CAVT — A Congestion Avoidance Visualization Tool

Michael Welzl
Leopold Franzens University of Innsbruck
Institute of Computer Science
Technikerstr. 25/7
A-6020 Innsbruck, Austria
michael.welzl@uibk.ac.at

Max Mühlhäuser
Darmstadt University of Technology
Dpt. of Computer Science, Telecooperation
Alexanderstr. 6
D-64283 Darmstadt, Germany
max@informatik.tu-darmstadt.de

ABSTRACT
We present CAVT, a simple yet powerful tool to visualize the behavior of congestion control mechanisms. Essentially being a simulator that builds upon the well-known Chiu/Jain vector diagrams, CAVT provides researchers, teachers and students with a new abstraction level that has already shown its value in practice.

Categories and Subject Descriptors
C.2.5 [**Computer Communication Networks**]: Local and Wide-Area Networks—*Internet*

General Terms
Design, Experimentation, Theory, Verification

Keywords
AIMD, Congestion Control, Protocol Design, Stability, TCP, Vector Diagrams

1. INTRODUCTION
Congestion control can sometimes be tricky, be it for the student attempting to comprehend the underlying concepts, the teacher who tries to get them across or the researcher. While our tool is clearly useful for all these groups, this paper was mainly written with network researchers in mind. When designing a new or analyzing an existing mechanism, a plethora of questions come up:

- Is the mechanism stable?
- Does it scale?
- How efficient is it in terms of capacity utilization and speed of convergence?
- Is it fair? Does it realize or at least approximate a certain type of fairness? Is it TCP-friendly?
- Does it show a smooth rate — i.e. is it applicable for streaming media applications?

Each question leads to a number of issues that should be examined; for instance, stability can depend on factors like the underlying rate control strategy or possibly an existing self-clocking property, which is normally lost if a mechanism is rate based. Often, due to the complexity of the mathematical models required, the stability of a mechanism is only proved in a simplified scenario (typically with equal round-trip times (RTTs) for all sources) while some intuition is required to achieve stability in more realistic cases (usually shown via simulation).

Clearly, there is a gap in the range of available modeling methods: while the simple case of users who share a single resource with an underlying fluid model and equal RTTs can be treated like a single control loop and can therefore be analyzed using common control theory methods, the next step — the case of heterogeneous round-trip times — is exceedingly complex. For instance, it is going to be very difficult for a researcher who is not a control theoretician to apply the control theoretic model for the heterogeneous feedback delay case known as a "Smith predictor" [7].

This paper describes a building block for a bridge across this gap. Following an introduction to Chiu/Jain vector diagrams in section 2, a tool based on these diagrams is described; its usage is explained in section 4, some details about the internal software structure are given in section 5 and section 6 concludes.

2. VECTOR DIAGRAMS
TCP roughly realizes a control algorithm called *AIMD* ("Additive Increase, Multiplicative Decrease"); this fact 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: system state transitions are regarded as a trajectory through an n-dimensional vector space — in the case of two controls (which represent two synchronous users in a computer network), this vector space is two-dimensional and can be drawn and analyzed easily.

Fig. 1 shows a vector diagram with three of the linear control algorithms in [3]: AIAD ("Additive Increase, Additive

