Vector Representations for the Analysis and Design of Distributed Controls

Michael Welzl
Computer Science Dept. / University of Innsbruck
Technikerstr. 25, A-6020 Innsbruck
Austria

ABSTRACT

We present novel methods of using vector diagrams of distributed control systems. A classic method is described and enhanced with four extensions, the benefit of three of them is demonstrated by examples. Our scenario is restricted to congestion control in computer networks, but we expect our results to be applicable to a much broader spectrum of control problems.

KEY WORDS: Distributed Controls, AIMD, TCP, Stability

1. INTRODUCTION

Congestion control in computer networks with a connectionless network layer has been an important research topic for many years. The global success of the Internet and its immense scalability have led to a common view of congestion control which is mainly based on the foundations of the TCP ("Transmission Control Protocol") mechanisms described in [1] and the "end to end argument" [2] – the fact that TCP basically realizes an AIMD ("Additive Increase, Multiplicative Decrease") algorithm is often mentioned as the primary reason for its stability and the stability of the Internet as a whole. This reasoning goes back to early work by Chiu and Jain [3], where AIMD is identified as a feasible synchronous control both algebraically and by means of vector representations: the system state transitions are regarded as a trajectory through an n-dimenstional vector space – in the case of two controls (which represent two users in a computer network), this vector space is two-dimensional and can be drawn and analyzed easily.

Such vector diagrams are a very simple means of explaining the convergence and stability properties of distributed controls. Moreover, they can help to give an intuitive idea of the quality of a control mechanism. Strict mathematical analysis, on the other hand, can become exceedingly complex and often leads to simplifications that appear to be less realistic than the two-user case[1]. Examples are [5] ("Under the assumption that queueing delays will eventually become small relative to propagation delays, we derive stability results for a fluid flow model of end-to-end Internet congestion control") and the contradictive cases [6] ("We assume in this paper that negative feedback received by sources is rare, and is proportional to the source rate.") and [7] ("We assume that the probability of having a packet loss within a window of *x* consecutively transmitted packets does not depend on their transmission rate.").

Since their appearance in [3], vector representations have become a common tool to explain the state transitions of congestion control mechanisms. Yet, the occurence of these diagrams appears to be limited to cases which are similar to the case examined by Chiu and Jain in two aspects[2]:

1. The feedback and control loop for all controls is synchronous, which means that each control sees the same feedback at any time.
2. The feedback is binary, so the information is limited to "there is congestion" vs. "there is no congestion". In the case of TCP, this feedback is traditionally implicit (based on packet loss). The recent addition of a congestion bit in the IP header, ECN, turns this into explicit feedback, but the meaning stays the same [10].

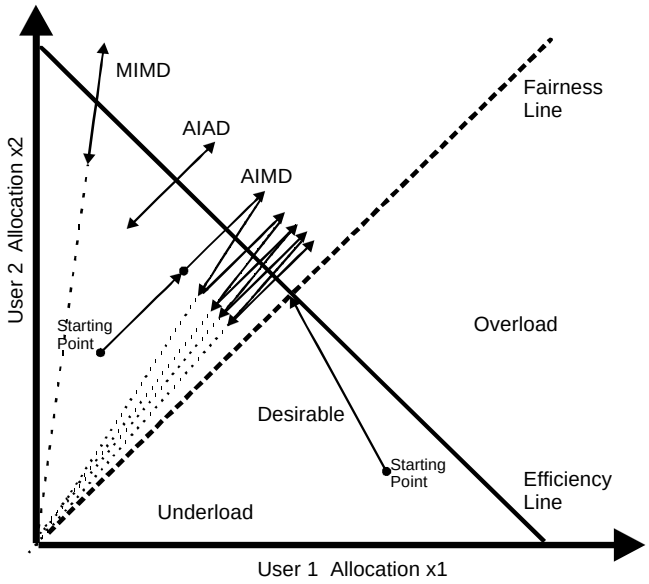
This paper is organized as follows: section 2 describes the traditional usage of vector representations as it is found in [3] and its successors. In section 3, we explain how the usage of vector diagrams can be extended; examples are given in section 4. Concluding remarks are presented in section 5.

