 \quad 6 \quad 2 \quad 4 \quad 1 \quad 3)$$
and
$$B = (4 \quad 5 \quad 6 \quad 2 \quad 5 \quad 1 \quad 3)$$
Then $A \equiv_{\overline{FB}} B$ *and* $SUS(A), SUS(B) = 3$.

However, as we show in the proof of the following result we can easily obtain a compact representation of all sequences in the preimage of a given string. This representation allows us to count the number of such sequences in polynomial time:

Theorem 2 *There exists an algorithm that, given $P = P_1 \ldots P_n$ a sequence of positive integers counts the number U of permutations with repeats $A = A_1, A_2, \ldots, A_n$ such that $A \in \overline{M}^{-1}(P)$ and $SUS(A) \leq 3$ in time polynomial in the sizes of P and U.*

Proof: We will attempt to give a characterization of sequences A with the properties listed in the above Theorem that will enable us to count them efficiently.

First, note that the conclusions of Lemmas 1 and 2 are still true for permutations with repetitions as well, since its proof we never used the fact that we are getting distinct elements.

Lemma 4 *No repeat packet is added to list L_3.*

Proof: This is a simple consequence of the fact that every packet added to list L_3 is (by Lemma 1) an ACK packet. Once received, such an ACK packet is longer considered (by the convention we made in Definition 5) among repeats. □

Lemma 5 *Consider a permutation with repeats A with the properties in Theorem 2 and its subsequence $\overline{A}$ obtained by removing all repeats. Then packets of A that belong to $\overline{A}$ are set when running $SUSGreedy(A)$ on the same list as when running $SUSGreedy(\overline{A})$.*

Proof: For lists L_1 and L_3 this follows directly from Lemma 1 and Lemma 2 for permutations with repetitions.

On the other hand, a packet added by $SUS(Greedy(\overline{A})$ to L_2 could only be added by $SUSGreedy(A)$ to L_2 or L_3, because, by Lemma 1 packets added to L_1 are the largest seen so far, which the given packet is not.

It cannot, however, be added by $SUSGreedy(A)$ to L_3 because, by Lemma 2 for permutations with repetitions, it would have to be an ACK packet; however we assumed it's a repeat. □

With Lemma 5 settled, proving Theorem 2 is easy: first we eliminate the -1's from sequence W and apply Theorem 1 to recover the unique permutation in the preimage. That allows us to the values A_i for all indices

i such that $W_i \neq -1$. In particular let I_2 be the (ordered) set of indices i for which packet A_i goes to list L_2. We can also simulate the buffering process and determine, for any such i the set B_i of packets buffered at moment i, *excluding the largest such packet.*

It remains to determine the possible values A_i for those positions corresponding to repeats (i.e. $W_i = -1$). Such a packet can either be:

1. A repeat of the largest packet on the list L_1 at time i, or
2. A repeat of a packet that is buffered at stage i that will be added by $SUSGreedy(A)$ on list L_2.

We will call indices i *positions of type 1* and *of type 2*, based on which of the two cases applies.

Repeats of the first type do not create any additional constraint for subsequent packets, as they don't modify the value of the largest packet on list L_1, or other parameters (ACK, etc). Also the value of such a packet is uniquely determined: it's simply a copy of the last element on list L_1 (if $P_1 = -1$ simply return 0, as the sequence cannot start with a repeat packet).

As for repeats of the second type, there are three constraints they have to satisfy:

1. First, any such packet has to be a repeat of a copy of a packet that is buffered at the moment it is added onto L_2, *different from the largest buffered packet* (or else it would be added to L_1).
2. Their addition to L_2 must result in the maintenance of an increasing order on this list.

The first condition simply states that $A_i \in B_i$. The second condition requires some care. Let $1 \leq i \leq n$ and let $i_1 < i < i_2$ be the consecutive moments in I_2 when non-repeat packets are added to L_2. Then the value A_i of the repeat packet must correspond to a value $A_{i_1} \leq A_i < A_{i_2}$. One of i_1, i_2 might fail to exist, in which case we replace the missing limit in the previous inequality by 0 (in the case of the lower value) respectively $+\infty$.

