Counting Preimages of TCP Reordering Patterns

Anders Hansson

Discrete Simulation Sciences (CCS-5),
Los Alamos National Laboratory
P.O. Box 1663, Mail Stop M997,
Los Alamos, NM 87545, USA

Gabriel Istrate *

eAustria Research Institute,
Bd. Corneliu Coposu no. 4, cam. 045B
Timişoara, RO-300223, Romania

Abstract

Packet reordering is an important property of network traffic that should be captured by analytical models of the Transmission Control Protocol (TCP). We study a combinatorial problem motivated by RESTORED [1], a TCP modeling methodology that incorporates information about packet dynamics. A significant component of this model is a many-to-one mapping B that transforms sequences of packet IDs into *buffer sequences* in a manner that is compatible with TCP semantics. We show that the following hold:

- There exists a linear time algorithm that, given a buffer sequence W of length n, decides whether there exists a permutation A of $\{1, 2, \ldots, n\}$ such that $A \in B^{-1}(W)$ (and constructs such a permutation, when it exists).
- The problem of counting the number of permutations in $B^{-1}(W)$ has a polynomial time algorithm.
- We also show how to extend these results to sequences of IDs that contain repeated packets.

Key words: TCP, packet reordering, matchings.

* Corresponding author
Email addresses: hansson@lanl.gov (Anders Hansson),
gabrielistrate@acm.org (Gabriel Istrate).

1 Introduction

Consider a sequence of *TCP packets*, identified by their integer IDs, as handled by their *receiver*. The receiver must forward the packet sequence to an *application*, subject to respecting *packet sequence integrity*. That is, at every moment the IDs of packets forwarded to the application must form a contiguous sequence $1, 2, \ldots, m$, for some $m \geq 1$. Packets can arrive out-of-order and thus need to be buffered. Several copies of a packet can arrive, but only one copy of a given 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 as soon as possible, i.e., as soon as the packet sequence integrity constraint is satisfied.

A given sequence $A = (A_1, \ldots, A_n)$ of packet IDs yields a corresponding sequence $B(A) = (B_{A,1}, \ldots, B_{A,n})$ representing the evolution of the buffer size. In this paper we are interested in the following problem: given a sequence of positive integers W, what is the complexity of

(1) Deciding whether there exists a permutation A with $W = B(A)$?
(2) Counting the number of permutations in the set $B^{-1}(W)$?

2 Motivation

The problem we described in the introduction arises in the context of analytical modeling of TCP dynamics. Therefore, the reader only interested in the combinatorial aspects of the problem can focus on the remaining sections. This section explains in detail the motivation for the problem.

While a lot of attention has been given to modeling the temporal aspects of TCP traffic (see e.g. Jaiswal *et al*. [2]), the dynamics of packet IDs has not received the same attention. As Bennett *et al*. [3] have shown, packet reordering is more widespread than originally believed, and is increasingly becoming so, due to technological advances such as link striping and mobile communications. Packet reordering has many severe effects on overall traffic characteristics, hence it is an important component of TCP dynamics (we refer the reader to [3] for further discussion).

Paper [1] introduced RESTORED, a methodology for semantic compression and regeneration of large TCP traces. RESTORED is based on the following observation: 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. A received packet

that allows the buffer to flush is called a *pivot packet*. All packets appearing in order are trivially pivots. RESTORED divides the received sequence into segments, bounded by pivot packets. Segments correspond to one of two *phases*:

- An *ordered phase*, in which no reordering is present, thus there is no need for buffering.
- An *unordered phase*, in which there is reordering and buffering. [1] Each occurrence of this phase ends when a pivot packet is received.

RESTORED preserves packet reordering properties of TCP traffic, up to a notion of semantic equivalence of packet traces. This notion is called *behavioral equivalence* and can be motivated as follows:

Definition 1 *Let ACK_i be defined as the smallest integer that does not appear among the first i packet IDs (also, define $ACK_0 = 1$). Parameter ACK_i is called* the acknowledgement (ACK) at stage i.