2. BACKGROUND

Fig. 1 shows a vector diagram along with three of the controls studied in [3]: AIAD ("Additive Increase, Additive Decrease"), MIMD ("Multiplicative Increase, Multiplicative Decrease") and the already mentioned

[1] If, in the case of distributed linear controls, a system meets the conditions for Lyapunov stability, any superposition of such systems is stable [1] [4]. This fact can be used to inductively prove global stability.

[2] See [8] and [9] for examples.



AIMD. MIAD ("Multiplicative Increase, Additive Decrease") is missing because the other three controls suffice to explain the reasons that led to the choice of AIMD for TCP.
Each axis in the diagram represents a user in the network. Therefore, any point (x1, x2) represents a two-user allocation. The sum of the system load must not exceed a certain limit, which is represented by the Efficiency Line. The load is equal for all points on lines which are parallel to this line. One goal of the distributed control is to bring the system as close as possible to this line.

Another goal is to achieve fairness between the two users. A very basic definition of fairness (given a computer network where a user utililizes several resources, there are many other definitions which describe realizations of fairness according to different objectives) is that the system load consumed by user 1 should be the same as the load consumed by user 2. This is true for all points on the Fairness Line (note that the fairness is equal for all points on all lines which pass through the origin. Following [3], we therefore call any such line "Equi-Fairness Line"). The optimal point is the point of intersection between the Efficiency Line and the Fairness Line. The "Desirable" arrow in fig. 1 represents the optimal control: it moves to the optimal point and stays there (is stable). It is easy to see that this control is unrealistic for binary feedback: given that both users get the same feedback at any time, there is no way for user 1 to interpret the information "there is congestion" or "there is no congestion" differently than user 2 – but the Desirable vector has a negative x1 component and a positive x2 component.

Adding a constant positive or negative factor to a value at the same time corresponds to moving along at an angle of 45°. This effect is produced by AIAD: the system starts at a point underneath the Efficiency Line and moves upward at an angle 45°. The system ends up in Overload state (the state transition vector passes the Efficiency Line), which means that it now provides the feedback "there is congestion" to the users. Next, both users decrease their load by a constant factor, moving back along the same line. With AIAD, there is no way for the system to leave this line.

MIMD is similar, but a multiplication by a constant factor corresponds with moving along an Equi-Fairness Line. AIMD actually approaches the optimal point, but due to the binary nature of the feedback, the system cannot converge to a stable point but to an equilibrium – it will eventually fluctuate around the optimum. Note that these are by no means all possible controls: other examples are MIAD, controls with both an additive and multiplicative component [8] and nonlinear controls [9].

3. EXTENDED USE OF VECTOR DIAGRAMS

In the design process of a congestion control mechanism, it is always important to distinguish between *necessary* and *sufficient* conditions: while the explanations in the last section suffice to prove that AIAD and MIMD are not feasible, the explanations are not sufficient to prove the usefulness of AIMD due to the limitations of the model. In other words, it is necessary for a distributed control mechanism to converge towards the optimal point in the case of synchronous control loops. Mechanisms which do not fulfil this property are not feasible. But a mechanism must also work in the case of asynchronous control loops ("heterogeneous round-trip times" in terms of computer networks).

Asynchronous loop delay
There is no reason why the diagram should not be used to analyze the asynchronous case other than that the development of state transitions is less obvious. We therefore chose to write a simple simulator for the two user case. The system convergence should be independent of the initial state, so our simulator provides an interactive graphical user interface where the user can set the starting point and take a look at a trajectory by clicking the mouse in the diagram. This way, it is easy to test the asynchronous case: we assumed that the feedback will be given somewhere along the path from the sender to the receiver (after less than half a round trip time) and therefore chose to update the feedback for a user after a fourth of its loop delay. With this simplified model, we ignore queuing delay fluctuations; typical TCP effects such as round-trip time estimation and timeouts are not modelled either. We believe that despite these simplifications, our model suffices to get a basic idea of the nature of a control mechanism – and this is just what is needed during the design stage.