Define, for each i with $P_i = -1$,

$$C_i = B_i \cap [A_{i_1}, A_{i_2})$$

Thus, to specify a sequence A as in Theorem 2 we have to specify:

- (i): which of the indices i with $P_i = -1$ are of the second type.
- (ii): an integer $A_i \in C_i$ for any such i such that the resulting sequence is monotonically nondecreasing.

The conclusion of this argument is that permutations with repeats A in the preimage of W are in bijective correspondence with *nondecreasing partial functions f defined on a subset of indices $i \in \{j|W_j = -1\}$ with the restriction that for any such i such that $f(i)$ is defined $f(i) \in C_i$.*

We can count such sequences in polynomial time by dynamic programming as follows:

1. we start with an empty table and go in increasing order of indices i.
2. We count partial functions by their largest value.
3. Given a possible value A_i in the list C_i, its count is one plus the sum of all the counts of sequences ending in values of the table less or equal to A_i taken *from previous stages.*
4. The output value is one (corresponding to the empty sequence) plus the sum of all the counts of elements in the table.

Example 5 *We illustrate the above dynamic programming algorithm on three sets (without necessarily corresponding to packet sequences), $A_1 = \{1, 2\}$, $A_2 = \{2, 3\}$, $A_3 = \{3, 4\}$. The table corresponding to counting the partial functions is displayed in Figure 2.*

Stage	index	count
1	1	1
1	2	1
2	2	3
2	3	3
3	3	9
3	4	9
TOTAL	-	27

Figure 3: The dynamic programming counting algorithm on data from Example 6

It is clear that the dynamic programming runs in polynomial time, thus completing the proof of Theorem 2.

□

5 Applications to RESTORED

5.1 Explaining RESTORED behavior

The result we just proved allows the reinterpretation of results in [8, 6]. In that paper it was shown experimentally that RESTOREDis able to recover several measures of quality of service, among them the following metric [9]. For ease of presentation, our version of the metric is presented in the case of permutations (i.e. sequences with no repeats or packet losses). See [9] for the general definition:

Definition 7 Reorder Density (RD). *Consider an implementation-dependent parameter DT that is a positive integer or ∞. Given a permutation π we define the reorder density of π as the distribution of* displacements *$\pi[i] - i$, restricted to those displacements in the range $[-DT, DT]$.*

In general it is simply unreasonable to expect to perfectly recover an arbitrary measure W of reordering. The reason is that RESTORED replaces an input sequence A with a sequence B from the preimage $\mathcal{F}^{-1}(\mathcal{F}(A))$. unless W is mostly constant on set $\mathcal{F}^{-1}(\mathcal{F}(A))$ one cannot expect that the two values of W for the original and reconstructed sequence will be identical.

The above discussion motivated the following definition from [6] (stated here for simplicity for $\equiv_{FB}$):

Definition 8 *A metric M is* consistent with respect to $\equiv_{FB}$ *if for any two ID sequences A and B, $A \equiv_{FB} B \implies M(A) = M(B)$. In other words, a consistent measure M takes equal values on equivalent sequences.*

Example 6 *By equation (1), every measure defined in terms of the time series of parameter $RcwWindow$ (e.g. the average value of this parameter) is consistent with respect to $\equiv_{FB}$.*

In particular, since RESTORED (in the form used in [8, 6]) guarantees that, on sequence A it will reconstruct a sequence $R(A)$ such that $R(A) \equiv_{FB}$

A, it is not really that surprising that RESTORED should be able to capture any metric consistent with respect to $\equiv_{FB}$.

The reason that we found the experimental results from [8] somewhat paradoxical is that RD is an example of an *inconsistent measure* (at least on "theoretical" examples) according to the terminology of Definition 8.