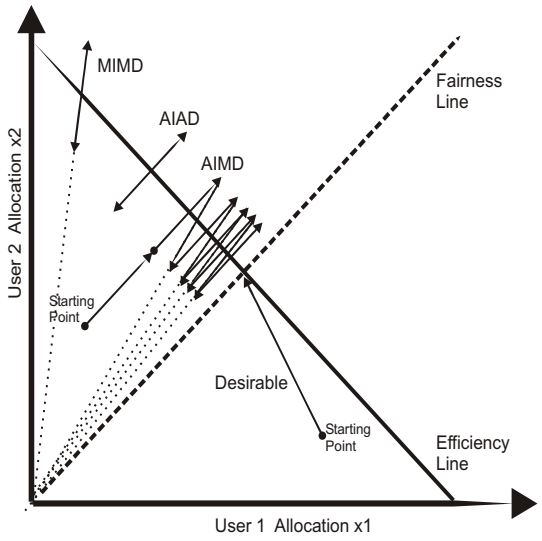

Figure 1: Vector representations of distributed linear control algorithms

Decrease"), MIMD ("Multiplicative Increase, Multiplicative Decrease") and the already mentioned AIMD. MIAD ("Multiplicative Increase, Additive Decrease") is missing for the sake of clarity because the other three algorithms 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 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 (in a computer network where users share several resources, there are many other definitions) 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 of the Efficiency Line and the Fairness Line. The "Desirable" arrow in fig. 1 represents the optimal control: it quickly moves to the optimal point and stays there (is stable). It is easy to see that this control is unrealistic for binary feedback: provided that both users see 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 the rate at the same time corresponds to moving along at a 45 ° angle. This effect is produced by AIAD: both users start at a point underneath the Efficiency Line and move upward at an angle of 45 °. The system ends up in an overloaded state (the state transition vector passes the Efficiency Line), which means that it now sends 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.

The same is true for MIMD, but here, a multiplication by a constant factor corresponds with moving along an Equi-Fairness Line. AIMD actually approaches perfect fairness and efficiency, but due to the binary nature of the feedback, the system can only converge to an equilibrium instead of a stable point — 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 [5] and nonlinear controls [1].

3. A NEW ABSTRACTION LEVEL

As we have seen, vector diagrams are a very simple means to analyze the behavior of congestion control mechanisms. At first sight, the case of two users sharing a single resource may appear to be too unrealistic; on the other hand, while these diagrams may not suffice to prove that a mechanism is worthwhile, it is likely that a mechanism that does not work in this case is generally useless. Given the complexity of a real computer network, it is a good idea to approach a new design by gradually extending a model in small steps. Therefore, one should perhaps start with the simplest possible case — a very high abstraction layer.

A straightforward example of designing a new mechanism is to follow the abstraction levels 1 to 7 listed in table 1 and reconsider all the levels that were reached so far as soon as something goes wrong: at first, it needs to be checked mathematically whether the mechanism itself yields the desired quality if only one flow uses a single resource. Only if the answer is "yes", two synchronously acting users should be looked at using vector diagrams; if the mechanism still works, n users should be examined using mathematical methods. Then, one should proceed to the next abstraction level, and so forth.

The next step is usually where the difficulties start: how to extend the model to heterogeneous RTTs? Immediately going for discrete simulation as with the popular *ns-2* network simulator may not be a good choice because of the large amount of additional factors that come into play (queuing delay, phase effects, rate granularity limitations from packet sizes, ..). On the other hand, it is hard to imagine how a mechanism would work if, for example, one user would update the rate twice as fast as the other just by looking at a vector diagram. This is where the new *Congestion Avoidance Visualization Tool (CAVT)* tool that we present in this paper comes into play:[1] it enables researchers to graphically examine the behavior of a mechanism with two users, one resource, a fluid model but unequal RTTs before undertaking further steps. At first, it is much easier to gauge a mechanism based on the intuitively understandable behavior seen with this tool than based on results from more realistic simulations. Such results should rather be used to fine-tune the behavior afterwards.

CAVT is a small Java application that provides an interac-

[1]Throughout this document, the terms "congestion control" and "congestion avoidance" are used synonymously despite the difference of operating near the "knee" or the "cliff" [3].

Table 1: Common abstraction levels for network analysis and design

Abstraction level	No. users	RTTs	No. resources	Traffic model	Method
1	1	equal	1	fluid	maths
2	2	equal	1	fluid	vector diagrams
3	n	equal	1	fluid	maths
4	**2**	**heterogeneous**	**1**	**fluid**	**CAVT**
5	n	heterogeneous	1	discrete	"normal" simulation
6	n	heterogeneous	m	discrete	"normal" simulation
7	n	heterogeneous	m	discrete	real life experiments