The previous definition relies on the simplifying assumption that in the implementation of TCP each received packet is ACKed, and that value ACK_i is the only information carried by the ACK packet. Of course, real-life acknowledgment policies of TCP can be more complicated [4].

Consider now the following two packet ID sequences: 4 2 3 1 and 4 3 2 1.

Both these sequences trigger identical ACK responses, namely 1 1 1 5, i.e., we arrive at the following two mappings:

$$\begin{aligned} &4\ 2\ 3\ 1 \rightarrow 1\ 1\ 1\ 5, \\ &4\ 3\ 2\ 1 \rightarrow 1\ 1\ 1\ 5. \end{aligned} \tag{1}$$

Since TCP is a *receiver-driven* protocol, assuming identical network conditions, and discounting possible differences in the value of the congestion window at the beginning of the sequences, *the two ID sequences trigger identical responses from the receiver, and should thus be regarded as indistinguishable from the standpoint of TCP dynamics*.

Definition 2 *Two sequences of packets P and Q are* behaviorally equivalent *(written $P \equiv_{beh} Q$) if they lead to the same sequences of ACKs.*

In practice one might want a notion of equivalence that is even more restrictive than behavioral equivalence. This was, for instance, the case of RESTORED. Its original motivation was to provide a way to compress TCP traces and estimate various

[1] A technical assumption we will employ is that duplicates of packets that have already been uploaded to the application layer are discarded. This is a sensible assumption, given TCP behavior.

measures of quality of service of the original traces by reconstructing "compatible" sequences. Many measures of packet reordering have been proposed in the networking literature [5–7]. Given such a measure M, one way to guarantee that sequences produced by RESTORED resemble the original sequence with respect to measure M is:

(1) Identify an equivalence notion of ID sequences $\equiv$ such that M *is consistent with respect to* $\equiv$, that is

$$(\forall A, B)\text{: } (A \equiv B) \Rightarrow (M(A) = M(B)). \tag{2}$$

(2) Make sure that for any sequence A, the sequence $R(A)$ regenerated by RESTORED satisfies $R(A) \equiv A$.

(See also [8] for more discussion and clarification). Behavioral equivalence might be too coarse (as an equivalence relation) to guarantee consistency of many reordering metrics and, thus, needs to be refined. In a companion paper [9] we have considered such an equivalence notion, based on the following notion of buffer size:

Definition 3 *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.*

$$\text{FB}(A_i) = H_i - L_i. \tag{3}$$

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 $\text{FB}(P)$ *associated with a sequence P of packet IDs is simply a time-series of* FB *values computed after each packet has been received.*

Two sequences of packet IDs P and Q are FB equivalent *(written $P \equiv_{\text{FB}} Q$) if* $\text{FB}(P) = \text{FB}(Q)$.

This definition is directly related to the semantics of TCP, since it preserves quantities such as the size of the AdvertisedWindow (see [10]). Inverting the mapping FB can be done in polynomial time [9]. However, the complexity of computing the cardinality of the preimage $\text{FB}^{-1}(W)$ was left open, and was only solved in two special cases.

In this paper, we use a different notion, introduced below, for which more precise results can be obtained.

Definition 4 *Buffer size is the smallest size of a buffer that can store all out-of-order packets. Two sequences of packets P and Q are* buffer equivalent *(written*

$P \equiv_{buf} Q$) if $B(P) = B(Q)$, that is the sequences of buffer sizes associated with receiving P and Q are identical.