Observation 2 *If* $A = (4\ 3\ 2\ 1)$ *and* $B = (4\ 2\ 3\ 1)$ *then the distributions of displacements are* $D(A) = \begin{pmatrix} \textit{-3} & \textit{-1} & \textit{1} & \textit{3} \\ \textit{1/4} & \textit{1/4} & \textit{1/4} & \textit{1/4} \end{pmatrix}$ *and* $D(B) = \begin{pmatrix} \textit{-3} & \textit{0} & \textit{3} \\ \textit{1/4} & \textit{1/2} & \textit{1/4} \end{pmatrix}$*, respectively. It is easy to see that, no matter how we set the parameter* DT *to either a positive integer or* ∞*, the truncated versions of distributions* $D(A), D(B)$ *are going to be different. Thus* $A \equiv_{FB} (B)$ *but* $D(A) \neq D(B)$*, which means that measure RD is inconsistent independently of the value of threshold parameter* DT*.*

However, Theorem 1 forces us to reevaluate the real-life applicability of this simple example: since the vast majority of traces used in the experimental benchmarking from [8] had $SUS \leq 3$ the "theoretical" inconsistency of RD from Observation 2 is less stringent "in practice": in particular since no two permutations map to the same reordering pattern and, crucially, *repeat packets are discarded before computing RD* [9], it is no longer that surprising that RD could be largely consistent "in real life", even though theoretically inconsistent.

5.2 Redesigning RESTORED

As hinted in [8] (and witnessed by our subsequent papers [7], [6]) mapping $\mathcal{M}$ was by no means the only many-to-one mapping we tried in developing RESTORED. Indeed, in these subsequent papers we mathematically analyzed a version of the function FB, that only takes into account effectively buffered packets in the definition of buffer size, and showed that it has good properties.

More generally, the conditions we want any good compression methods for reordering patterns $\mathcal{F}$ to satisfy are the following:

1. Mapping $\mathcal{F}$ has to be easily computable.
2. Mapping $\mathcal{F}$ should have many preimages of a given coded sequence, *especially those arising from "real-life" observations.*

3. Mapping $\mathcal{F}$ has to be invertible in polynomial time: there has to be a polynomial time algorithm that, given a coded sequence X computes a preimage $A \in \mathcal{F}^{-1}(X)$.

4. A stronger form of condition 3. requires to be able to sample in polynomial time (almost) uniformly from the set of preimages $A \in \mathcal{F}^{-1}(X)$ of a coded sequence X.

The first condition ensures good compression. The second one ensures computational tractability. Condition 4 (and its weaker version 3) is used in the regeneration phase of RESTORED .

In [5] we have shown that mapping FB satisfies condition 3. Stronger condition 4 is also true (but this was not explicitly stated /shown in that paper there), since sampling matchings in a bipartite graph has a fully polynomial randomized approximation scheme. The sampling algorithm is too complicated to be of practical use, and was never implemented as part of RESTORED .

For mapping $\overline{FB}$ (that somewhat differs from FB on sequences with repeat packets), the following simple Corollary of Theorem 2 shows that the situation is much better: a simple sampling algorithm can be obtained by adapting the algorithm used to count preimages:

Corollary 3 *There exists an algorithm that, given a sequence of integers P generates a random permutation with repeats $A \in \overline{M}^{-1}(P)$ and $SUS(A) \leq 3$.*

Proof: In the proof of Theorem 2 we counted permutations with repeats with the desired properties by mapping them bijectively to nondecreasing partial functions with a special structure.

The problem of counting such nondecreasing partial functions is easily seen to be *self-reducible*: if i is the smallest index with P_i, such a partial function f is in one of the following situations:

- f is undefined at i, and in bijective correspondence with a partial function on the (smaller) domain obtained by eliminating i. We can use the previous algorithm to compute $P_{\emptyset}$, the number of such partial functions.

- f assumes a value $\lambda \in C_i$. In this case the restriction of f to the domain obtained by eliminating i is a similar object (a nondecreasing partial function with a similar special structure). For every λ we can use the previous algorithm to compute P_λ, the number of such partial functions.