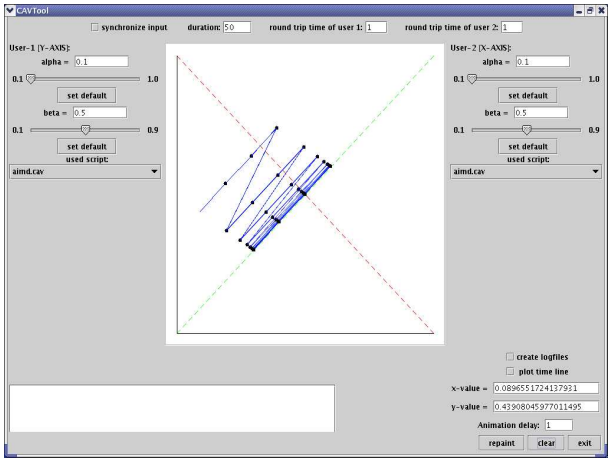

Figure 2: Screenshot of CAVT showing an AIMD trajectory

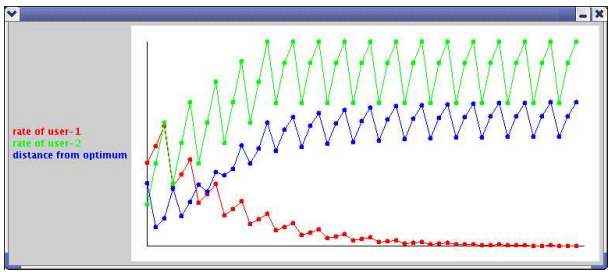

Figure 3: Screenshot of the CAVT time line window (user 1 = AIAD, user 2 = MIMD, equal RTTs)

tive graphical user interface where the user can set a starting point and view the corresponding trajectory by clicking the mouse in the diagram. This way, it is easy to test a congestion control mechanism in a scenario where the RTTs are not equal: the rate of a sender simply needs to be updated after a certain number of time units. In the present version, precise traffic information is fed to the sources and no calculations are done at routers. Also, there is no underlying queuing model; thus, assuming perfectly fluid behavior, the state of the network during time instances t_{i-x}, t_i and t_{i+x} must be given to the source with RTT x at time instances t_i, t_{i+x} and t_{i+2x} respectively.

Figure 2 shows a screenshot of CAVT showing an AIMD trajectory; time is visualized by the line segments becoming brighter as they are drawn later in (simulated) time, but this effect is not noticeable with AIMD and equal RTTs because its convergence causes several lines to overlap. In addition to tuning individual parameters for each rate allocation strategy, it is possible to alter the increase and decrease parameters α and β, change the RTTs and generate log files. Depending on the mechanism that was chosen, a varying number of parameter values with corresponding sliders is visible on each side of the drawing panel. Log files contain the trajectory seen in the GUI as well as the evolvement (time on the x-axis) of the rate of each user and the distance d of the current point p from the optimum o, which is simply the Euclidean distance:

$$d = \sqrt{(o_x - p_x)^2 + (o_y - p_y)^2} \quad (1)$$

The white space in the lower left corner is used for status and error messages. CAVT has the following features:

- A facility to choose between various existing mechanisms such as AIMD, MIAD, AIAD and some more complex mechanisms as in [1]. These mechanisms were defined with a very simple script language, which can be used for further experimentation.
- An additional window that shows time on the x-axis plotted against the rate of each user as well as the distance from the optimum during operation (enabled or disabled via the "plot time line" switch at the lower right corner). Figure 3, a screenshot of this window, depicts the evolvement of a trajectory where AIAD competes against MIMD with $\alpha = 0.1$ and $\beta = 0.5$. The lower line represents user 1 and the upper line represents user 2 — clearly, MIMD is more aggressive despite its stronger fluctuations. Also, the distance from the optimum (middle line) does not appear to converge to 0.
- Creation of log files that can be used with common plotting tools such as `xgraph` or `gnuplot` (enabled or disabled via the "create logfiles" switch at the lower right corner).
- GUI control of the trajectory drawing speed in order to facilitate recognition of the evolvement over time.

4. USING CAVT

The current version of CAVT can visualize arbitrary congestion control mechanisms as long as their decisions are based on traffic feedback only and do not use factors such as delay (as with TCP Vegas [6]); this covers all the TCP-friendly mechanisms that can be designed with the framework presented in [1] or even the "CYRF" framework in [9].

As mentioned earlier, a congestion control mechanism is specified using a special and simple script language. In this language, AIMD looks like this:

```
#define_param alpha range 0 to 1 default 0.1
#define_param beta range 0 to 1 default 0.5

    if(traffic < 1) {
        rate = rate + alpha;
    }
    else {
        rate = rate * beta;
    }
```

`alpha` and `beta` being the rate increase and decrease factors, respectively. These parameters can be tuned using the GUI because the `define_param` command was used — this is a preprocessor directive which causes the GUI to display a slider and requires the user to define an upper and lower limit as well as a default value. The AIMD script above was used to plot the trajectory in fig. 2; the two sliders at the left and right-hand side of the GUI therefore correspond with the `alpha` and `beta` parameters in the script.