From a combinatorial perspective, buffer equivalence is more natural than FB equivalence. Its relation with behavioral equivalence is, however, slightly more complicated:

(1) Buffer equivalence is *not* a refinement of behavioral equivalence in general. Indeed, sequences of packet IDs 2 3 3 1 and 3 4 1 2 are buffer equivalent (they both map to sequence 1 2 2 0) but *not* behaviorally equivalent (the ACKs are 1 1 1 4 and 1 1 2 5, respectively). This stands in contrast to FB equivalence which is indeed [9] a refinement of behavioral equivalence.
(2) Buffer equivalence refines behavioral equivalence when restricted to *permutations* (sequences with no repeats or lost packets). For a formal statement and proof of this claim see Proposition 1 below.
(3) Finally, buffer equivalence is incomparable (as an equivalence notion) with FB equivalence [8].

On the other hand there exist reordering metrics M defined in the networking literature (e.g. *reorder buffer density* [11]) with the following properties:

(1) M only depends on packets received for the first time, and not on repeat packets.
(2) M is inconsistent with respect to FB equivalence but consistent with respect to buffer equivalence (metrics with opposite consistency properties exist as well; see [8] for further details).

The recovery of such metrics via the argument described in equation (2) motivates the problem we study in this note: inverting the many-to-one map B and counting the size of its preimage. Results for map B are slightly stronger than those proven in [9] for map FB. Namely, computing the cardinality of the preimage of map B, as well as returning one element from the preimage can be done in polynomial time (even linear time for the latter problem).

3 Preliminaries

We will use notation $x \dot{-} y = max\{x - y, 0\}$.

We employ standard graph theoretic notions throughout. In this paper, graphs are always bipartite and undirected. Denote by $d(v)$ the degree of vertex v and by $N(v)$ the set of neighbors of v.

Definition 5 *A bipartite graph $G = (V_1, V_2, E)$ is* doubly convex *if there exist permutations π_1, π_2 of vertex sets V_1, V_2, respectively, such that for every $i \in \{1, 2\}$*

and every vertex $v \in V_i$ the set of vertices w that are adjacent to v forms an interval (i.e. a set of consecutive nodes) of $\pi_{3-i}(V_{3-i})$.

Definition 6 *A sequence of IDs W is a* valid buffer pattern *if there exists a permutation A of $\{1, 2, \ldots, |W|\}$ such that $B(A) = W$.*

Note that any valid buffer pattern W necessarily ends in a zero, since for $A \in B^{-1}(W)$ all packets in A can be passed to the application layer when the last packet in A is received. Also, without loss of generality, one can assume that *the only* position in a valid buffer pattern that is equal to zero is the last one, since one can decompose a given pattern W into disjoint segments, bounded by those positions equal to zero (where the buffer, therefore, gets flushed). To each such segment one can associate a permutation of a contiguous set of IDs.

4 Inverting Buffer Sequences

Our main result is

Theorem 4.1 *The following are true:*

(1) There is an algorithm that, given an encoding $W = W_1 \# W_2 \# \ldots \# W_n \# \#$ of a sequence of positive integers as input (the W_i's are integers in binary notation and $\#$ is a new symbol) decides in time $O(|W|)$ whether W is a valid buffer pattern, and if this is the case constructs a permutation A such that $A = B^{-1}(W)$.
(2) Counting the cardinality of the set of permutations in the preimage $B^{-1}(W)$ can be done in polynomial time.

Proof.

We will provide, in essence, a reduction of the problem above to the problem of finding a maximum matching in a special class of doubly convex bipartite graphs [12]. The complexity of this problem is linear in the number of vertices of the graph [12]. Since the size of the bipartite graph that is created by reduction is linear, the overall complexity of the problem is linear.

A valid buffer sequence consists of positive integers, with the exception of the last entry, which is zero. Any two consecutive values of the buffer sequence W_i and W_{i+1} can only be in one of the following situations:

(1) $W_i = W_{i-1} + 1$. This situation corresponds to one new out-of-order packet being received at stage i. This holds for $i = 1$ as well, if we let $W_0 = 0$.
(2) $W_i < W_{i-1}$. This situation corresponds to the newly received packet causing a non-empty portion of the buffer to be flushed. In particular the ID of the

received packet can be inferred at this stage, and is equal to the smallest index of a packet not received so far.