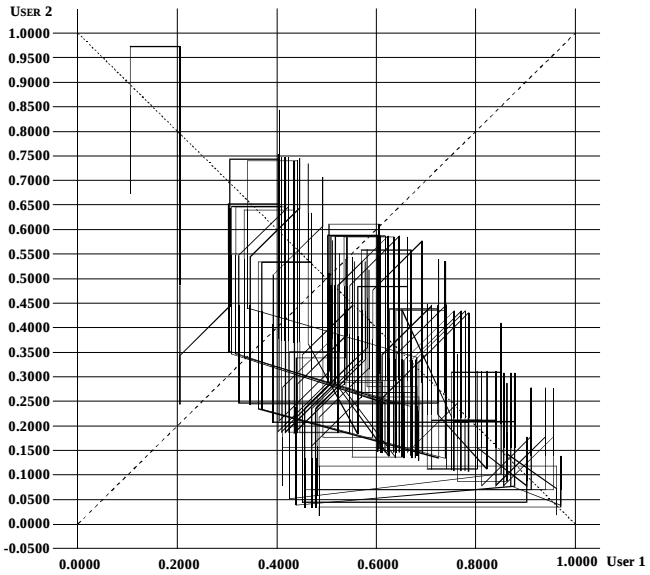


Fig. 2: AIMD with asynchronous loop delay

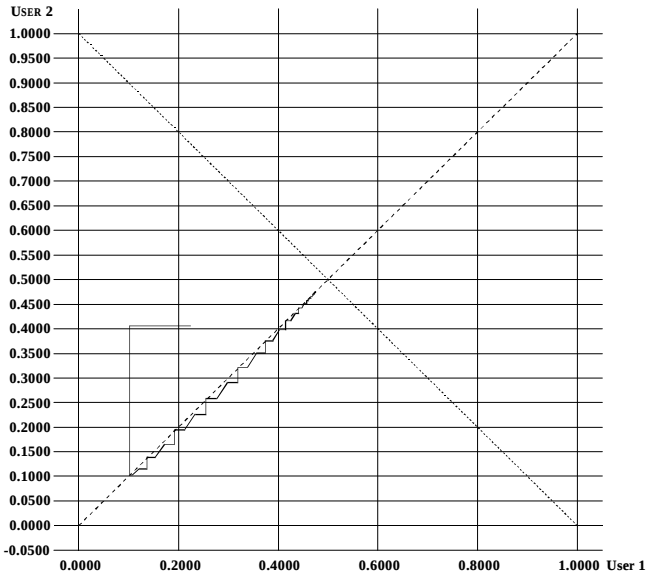


Fig. 3: The "CAPC" ATM ABR switch mechanism

Different feedback
While implicit and explicit binary feedback are currently used in the Internet, other kinds of feedback may be advantageous. ATM networks provide a service called "Available Bit Rate" (ABR), which comprises a framework for fair and optimal bandwidth sharing based on so-called "Explicit Rate Feedback". Here, ATM switches calculate an Explicit Rate value which is fed back to the users and used to tune the congestion control mechanism at the end points. The more complex nature of the feedback naturally makes the mathematical analysis of ABR mechanisms more difficult. Vector diagrams can be used to analyze the behaviour of an ATM ABR switch mechanism just as well as AIMD congestion control.

A greater degree of realism
The simplifications made for our simulator may or may not be sufficient to aid in the design of a new congestion control mechanism, but they are definitely not feasible for in-depth analysis of congestion control mechanisms. Mathematical analysis of the asynchronous case is difficult as the network can not be modelled by a single control system anymore. On the other hand, there are much more complex simulators which cover many specific networking effects and are capable of recording the system state. With this data, it is possible to plot the utilization of one user against the utilization of another user and thus get a vector representation which gives more insight into the convergence properties of the control than traditional bandwidth utilization plots.