The CAVT script language defines two special variables:

1. `traffic` — the recently measured traffic
2. `rate` — the rate of the sender

In the present version, the only available control structures are the sequence and the `if` statement; loops are not available as they naturally occur every RTT. Variables, which are always of type "floating point", follow the standard naming convention found in languages such as Java or C and do not need to be declared (a variable is implicitly declared as it is assigned a value). In addition to the standard math operations, the following commonly known functions can be used: `abs`, `sin`, `cos`, `tan`, `exp` (euler = `exp(1)`), `int`, `max`, `min`, `pow`, `sqrt` and `random`. If, for instance, we would like to assume that an AIMD sender cannot exceed the capacity because it is limited by the first link, we would need to change the script as follows:

```
#define_param alpha range 0 to 1 default 0.1
#define_param beta range 0 to 1 default 0.5
maxrate = 1.0; # rate can never exceed 100%

    if(traffic < 1) {
        rate = rate + alpha;
        rate = min(rate, maxrate);
    }
    else {
        rate = rate * beta;
    }
```

CAVT does not provide a development environment; the script files can be generated using a standard text editor. Upon execution (scripts are not compiled but interpreted), errors are shown in the white area at the bottom of the screen. As soon as a file with the ending ".cav" is found in the CAVT home directory, it is made available in the drop down menu of each user (to the left and the right of the trajectory canvas). These graphical elements are shown in fig. 2.

The tool can be used to

- Analyze the behavior of a single mechanism
- Analyze interactions between different congestion control mechanisms (due to the underlying object oriented structure, it is straightforward to assign a different mechanism to each user)
- Teach congestion control because of its visually oriented nature that helps students understand the dynamics of congestion control
- Design a new mechanism

In what follows, we will take a closer look at how to utilize CAVT for analysis and design purposes.

4.1 Analysis

One particularly interesting mechanism for analysis in the case of heterogeneous RTTs is AIMD, which is the predominant congestion control method in the Internet. With CAVT, it turned out that, under these circumstances, its behavior is nowhere near the behavior described in [3]. Depending on the different RTTs and the starting point (which seems to play a major role albeit this should not be the case!), AIMD can show pretty severe fluctuations. Interestingly, the fairness does not necessarily seem to stabilize in all cases.

Figure 4 shows an example trajectory of AIMD with RTTs of 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). From looking at this diagram, which was generated from CAVT log files, it is intuitively clear that the stability of TCP cannot only be accredited to AIMD, but, as a matter of fact, its self-clocking property seems to play a fundamental role. The finding that self-clocking is a very important aspect conforms with the results in [4], where the stability of equation-based congestion control is enhanced by introducing the "conservation of packets" principle. Also, one can see that time invariant analytical models of AIMD as in [8] can only describe the long-term average behavior while neglecting the somewhat weird short-term dynamics.

Clearly, such a diagram does not represent a proof of any sort; among other things, it is impossible to detect the presence of equilibrium points from looking at fig. 4. Also, since there is not even a queuing model for the single resource, there are too many simplifications. The information from the figure is nothing but a hint: "AIMD may not be the ideal solution in the heterogeneous RTT case."

It would seem that users of CAVT would benefit from being able to visualize composed algorithms as in standard

