To appear in Journal of Computer Networks,
Elsevier Science, (Accepted March, 20, 2006).

Symbiotic Rate Adaptation for Time Sensitive Elastic Traffic with Interactive Transport

Javed I. Khan and Raid Y. Zaghal
Networking and Media Communications Research Laboratories
Department of Computer Science, Kent State University
233 MSB, Kent, OH 44242
javed|rzaghal@cs.kent.edu

Abstract*—Interactivity in the transport protocol can greatly benefit transport friendly applications generating streaming traffic. Recently we have developed iTCP, which can provide event notification to the subscriber of its communication service. This is operationally state equivalent to the conventional TCP except applications can optionally subscribe, receive, and in real-time react to selected local end-point events. This simple extension opens the horizon for a spectrum of smart application level solutions to be realized for many of the current hard problems. In this paper we demonstrate a new paradigm of congestion management for time sensitive elastic traffic. Based on the transport layer feedback, a rate adaptation mechanism kicks in. This mechanism provides a generation rate control with guaranteed TCP buffer delay. We have recently implemented and tested the real system on the Active Network (ABone) testbed for video streaming to worldwide sites. In this paper we share the performance of this system and report observed dramatic improvements in time-bounded streaming traffic.*

1 Introduction

In any real world service provisioning system interactivity can be considered a vital aspect of the design. If subscriber system is made aware of selected service level events, it is possible in many cases to device smarter solutions at the subscriber layer or above when the service layer encounters difficulty. Typically, upper layer entities have more global view of the system and thus are able to execute solutions that are more holistic and effective. Unfortunately, current network transport protocols do not have this transparency. It seems, for the case of networking that this critical shortcoming at the interface between existing applications and current networks has made solutions to some of the networking problems remarkably harder. With the advent of sophisticated applications and their advanced transport needs, current transport services are becoming increasingly inadequate. This inadequacy has also prompted recent attempts towards recreating new and much complex transport services which would partially recreate new functionalities at the network or system middle layers [1, 2, 24]. For example, Congestion Manager [1, 2] is a system layer component which provisions aggregate congestion control when multiple streams from the same end-point attempt to send. Unfortunately, majority of these ─though they offer specific functional advantages─ enormously increase the network or system layer complexity at the end. Unfortunately, such complex and permanent additions to the system layer appear to be questionable. When their complexities are weighted against their general advantages over a broad range of applications, they do not seem to be gaining any acceptance. Due to the same inadequacy, in the last few years it has also been felt that for advanced applications (such as real-time streaming); the applications themselves have to be more integrated in the solution. Particularly promising is the research in the new TCP friendly paradigm [13, 15, 16, 21, 22, and 23]. Due to the lack of convenient means to obtain network states, these systems rely completely on application-layer techniques to face the network impairment. Several works such as [4, 18] suggested sending multilevel redundant information which will eventually increase the burden on the network. Many have to tediously guess the underlying network state using end-to-end estimations. These can be cumbersome and even inaccurate guess work. A set of recent works [22, 23] suggested rate control based on formal models. These are robust formally. However, they are designed to be used on RTP like end-to-end estimations. Due to the inherent round trip delay involved, the adaptation time constant for these end-to-end estimations can be unbearable for more time critical applications, and it may severely degrade the model's intended performance. Overall, it is very difficult to build network friendly applications if the network itself is non-friendly and unwilling to interact. In contrast, a network service layer which is interactive[1] and transparent to demanding applications

[1] It is interesting to note that the original TCP proposal (RFC007, RFC793) did call for interlayer interactivity, but subsequent implementations ignored it. Some forms of callback facility have

can open up a new paradigm of building applications and systems which are genuinely network friendly. Indeed much more can be gained; with such transparency, at the application (or subscriber) layers not only much more sophisticated and targeted solutions can be designed, but also this simple and intuitive approach can drastically reduce the complexity of network system layers. It can also empower a new generation of truly transport friendly applications which can adapt and execute more effectively with dynamic network conditions ─which are not possible at network layers only. To provision such interactivity, we have recently implemented and released iTCP [8, 10]. We have also demonstrated a decoupling formalism called *Transientware* (T-ware), where sophisticated yet disposable application-level components can be selectively invoked when subscribed events occur at the transport layer [9]. This system makes a subset of its internal events/states accessible. The system is internally state equivalent to conventional TCP. Thus, it does not require all systems to be necessarily modified. Only a sending application at one endpoint which needs advanced adaptation may choose to use it. Also, its interface is legacy compatible, thus, other applications at the same endpoint can run without modification

A particular problem we try to address with this new interactive TCP is the optimum rate management-based congestion response and particularly the one for time-sensitive streaming traffic. Most of the network level schemes for congestion response are based on delaying traffic at various network points. The more classical schemes depend on numerous variants of packet dropping in the network, prioritization (graceful delay in router buffer), admission control (delaying at network egress points), etc. However, a key aspect to note in all is that they introduce *time distortion* in the transport pathway of the application. Though this is harmless to time-insensitive traffic such as email or FTP, but they distort the temporal characteristics of time-sensitive traffic such as multimedia streaming or control data. The recent solutions here too are based on complex network or system layer addition (such as [1]). We have demonstrated simple interactivity-based congestion management scheme for time-sensitive elastic traffic. Till to-date effectively no handle was given to application to participate in the congestion management. In contrast to network or system layer solutions, the general principle we follow is simple and intuitive. It seems an effective delay conformant solution for time-sensitive traffic can be designed if the original data volume can be reduced by its originator─ the application.