(3) $W_i = W_{i-1}$. This situation corresponds to the packet received at this stage being the first packet not previously received. Receiving this packet does not cause any other packet to be sent to the application layer.

If the input sequence fails to satisfy these conditions (for instance if there exists an index i with $W_i - W_{i-1} > 1$) then the set of permutations in $B^{-1}(W)$ is empty. Otherwise, let S_1, S_2, S_3 be the set of indices corresponding to the three cases listed above.

During the course of the algorithm we will keep track of the value ACK_i, computed assuming that W is a valid buffer pattern. Initially $ACK_0 = 1$. We have the following recurrence relations (mirroring the three cases described above):

(1) The newly received packet is out-of-order. Thus, it does not change the value of parameter ACK. Therefore

$$ACK_i = ACK_{i-1}. \tag{4}$$

(2) The newly received packet has ID ACK_{i-1}. In addition, it makes the buffer shrink in size from W_{i-1} to W_i, which means that

$$ACK_i = ACK_{i-1} + W_{i-1} - W_i + 1. \tag{5}$$

(3) The newly received packet has index ACK_{i-1} and does not cause the buffer to shrink any more. Therefore

$$ACK_i = ACK_{i-1} + 1. \tag{6}$$

For all indices $i \in S_2 \cup S_3$, the index of the received packet is uniquely determined, and equal to ACK_{i-1}.

We will now create a bipartite graph $G = (V_1, V_2, E)$. Nodes in V_1 correspond to stage indices $i \in \{1, \ldots n\}$. Nodes in V_2 will correspond to packet IDs. First, let $V_1 = S_1$, and let $V_2 = \{1, \ldots, n\} \setminus \{ACK_{i-1} | \, i \in S_2 \cup S_3\}$. Clearly $|V_2| = n - |S_2 \cup S_3| = |S_1| = |V_1|$. Second, given node $i \in V_1$, add edges to all vertices $j \in V_2$ such that $j > ACK_i$.

With this definition we have:

Lemma 4.1 *Permutations from the set $B^{-1}(W)$ are in bijective correspondence with elements of $MATCH(G)$, the set of all perfect matchings in G. In particular $B^{-1}(W) \neq \emptyset$ if and only if G has a perfect matching.*

Proof.

Each permutation can be seen as a set of pairs (i, j). By the previous discussion, the set of acknowledgements $\{ACK_i\}_{i\geq 0}$ is the same for any permutation in $B^{-1}(W)$. Moreover, for all $\sigma \in B^{-1}(W)$ and index $i \in S_2 \cup S_3$, $\sigma[i] = ACK_{i-1}$. Also, for such a permutation σ, by definition of graph G it is easy to see that all pairs $(i, \sigma[i])$ with $i \in S_1$ are edges in G. Hence σ corresponds to a perfect matching in G.

Conversely, every perfect matching M in G naturally corresponds to a sequence of pairs, that can be completed (by adding all pairs (i, ACK_{i-1}) for all values i not in V_1) to a mapping A defined on $\{1, \ldots, n\}$. A is actually a permutation. Indeed, the values of parameter ACK_i, $i \in S_2 \cup S_3$, are all different, and are not included in V_2. It follows that A maps n numbers onto n different numbers, hence it is a bijection.

To show that $A \in B^{-1}(W)$, assume that this was not the case, and let i be the smallest index such that $B_{A,i} \neq W_i$. Thus $B_{A,i-1} = W_{i-1}$ where, by convention $B_{A,0} = 0$.

Case 1 $B_{A,i} = B_{A,i-1} + 1$. Since $W_i \neq B_{A,i}$ and $W_i - W_{i-1} \leq 1$, the only possible alternatives are $W_i = W_{i-1}$ or $W_i < W_{i-1}$. But then index i is not in V_1 and is matched in A to integer ACK_{i-1}. This contradicts the assumption that $B_{A,i} = B_{A,i-1} + 1$, since the packet with ID ACK_{i-1} is the first not received in the first $i - 1$ phases, and can thus be uploaded at stage i. The contradiction comes from our assumption that sequences $B(A)$ and W are different.