Now it's easy to generate a random nondecreasing partial function f:

1. f will be undefined at i with probability $\frac{P_\emptyset}{P_\emptyset+\sum_\lambda P_\lambda}$.
2. f will assume a value $\lambda \in C_i$ with probability $\frac{P_\lambda}{P_\emptyset+\sum_\lambda P_\lambda}$.
3. Quantities $P_\emptyset$, P_λ can be efficiently computed using the dynamic programming algorithm from the proof of Theorem 2.

□

In conclusion, using mapping $\overline{FB}$ is preferred to mapping FB for ease of sampling reasons. This was **not** clear at the time we did the experiments[2] but is now in retrospect.

6 Conclusion and Acknowledgment

In this paper we proved two related combinatorial results, Theorems 1 and 2, inspired by our earlier experimental work [8]. The first result has been shown to illuminate some experimental behavior in [8]; the second one can be used to improve the regeneration method from that paper.

I thank Anders Hansson for useful discussions. This work has been supported in part by CNCS IDEI Grant PN-II-ID-PCE-2011-3-0981 "Structure and computational difficulty in combinatorial optimization: an interdisciplinary approach".

References

[1] D. Aldous and P. Diaconis. Longest increasing subsequences: from patience sorting to the Baik-Deift-Johansson theorem. *Bulletin of the American Mathematical Society* 36.4 (1999): 413-432. doi:10.1090/S0273-0979-99-00796-X

[2] in fact we experimented with both FB and $\overline{FB}$, but the experimental results were very similar, so we saw no reason to present both versions

[2] J. Bennett, C. Partridge, and N. Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Networking*, 7(6), 1999, pp. 789-798. doi: 10.1109/90.811445

[3] J. Bellardo and S. Savage. Measuring packet reordering. In *Proc. 2nd ACM Workshop on Internet Measurement*, pages 97–105, 2002. doi: 10.1145/637201.637216

[4] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992. doi:10.1145/146370.146381

[5] A. Hansson, G. Istrate, and S. Kasiviswanathan. Combinatorics of TCP reordering. *Journal of Combinatorial Optimization*, 12(1–2):57–70, 2006. doi: 10.1007/s10878-006-8904-0

[6] A. Hansson, G. Istrate, and G. Yan. Packet reordering metrics: Some methodological considerations. In *Proceedings of the Second International Conference on Networking and Services (ICNS'06)*. I.E.E.E. Computer Society Press, 2006. doi:10.1109/ICNS.2006.80

[7] G. Istrate and A. Hansson. Counting preimages of TCP reordering patterns. *Discrete Applied Mathematics*, 156(17):3187–3193, 2008. doi: 10.1016/j.dam.2008.05.011

[8] 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. doi: 10.1007/11753810_11

[9] A. P. Jayasumana, N. M. Piratla, A. A. Bare, T. Banka, and R. Whitner. Improved packet reordering metrics. RFC 5236, 2008.

[10] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet, Second Edition.* Addison Wesley, 2003.

[11] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network* 16(5), pp. 28–36, 2002. doi:10.1109/MNET.2002.1035115

[12] C. Levcopoulos and O. Petersson. Sorting shuffled monotone sequences. *Information and Computation*, 112(1):37–50, 1994.

[13] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, and J. Perser. Packet reordering metric for ippm. IETF RFC 4737, 2006.

[14] N. M. Piratla, A. P. Jayasumana, and A. A. Bare. Reorder Density (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. doi: 10.1007/11422778_7

[15] N. Piratla, A. Jayasumana, A. Bare, and T. Banka. Reorder buffer-occupancy density and its applications for measurement and evaluation of packet reordering. *Computer Communications*, 30(9):1980–1993, 2007. doi: 10.1016/j.comcom.2007.03.001

[16] W.R. Stevens. *TCP/IP Illustrated*. Addison Wesley, 1994.