To demonstrate the efficacy of this principle, we have also designed a corresponding advanced video rate transcoder system [11, 12] that works in symbiosis with the network. This transcoder actively participates in a sophisticated symbiotic back-off scheme in application layer with deep transport level knowledge resulting in more effective joint quality/delay sensitive communication.

The adaptation is applicable for traffic where it is possible to dynamically adjust the data generation rate. We call it *elastic traffic*. Most perceptual data, such as audio and video streams generally belongs to this traffic class. The resulting scheme is similar in spirit to the TCP-friendly approaches. However, there is a fundamental difference in how it is done; the network or system layers remain as simple as possible. The responsibility of the network layer is simply to pass on only selected end-point events to the application. Since, the solution is now implemented at the application level; therefore it can be made much more sophisticated without a significant increase in network layer complexity.

In this paper, we provide a mechanism where, in the face of congestion, a generator can still provide sending buffer delay guarantees. As we will show the overall solution is not only intuitive and simple, but also surprisingly effective when compared to many other recently proposed schemes, which have involved much more complex system/network layer reorganization. We have tested the entire system for symbiotic video streaming to worldwide destinations using the Active Network Backbone (ABone) testbed, which has been recently developed with new facilities for automatic deployment of new protocols. In this paper we report performance of the system on the ABone.

The paper is organized as follows; next we show TCP's congestion control mechanism and internal events. In section 3 we discuss iTCP, its event model and an overview of its implementation architecture. A complete discussion of iTCP can be found in our technical report [10]. In section 4 we explain our mathematical model for *symbiosis throttling* which provides optimal video encoding rate. In section 5, we show how the T-ware modules were implemented based on the symbiosis throttling model and iTCP feedback, and in section 6 we discuss experimental results. Section 7 concludes the paper.

only been marginally cited in recent literature. Unfortunately none gives the issue any in depth technical consideration.

2 Congestion Control in TCP

TCP is a connection-oriented unicast protocol that offers reliable data transfer as well as flow and congestion control. TCP maintains a congestion window that controls the number of outstanding unacknowledged data packets in the network. Sending data consumes slots in the sender's window, and the sender can send packets only when free slots are available. When an acknowledgment (ACK) for outstanding packets is received, the window is shifted so that the acknowledged packets leave the window and the same number of free slots becomes available.

2.1. Congestion Control Algorithms

On startup, TCP performs slow-start, during which the rate roughly doubles each roundtrip time to quickly gain its fair share of bandwidth. In steady state, TCP uses an additive increase, multiplicative decrease mechanism (AIMD) to detect additional bandwidth and to react to congestion. When there is no indication of loss, TCP increases the congestion window by one slot per roundtrip time. In case of packet loss indicated by a *retransmission timeout*, the congestion window is reduced to one slot and TCP re-enters the slow-start phase. Packet loss indicated by three duplicate ACKs results in a window reduction to half of its previous size. Therefore, the two principal mechanisms that TCP uses to detect network congestion are *(i)* when the retransmission timer times out and *(ii)* when three duplicate ACKs arrive. Two algorithms then contribute to the TCP congestion control behavior; these are the classic algorithm of slow-start and congestion-avoidance [5], and the augmentation of fast-retransmit and fast-recovery [6]. Figure 1 and Figure 2 respectively show the relevant details of the two algorithms.

2.2. Congestion Control Events

Table 1 lists six events that internally occur when the TCP invokes a congestion control algorithm. Although many other TCP events might occur during a TCP session (e.g., flow control events or connection establishment and termination events), we are only interested in congestion control events.

In table 1, the column labeled (SSCA) refers to events that take place in the Slow Start/Congestion Avoidance algorithm, and the label (FRFR) refers to events that take place in the Fast Retransmit/Fast Recovery algorithm. These events are also presented in figure 3. The graph given in figure 3a shows the sequence of events of the SSCA algorithm and their affect on effective bandwidth. Figure 3b shows the same sequence for the FRFR algorithm. However, in general design we expect only a subset of the internal events of the protocol to be of interest to the subscriber application. Only a subset of these is made accessible via the interface. An application instance typically subscribes even to a subset of the accessible events. The column (Sub) shows subscribable events in our design.

3 iTCP: interactive TCP

3.1 Model Architecture

The *Interactive Transparent Networking* paradigm provides a framework for applications to receive instant notification about the service state information from various network layers on which it critically relies. In this section we provide a brief description of the framework. More information can be found in our previous work in [8, 9, 10]. The purpose of the framework is to provide a means to application programs running in the upper layers to subscribe to a selected set of transport events.