Similar arguments can be applied in the two remaining cases for the evolution of sequence $B(A)$, and the conclusion of the argument is that $A \in B^{-1}(W)$.

□

Lemma 4.2 *Let $a_1 \geq a_2 \geq \ldots \geq a_m$ be the number of ones on the first, second, $\ldots, m$'th row of M_G, the adjacency matrix of G (call $(a_1, \ldots, a_m)$* the type *of M_G). Then we have*

(1) G has a perfect matching if and only if for all $i = 1, \ldots, m$, $a_i \geq m + 1 - i$. When this condition holds, a perfect matching in G can be constructed by taking elements on the diagonal of M_G.
(2) The number of matchings in G is given by

$$|MATCH(G)| = a_m(a_{m-1}\dot{-}1)(a_{m-2}\dot{-}2)\cdot\ldots\cdot(a_1\dot{-}(m-1)). \qquad (7)$$

Proof.

Denote the cardinality of set $MATCH(G)$ by $\Gamma(a_1, \ldots, a_m)$ (to highlight its dependency on parameters $a_1, \ldots, a_m$). Expand the permanent across the last row. Since $a_1, \ldots, a_{m-1}$ are all greater or equal to a_m, it follows that $\Gamma(a_1, \ldots, a_m)$ is the sum of the permanent of a_m minors, all of them of type $(a_1\dot{-}1, \ldots, a_{m-1}\dot{-}1)$. Thus, $\Gamma(a_1, \ldots, a_m) = a_m \cdot \Gamma(a_1\dot{-}1, \ldots, a_{m-1}\dot{-}1)$, and formula (7) immediately follows by noting that, for all $i \geq 1$, $(a\dot{-}(i-1))\dot{-}1 = a\dot{-}i$.

□

We now complete the proof of Theorem 4.1.

(1) Algorithm TwoStageGreedy in Figure 1 produces a perfect matching (if it exists). Its correctness follows from the recurrence relations for parameter ACK_i and Lemma 4.2 (2). With a little care the algorithm can be implemented in $O(|W|)$ time (using $O(|W|)$ additional memory) as follows:
   (a) We use two buffers, P and Q, each for $\lceil \log_2(n) \rceil$ integers. They are intended to hold numbers W_i and W_{i-1}. The for-loop can be implemented by simply scanning the input from left to right, copying the correct information into buffers P and Q. Only two buffers are needed, provided we keep switching roles of P and Q (they will alternately keep the last value W_i). All test conditions in the algorithm involving these numbers, as well as computing $W_i - W_{i-1}$, will be performed using buffers P and Q, and can be accomplished by scanning these buffers C times, for some fixed constant C.
   (b) The final for loop can be implemented in linear time by scanning buffer σ from left to right, using an additional counter for the value of index j.
   (c) In the algorithm we keep incrementing several counters. The problem of incrementing counters is well-known to have linear time algorithms via amortized analysis [13].

(2) Computing $|MATCH(G)|$ using formula (7) can be done in polynomial time as follows:
   (a) First, there is a *linear* time algorithm that, given input W, outputs the list of numbers $a_1, \ldots, a_m$.
   (b) Given these numbers, computing $|MATCH(G)|$ can be accomplished in time polynomial in $m + \lceil \log_2 |MATCH(G)| \rceil$ by the brute-force product computation in (7). Since $|MATCH(G)| \leq n!$ (simply because matchings correspond to permutations), it follows by Stirling's approximation that $\lceil \log_2 |MATCH(G)| \rceil = O(n \log n)$. Thus, the running time is polynomial in $|W|$.

□

The proof of Theorem 4.1 also implies that buffer equivalence is a refinement of behavioral equivalence for permutations:

Proposition 1 *Let P and Q be two* permutations *such that $P \equiv_{buf} Q$. Then $P \equiv_{beh} Q$.*

Proof.