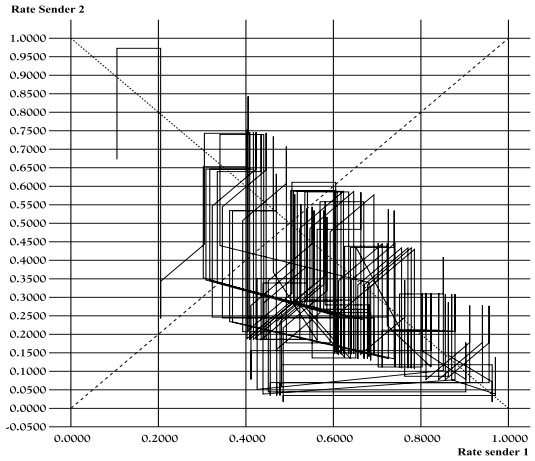

Figure 4: AIMD: RTT = 7 time units vs. RTT = 2 time units

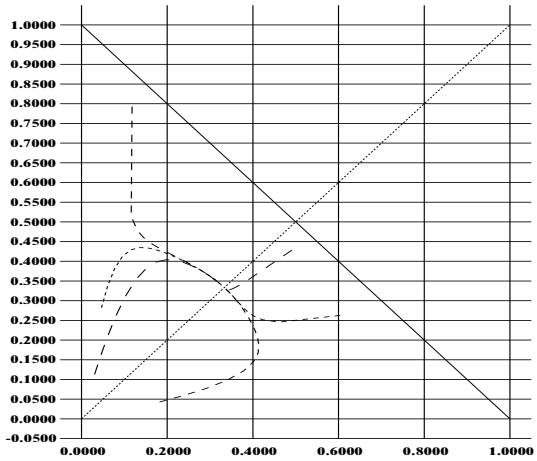

Figure 5: Some CADPC trajectories

TCP — this is however a nontrivial problem. The "conservation of packets" principle, or self-clocking property, for instance, is based on the assumption of discrete packets; thus, it cannot be incorporated in a fluid flow model. Similarly, RTT estimations only seem to make sense if at least a single queue is assumed for the given resource. While it is feasible to model a queue at this abstraction level, it was decided to leave this feature as a possible future extension because it adds significant complexity to the tool (e.g., how to choose the queue length? Does it really have to be a Drop-Tail queue?). Distinguishing between a "Slow Start" and "Congestion Avoidance" phase is useless until one can distinguish between a timeout and a loss notification event, which requires a RTT estimation feature to work properly. As such model extensions clearly build upon each other, all of a sudden, one ends up with quite a detailed model, giving rise to new problems such as appropriate parameter setting depending on the TCP implementation — it may in fact be inappropriate to introduce such complexity at this abstraction level.

4.2 Design

The "Performance Transparency Protocol" (PTP) is a means to efficiently query routers along a path for performance related data. In particular, it can be used to "ask" for a precise traffic measurement from the bottleneck router, leading to better adaptation because of more than just binary feedback. In the context of this document, the protocol itself is of no further concern; interested readers should consult [10] for more details.

It is obvious that, in contrast with mechanisms that strictly rely on implicit binary feedback from dropped packets, PTP places a burden on routers. Also, the very idea of sending additional signaling messages to control congestion is not very popular. Thus, any mechanism using PTP feedback should clearly yield notably better results than TCP and its "relatives". A prototype of CAVT was constructed as an aid to design such a mechanism in the first place, and, as we will see, it worked well. During this process, the abstraction levels in table 1 were strictly followed.

Due to the similarities between PTP feedback and the ATM "Available Bit Rate" service, the numerous existing ATM ABR switch mechanisms served as a starting point. The main differences are that with PTP, all calculations are performed at the end systems and that the available information is limited to a traffic measurement (e.g., there is no way to distinguish between individual flows). It appears that the most closely related ATM switch mechanism (no per-flow state, no flow counting, just available bandwidth information) is "Congestion Avoidance with Proportional Control" (CAPC) by Andrew Barnhart [2]. 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; additional scaling factors and an upper and lower limit ensure that the fluctuations diminish with each update step. Basically, convergence to fairness occurs even in the heterogeneous RTT case because the CAPC calculation is performed repeatedly at the switch, independent of the individual RTTs.

In CAVT, CAPC was implemented as a logic to be followed at the end nodes. In the case of equal RTTs, it immediately converged to fairness and efficiency — the real difference between this model and ATM ABR is only evident when the RTTs are changed. In several experiments, it became intuitively (visually) clear that, in order for the mechanism to reach fairness, a source should not adapt its rate to the current load (as with CAPC) but to the relationship between its current rate and the available bandwidth. In other words, if a user whose rate represents 40% of the capacity total load increases the rate very slightly, and a user whose load represents 5% of the capacity increases the rate drastically, the system converges to fair utilization of the single resource. This relationship prevails even if both users experience a different feedback loop delay.

Some example trajectories of the accordingly altered CAPC mechanism (now called "Congestion Avoidance with Distributed Proportional Control" to reflect its distributed nature) are shown in fig. 5; it evidently converges to the same point irrespective of the RTTs or the starting point. After some derivations and simplifications — abstraction level 3 in table 1 — the mechanism could be shown to realize discrete