Analysis of real network tests
It is a well known fact that simulations cannot replace real life tests, no matter how complex a simulator is. Apparently, there is no reason why real life measurements should not be examined in a vector diagram.

4. EXAMPLES

We now present some examples where extended use of vector diagrams illustrates features that would be hard to see by any other means. Apart from real network tests, which were omitted for brevity, all extensions from the previous section are utilized.

Asynchronous loop delay
Figure 2 shows an AIMD trajectory from our simulator where the round-trip-times of the users were chosen to be 7 and 2 time units, respectively. The additive-increase step was 0.1 and the multiplicative-decrease factor was 0.5, the simulation time was 175 time units (leading to 25 and 87 rate updates steps). Due to the heterogeneous nature of the loop delays, the results are hardly comparable with the AIMD trajectory in fig. 1 – in fact, from the trajectory shown in fig. 2, it is immediately clear that AIMD alone can show a very unstable behaviour. Thus, time invariant analytical models of AIMD fairness as in [6] can only describe the long-term average fairness but neglect the fact that the fairness appears to be oscillating.

Different feedback
Let us now consider the ATM ABR switch mechanism "CAPC" ("Congestion Avoidance with Proportional Control"). CAPC does not require any per-flow state in routers – it does not even keep track of the number of active flows, which is typically necessary in order to achieve fairness. The mechanism is described in [11] as follows:

```
rate = count / interval
delta = 1 - rate / r0
if delta >= 0
```

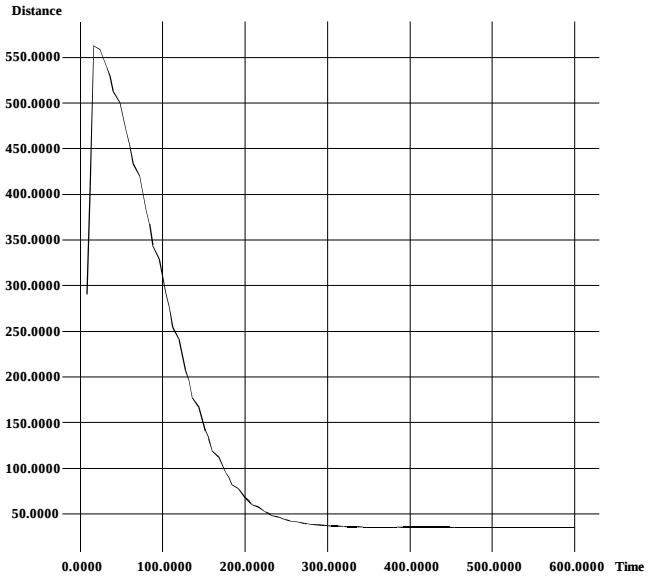


Fig. 4: CAPC distance from the optimal point

```
            ERX = 1 + delta * Rup
            ERX = min(ERX, ERU)
    else
            ERX = 1 + delta * Rdn
            ERX = max(ERF, ERF)
    ERS = ERS * ERX
```

count is a traffic counter which is increased upon each cell arrival; if it overflows, the above code is executed and the time since the last function call is given by interval. r0 is called the "target rate" – the system load is supposed to converge to r0. ERU and ERF are constants which serve as an upper and lower limit, respectively, and Rup and Rdn are constants which, as we will see later, represent a trade-off between the speed of convergence and the robustness of the system against load fluctuations and the magnitude of supported round-trip-times. ERS, initially set to "*some nice value, say R0 / 10*" [11], is eventually fed back to the end systems.

CAPC achieves convergence to efficiency by increasing the rate proportional to the amount by which the traffic is less than the target rate and vice versa. The additional scaling factors and the upper and lower limit ensure that the fluctuations diminish with each update step. We implemented CAPC in our simulator; an example trajectory is shown in fig. 3. It is plain to see that fairness is achieved by initially setting all sources to the same value. ERS is always updated multiplicatively; it is argued in [8] that multiplicative rate updates are normally regarded as "aggressive" while they should be seen as "proportional to the current load". As we have seen in section 2, multiplicative rate updates do not change the fairness in the synchronous case. Therefore, CAPC does not leave the fairness line if the round-trip-times are equal. In the case of fig. 3, the round-trip-times were 2 and 3, which accounts for minor perturberations, which are larger for more diverse round-trip-times.