Equations (4)–(6) show that the value of parameter ACK_i can be recovered directly

Algorithm TwoStageGreedy(W)

INPUT: a vector $W = W_1\#W_2\#\ldots\#W_n\#\#$ of nonnegative integers.

Let σ be a vector of n numbers of length $\lceil \log_2(n) \rceil$, initially all zero.
Let ACK be a vector of $n+1$ numbers of length $\lceil \log_2(n) \rceil$, initially all zero, with the exception of $ACK_0 = 1$.
Let $chosen$ be an n-bit vector, with all positions initially zero.
Let $W_0 = 0$.

for $i = 1$ to n
  if $(W_i - W_{i-1} > 1) \vee ((i < n) \wedge (W_i = 0)) \vee ((i = n) \wedge (W_i \neq 0))$
    reject
  else
    if $(W_i = W_{i-1})$
      let $\sigma[i] = ACK_{i-1}$;
      let $chosen[ACK_{i-1}] = 1$;
      let $ACK_i = ACK_{i-1} + 1$;
    else
      if $(W_i < W_{i-1})$
        let $\sigma[i] = ACK_{i-1}$;
        let $chosen[ACK_{i-1}] = 1$;
        let $ACK_i = ACK_{i-1} + W_i - W_{i-1} + 1$;
      else
        /* $W_i = W_{i-1} + 1$ */
        let $ACK_i = ACK_{i-1}$;

for $i = 1$ to n
  if $(\sigma[i] = 0)$
    let $\sigma[i]$ = the first $j > ACK_{i-1} + 1$ with $chosen[j] = 0$;

return σ.

Fig. 1. Algorithm for inverting buffer sequences

from the buffer sizes. Since P and Q are buffer equivalent, they have identical buffer size sequences and, consequently, identical sequences of parameter ACK_i. But it is easy to see that the sequence of packet IDs (more precisely the corresponding sequence of byte IDs) ACKed by the TCP protocol in the case of simple consecutive ACKs is precisely ACK_i. Therefore P and Q are behaviorally equivalent.

□

5 Reconstructing Packet Sequences with Repeats

Buffer equivalence is not a refinement of behavioral equivalence in the presence of repeats. The reason is that one cannot distinguish between the case when the newly received packet is a repeat and Case 3 in the proof of Theorem 4.1 (in both cases the buffer size stays the same). However, for a repeat packet the value of the ACK parameter does not change, while for a packet in Case 3 the value of the ACK parameter increases by one.

One can modify the notion of buffer equivalence (in a somewhat artificial way) to incorporate information whether the received packet is a repeat or not. For instance, one can define $B_{A,i}$ to be *minus* the buffer size when the i'th received packet is a repeat. Denote this new mapping by $\overline{B}$.

Definition 7 *Two sequences of packets P and Q are* modified buffer equivalent *(written $P \equiv_{\overline{buf}} Q$) if $\overline{B}(P) = \overline{B}(Q)$.*

The analog of Theorem 4.1 for mapping $\overline{B}$ is

Theorem 5.1 *Let $W = W_1 \# W_2 \# \ldots \# W_n \# \#$ be a sequence of integers.*

Deciding whether W is a valid buffer pattern, and in this case constructing an ID sequence A such that $A = \overline{B}^{-1}(W)$, can be done in linear time. Counting the cardinality of the preimage $\overline{B}^{-1}(W)$ can be done in polynomial time.

We only outline the proof, since it is very similar to that of Theorem 4.1. Given our use of negative numbers in the encoding, we no longer have the positivity constraint for elements of the candidate sequence W. However, we still require that only the last element be zero.

The construction of graph G is identical to that in the previous case, since in all stages in V_1 we can guarantee that a new packet is received. However, we do *not* have a parsimonious reduction of ID sequences to perfect matchings, since repeat packets can complete a matching in G in more than one way.