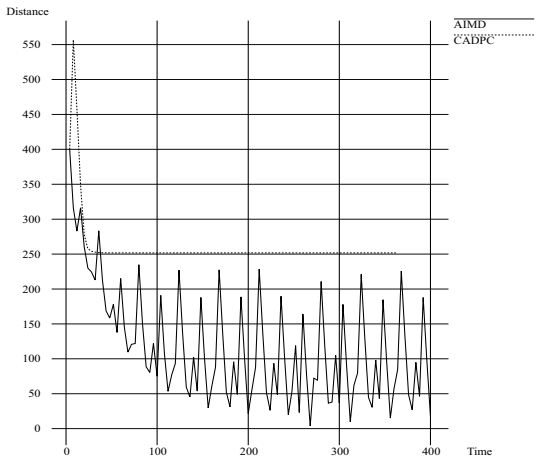

Figure 6: CADPC vs. AIMD distance from the optimum

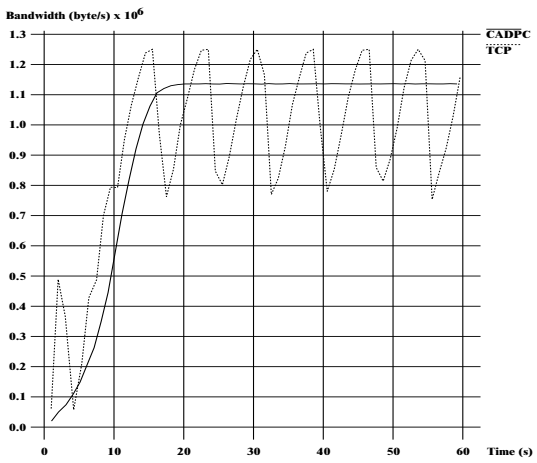

Figure 7: CADPC vs. TCP with a 10 Mbit/s bottleneck link

logistic growth, which is known to have a stable equilibrium at the point of convergence shown in the figure. The final rate update control law is:

$$x_i(t+1) = x_i(t)a\big(1 - x_i(t) - \lambda(t)\big) + x_i(t) \qquad (2)$$

where a is a constant factor ranging from 0 to 1 (the smaller, the smoother and less reactive the mechanism behaves), x is the rate of a sender, λ is the traffic measurement and all values are normalized with the nominal bottleneck link capacity. In the CAVT script language syntax, the CADPC code looks as follows:

```
#define_param a range 0.1 to 1 default 0.5

rate = rate * a * (1 - rate - traffic) + rate;
```

CADPC does not directly converge to the optimum but to $n/(n+1)$ where n is the number of flows in the system; this function rapidly converges to 1 as n increases. In a realistic network with hundreds and thousands of flows, the low point of convergence in fig. 5 can perhaps be regarded as a special case that may be negligible.

In conformance with table 1, the behavior of CADPC was then studied with the well-known "ns-2" network simulator. It showed extremely good results, some of which are reported in [12]; a large number of additional simulation studies are documented in [11]. One example is shown in fig. 7, where the total throughput of 10 CADPC and 10 TCP Reno flows using ECN (only flows of one kind at a time) shared a 10 Mbit/s bottleneck link. Clearly, the smooth convergence of CADPC coincides with the behavior that can be seen in fig. 6, which was generated from CAVT log files.

5. INTERNALS

There are numerous ways to extend this software: one could imagine drawing additional lines that are parallel to the efficiency line to illustrate various levels of congestion or including special network behavior (mechanisms that independently change the traffic measurements would, for instance,

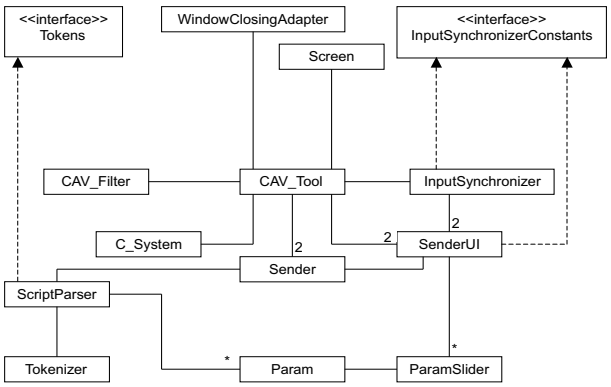

Figure 8: Class structure of CAVT

make it possible to study ATM "Available Bit Rate" switch mechanisms). As mentioned earlier, it would make sense to include a fluid flow model of a queue in order to enable studying the effect of RTT estimation. Such a queue could, of course, be extended to support various Active Queue Management mechanisms such as RED or REM. Another idea would be to support additional senders — for instance, it would be possible to have the user select two out of an arbitrary number of senders or even implement a three dimensional extension to directly visualize the effect that increasing the number of users has on a mechanism.