Interactive experiments with the simulator have shown that increasing Rup and Rdn slightly leads to faster convergence but greater deviation from the fairness line. Increasing Rup and Rdn more drastically makes the mechanism unstable: even minor differences between the round-trip-times then prevent CAPC from converging. This seems to explain the fact that there are different proposed parameter values for LAN/MAN and WAN scenarios in [11].

Since we restricted our observations to the case of two users, we can use very simple additional analytic methods based on a two-dimensional vector space. For instance, for a given point (system state) *p*, the distance *d* from optimality *o* is just the Euclidian distance:

$$d = \sqrt{(o_{x1} - p_{x1})^2 + (o_{x2} - p_{x2})^2}$$

Figure 4 shows the development of the distance from optimality for the CAPC trajectory in fig. 3. From this figure, it is easy to see that CAPC approaches optimality in an asymptotic manner.

A greater degree of realism

The TCP protocol contains many more features than AIMD: it is a window-based protocol, which accounts for a certain stop-and-go behaviour (the sender is consecutively granted specific amounts of data and not just informed about a feasible transmission rate); this accounts for a stabilizing effect known as “ack-clocking”: basically, when the system reaches equilibrium, a TCP sender should only send n data packets into the “pipe” when n packets have arrived at the receiver. This is also called the “conservation of packets” principle and is motivated by the general physics of flow [1].

We can use the vector representation method to observe how the underlying AIMD behaviour is distorted in a more realistic scenario – e.g., two TCP flows sharing a bottleneck link in the well known network simulator “ns” [12]. One such trajectory is shown in fig. 5 (fig. 7 shows the corresponding bandwidth/time plot); the movement looks similar to some trajectories we saw in our small diagram based simulator. It is hard to tell whether this behaviour is a result of queuing delay, the additional mechanisms in TCP or both.

We used the diagram to demonstrate a different effect: the impact of the Active Queue Management mechanism RED on TCP. Other than traditional DropTail queueing, RED introduces two queue length thresholds, *min_th* and *max_th*. If the queue length is smaller than min_th, packets will pass through unharmed. If max_th is exceeded, packets are dropped. If the queue length is in between min_th and max_th, a probabilistic marking

function is applied. This function contains some randomness, which is the key to avoid traffic phase effects (unwanted synchronization of flows) [13].

Figure 6 shows the trajectory for the very same scenario, but with RED instead of DropTail queueing. The corresponding bandwidth/time plot is shown in fig. 8. Evidently, the behaviour of TCP is “tamed” by RED – there are less dramatic fluctuations. The traditional way of comparing traffic flows would be to study the difference between figs. 7 and 8, but the differences between those figures is rather subtle. On the other hand, fig. 5 very clearly shows more fluctuations than fig. 6.

5. CONCLUSION

We have presented novel methods of using the vector representations of distributed control systems described in [11]. Three examples demonstrated the benefit of vector diagrams in analysis and design. We consider vector diagrams a powerful tool and hope that the ideas presented in this paper will support others in finding their own extensions for their specific use.

Our work is restricted to the context of congestion control in computer networks. We expect our method to be applicable to a much broader spectrum of control problems; this is a matter of future research.

6. ACKNOWLEDGEMENT

The idea of implementing a Chiu/Jain vector diagram simulator to study asynchronous loop delays was contrived by Ralf Hauber during lunch. This method led to the other extensions presented in this paper and can therefore be regarded as the primary reason for its existance. Also, the author is indebted to Andrew Barnhart for sending a hard copy of his CAPC ATM Forum Contribution.