A polynomial-time counting algorithm exists, nevertheless, since we can complement Lemma 4.2 with

Lemma 5.1 *We have*

$$|\overline{B}^{-1}(W)| = |MATCH(G)| \times \left(\prod_{i \in R} |W_i| \right), \tag{8}$$

where $MATCH(G)$ is the set of all perfect matchings in G, and $R = \{i \,|W_i < 0\}$, i.e. the set of stages in which a repeat packet arrives. In particular $\overline{B}^{-1}(W) \neq \emptyset$ if

and only if G has a perfect matching.

Also, the construction shows that modified buffer equivalence *is* a refinement of behavioral equivalence. Indeed, from the sequence of modified buffer sizes one can uniquely reconstruct the sequence of acknowledgments. The proof then proceeds just as the proof of Proposition 1.

6 Acknowledgments

We acknowledge the anonymous referees of this paper for very useful references and suggestions.

This work has been supported by the U.S. Department of Energy under contracts W-705-ENG-36 and DE-AC52-06NA25396.

References

[1] G. Istrate, A. Hansson, S. Thulasidasan, M. Marathe, C. Barrett, Semantic compression of TCP traces, in: Proceedings of the IFIP NETWORKING Conference, F. Boavida et al. (editors), Vol. 3976 of Lecture Notes in Computer Science, Springer Verlag, 2006, pp. 123–135.

[2] S. Jaiswal, G. Iannacone, C. Diot, J. Kurose, D. Towsley, Inferring TCP connection characteristics through passive measurements, in INFOCOM 2004, Proceedings of the Twenty-Third Joint Conference of the IEEE Computer and Communication Societies, vol.3, pp.1582-1592, 7-11 March 2004.

[3] J. C. R. Bennett, C. Partridge, N. Shectman, Packet reordering is not pathological network behavior, IEEE/ACM Transactions on Networking 7 (6) (1999) 789–798.

[4] W. Stevens, TCP/IP Illustrated, Vol.1: The Protocols, Addison Wesley, 1994.

[5] T. Banka, A. A. Bare, A. P. Jayasumana, Metrics for degree of reordering in packet sequences, in: Proceedings of the 27th IEEE Conference on Local Computer Networks, 2002, pp. 333–342.

[6] N. M. Piratla, A. P. Jayasumana, A. A. Bare, RD: A formal, comprehensive metric for packet reordering, in: Proceedings of the IFIP Networking Conference 2005, R. Boutaba et al. (editors), Vol. 3462 of Lecture Notes in Computer Science, Springer Verlag, 2005, pp. 78–89.

[7] A. Morton, L. Ciavattone, G. Ramachandran, S. Shalunov, J. Perser, Packet reordering metric for ippm, IETF draft, available from http://www.ietf.org/internet-drafts/draft-ietf-ippm-reordering-09.txt. Last accessed September 2005.

[8] A. Hansson, G. Istrate, G. Yan, Packet reordering metrics: Some methodological considerations, in: Proceedings of the Second International Conference on Networking and Services (ICNS'06), IEEE Computer Society Press, ISBN 0-7695-2622-5,2006.

[9] A. Hansson, G. Istrate, S. Kasiviswanathan, Combinatorics of TCP reordering, Journal of Combinatorial Optimization 12 (1–2) (2006) 57–70.

[10] L. Peterson, B. S. Davie, Computer Networks. A Systems Approach, 2nd Edition, Morgan Kauffman, San Francisco, CA, 2000.

[11] A. P. Jayasumana, N. M. Piratla, A. A. Bare, T. Banka, R. Whitner, J. McCollom, Reorder density and reorder buffer-occupancy density - metrics for packet reordering measurements, IETF draft, available from http://cnrl.colostate.edu/Reorder/draft-jayasumana-reorder-density-06.txt. Last accessed March 2006.

[12] W. Lipski, Jr., F. Preparata, Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems, Acta Informatica 15 (1981) 329–346.

[13] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms (Second Edition), M.I.T Press, 2001.