It was suggested to consider integrating CAVT with the *ns-2* network simulator instead of building the individual elements from scratch. However, we do not believe that this approach is fruitful: the strength of CAVT does not lie in its ability to visualize results from complex simulation scenarios (a function which is already provided by a wealth of visualization tools) but in its interactive user interface that provides a user with a very direct method to "play" with various congestion control strategies, which is helpful if one would like to gain more insight into their dynamics.

CAVT is an open source Java program. To facilitate its extension, it was designed in an object oriented manner.

Its simple class structure is shown as an UML diagram in fig. 8; here, broken arrows represent the implementation of an interface and multiple associations are denoted by the symbols "2" (two classes) and "*" (an arbitrary number of classes). The diagram only contains classes which are not available in the standard JRE — dependencies with such classes are also not shown.

The central class is `CAV_Tool`; it manages all objects and graphical elements. There are two objects of type `Sender`, each of which has its own `ScriptParser` and `SenderUI`. When something is changed in the user interface of a sender (a `ParamSlider` is moved), the respective `Sender` communicates this change from the `SenderUI` to its `ScriptParser`, where a script file is interpreted using the current parameter settings (realized via the `Param` class). The `Tokenizer` is used to disassemble the script file into easily parsable `Tokens` (which are defined in an interface). The class `CAV_Filter` takes care of loading ".cav" files; the time dependent trajectory plot is realized by the `C(anvas)_System`, which provides methods to add points and automatically scales the x-axis accordingly. The main window is represented by the `Screen` class. Finally, the `InputSynchronizer` function and the `InputSynchronizerConstraints` interface are related to a switch in the GUI which is called "synchronize input"; if this switch is enabled, any parameter change for one user will automatically affect the parameters of the other user.

6. CONCLUSION

In this paper, we introduced a new abstraction level at which congestion control can be looked at. This abstraction level allows researchers to bridge the gap between the cases of synchronous and asynchronous feedback loop delay in a simplistic manner without requiring in-depth knowledge of control theory. CAVT, the tool constructed for this purpose, can be used for numerous additional tasks, such as the analysis of existing congestion control mechanisms and their interactions as well as for teaching purposes.

CAVT is available from `http://www.welzl.at/tools/cavt`. We encourage readers to download, use and extend it at will; any feedback is more than welcome.

7. ACKNOWLEDGMENTS

The authors would like to extend their gratitude to Ralf Hauber, who inspired us to design a prototype of CAVT, and Christian Sternagel, who implemented and documented the version of CAVT described in this paper.

8. REFERENCES

[1] D. Bansal and H. Balakrishnan. Binomial congestion control algorithms. In *IEEE INFOCOM 2001, Anchorage, Alaska*, April 2001.

[2] A. W. Barnhart. Explicit rate performance evaluations. Technical Report Contribution ATM Forum/94-0983, ATM Forum Technical Committee, Traffic Management Working Group, October 1994.

[3] D. Chiu and R. Jain. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN*, 17(1):1–14, June 1989.

[4] S. F. Deepak Bansal, Hari Balalakrishnan and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *ACM SIGCOMM 2001, San Diego, CA*, August 2001.

[5] S. Gorinsky and H. Vin. Additive increase appears inferior. Technical Report Technical Report TR2000-18, Department of Computer Sciences, University of Texas at Austin, October 2000.

[6] S. O. L. Brakmo and L. Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *ACM SIGCOMM 1994*, pages 24–35, August 1994.

[7] S. Mascolo. Congestion control in high-speed communication networks using the smith principle. *Automatica*, 35(12):1921–1935, December 1999.

[8] J.-Y. L. B. Milan Vojnovic and C. Boutremans. Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times. In *IEEE INFOCOM 2000, Tel Aviv, Israel*, March 2000.

[9] N. Sastry and S. S. Lam. A theory of window-based unicast congestion control. In *IEEE ICNP 2002, Paris, France*, November 2002.

[10] M. Welzl. The ptp website: http://www.welzl.at/ptp.

[11] M. Welzl. *Scalable Performance Signalling and Congestion Avoidance*. Kluwer Academic Publishers, August 2003.

[12] M. Welzl and M. Mühlhäuser. Scalability and quality of service: a trade-off? *IEEE Communications Magazine*, 41(6):32–36, June 2003.