REFERENCES

[1] V. Jacobson, Congestion Avoidance and Control, Proc. SIGCOMM '88, Palo Alto, CA, 1988.

[2] J. H. Saltzer, D. P. Reed, & D. D. Clark, End-to-End Arguments in System Design, Proc. Second International Conference on Distributed Computing Systems, 1981, 509-512.

[3] D. Chiu and R. Jain, Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks, *Journal of Computer Networks and ISDN*, *17*(1), 1989, 1-14.

[4] D. G. Luenberger, *Introduction to Dynamic Systems - Theory, Models, and Applications* (John Wiley & Sons, New York 1979).

[5] R. Johari & D. Tan, "End-to-End Congestion Control for the Internet: Delays and Stability". To appear in IEEE Transactions on Networking.

[6] Milan Vojnovic, Jean-Yves Le Boudec, & Catherine Boutremans, Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times, Proc. IEEE Infocom 2000.

[7] Injong Rhee, Volkan Ozdemir, & Yung Yi, TEAR: TCP emulation at receivers -- flow control for multimedia streaming, Technical Report, Department of Computer Science, NCSU. Available from: http://www.csc.ncsu.edu/faculty/rhee/export/tear_page/

[8] S. Gorinsky & H. Vin, Additive Increase Appears Inferior, Technical Report TR2000-18, Department of Computer Sciences, University of Texas at Austin.

[9] D. Bansal & H. Balakrishnan, Binomial Congestion Control Algorithms, Proc. IEEE INFOCOM 2001.

[10] K. Ramakrishnan & S. Floyd, A Proposal to add Explicit Congestion Notification (ECN) to IP, RFC 2481, January 1999.

[11] A. W. Barnhart, Explicit Rate Performance Evaluations, ATM Forum Technical Committee, Traffic Management Working Group, Contribution ATM Forum/94-0983 (October 1994).

[12] S. McCanne & S. Floyd, ns Network Simulator, http://www.isi.edu/nsnam/ns/

[13] S. Floyd & V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, *1*(4), 1993, 397-413.

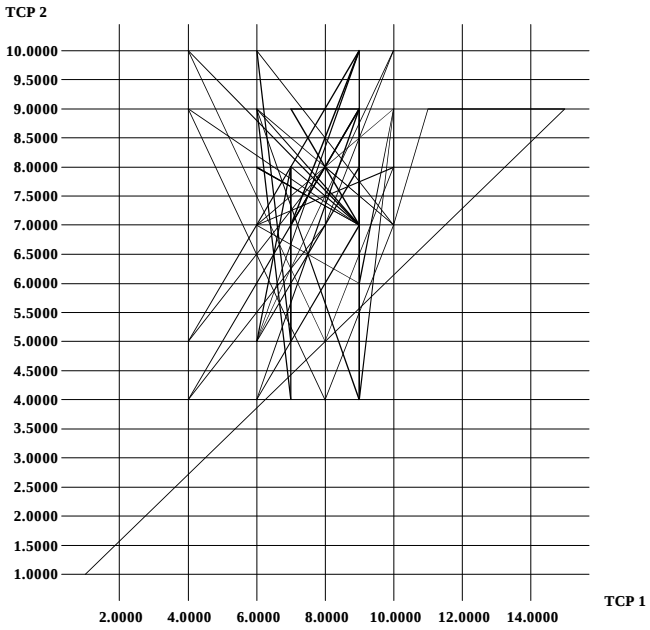


Fig. 5: Vector diagram of two TCP sources sharing the same bottleneck with DropTail queueing

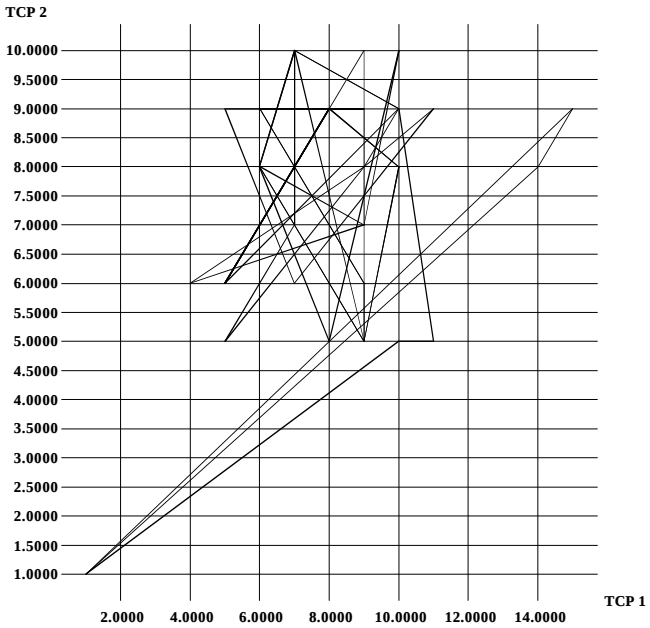


Fig. 6: Vector diagram of two TCP sources sharing the same bottleneck with RED queueing

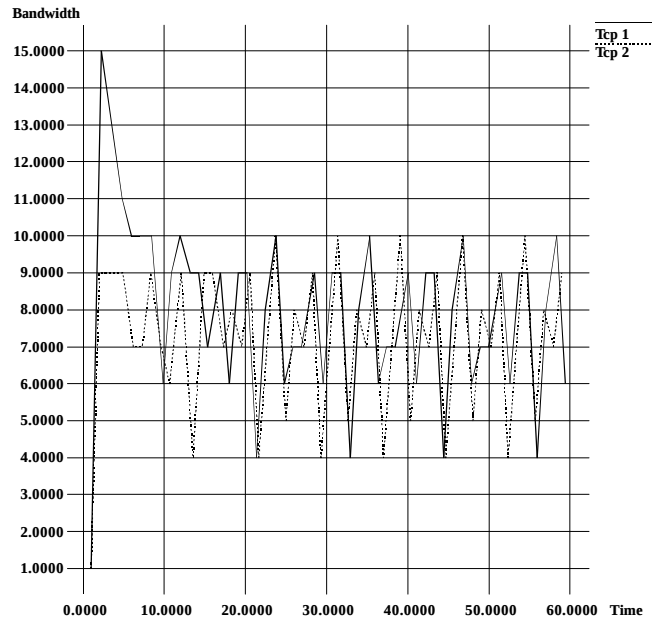


Fig. 7: Bandwidth / time plot of two TCP sources sharing the same bottleneck with DropTail queueing

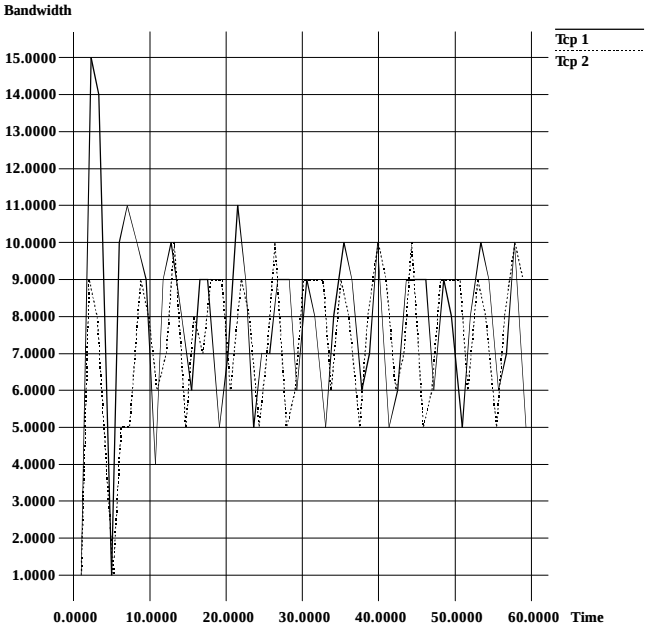


Fig. 8: Bandwidth / time plot of two TCP sources sharing the same bottleneck with RED queueing