We have provisioned the above in the following way. Under this framework protocols are modeled as per their extended finite state model (EFSM). The protocols, particularly the transport protocol then makes a selected set of events accessible. Also, some selected standard compliant state variable are also made readable. However, this process does not involve changing any functionality of the core protocols themselves. Thus this reengineering has no impact on the network side dynamics of the protocol. Applications can then selectively subscribe to the available events, if any. By subscribing, an application program essentially binds itself to the network protocol and declares that it wishes to be notified–through signaling—when certain events (state transitions) occur at that level. Once they are notified, subscribers can also then pull up the required service state information when available, perform the actual action by application level components called *Transientware Modules* (or *T-ware*) which run at the application layer. In figure 4 we show the general architecture of a TCP-based interactive protocol.

Upon opening the socket, an adaptive application can bind a *T-ware* to a designated TCP event by subscribing with the kernel. This is represented by arrows 1 and 2 in figure 4. The binding is optional; if the application chooses not to subscribe, the system defaults to the silent mode identical to classic TCP. When the event occurs in TCP, the kernel sends a signal (3a) and at the same time it saves the event information (3b). A special *Signal Handler* catches the signal and probes the kernel for the event type (4a, 4b). The handler then invokes the appropriate *T-ware* module to serve the event (5). These *T-ware*s are usually small programs supplied by the user or by a third-party as ready-to-run executables custom-

designed to handle certain events. One *T-ware* is forked by the signal handler per signal to take some action knowing that the event (`evt`) has just occurred in the kernel space (e.g., to reduce outbound bit rate to the transport layer when a congestion event is reported).

The probing API allows the *T-ware* to probe additional information about the state of the TCP connection (6a, b). This information includes relevant fields from the TCP control block such as: current *send window size* (`snd_wnd`), *congestion-control window size* (`snd_cwnd`), *threshold for slow-start* (`snd_ssthresh`), *retransmit value* (`t_rxtcur`), and *round-trip time* (`t_rtttime`). The *T-ware* can use these transport state parameters to determine precise adaptive action plan for the application (for example calculate a new generation schedule that guarantees certain delivery time bound of the traffic). The model allows the *T-ware* to probe the TCP at any time to get the updated values of these parameters. The model also allows event access control which allows the network administrator to restrict access to kernel state variable by each *T-ware* module.

3.2 Compatibility & Interoperability

In the design of the Interactive TCP protocol we have retained three important protocol engineering principles namely (i) *network functional compliancy* (ii) *state-transition* compliancy, and (iii) *default-to-classical extension* interface model. These principles provide important advantages to the scheme. Its interface mechanism with IP layer remains functionally identical with TCP classic. Thus, it remains fully operational with all other currently deployed TCP remote-hosts. Only the server/transport hosts which are interested to run advanced adaptive software applications need to locally upgrade to the extended model. Other communicating hosts need not to modify anything on their site. Secondly, its *state-transition* behavior also remains identical to that of TCP classic. Thus, all other network embedded transparent elements which rely on certain assumptions about TCP behavior (such as congestion control schemes, fairness,) will also not be affected. Thirdly, the *default-to-classical* extension of application programming interface (API) enables the TCP interactive host to concurrently run all other existing legacy applications without any code modification. Thus, it keeps all legacy applications 100% compatible even on the updated host. Notably, many other suggested congestion management schemes potentially violate these critical protocol extension principles.

The above provisioned interactivity has dramatic consequences on applications. The proposed interactivity opens up a new horizon in implementing advanced optimization in the network. Armed with the knowledge of the network states, much more sophisticated and efficient solutions to various network impairment problems can now be easily implemented-which are not otherwise possible from classical closed network layer. Below we now show one such case where the application can ensure optimum traffic control to the point that it can ensure even delay guarantee.

4 Symbiosis Throttling Model

The key to the system is the intermediate event gluing mechanism—or as we call it *symbiosis throttling*. It performs the key task of dynamically specifying the target rate for the application based on the event notification interrupt. The idea is to accept the event feedback provided by the underlying interactive transport layer, and generate a corresponding rate feedback for rate formation capable applications. This feedback is estimated in a way that ensures transport service with applications specified delay conformation over the otherwise classic transport service.

The main idea is that when a time-out event ($\xi = 1$) occurs in the transport, we let the subscriber rate retract to a smaller rate. We call this retraction state as *frugal state*. We now show how to optimally design the frugal state's retraction point.

4.1 Analysis of Symbiotic Throttling

Let *g(t)* be the generation function denoting the data rate at which the rate formation capable application produces data as a function of time. Let *w(t)* is the bandwidth function provided by the transport channel over which, the application sends the data. Figure 5 explains the model.

During normal operation $w(t) \geq g(t)$. When a loss event is detected (e.g., timeout) the transport bandwidth retracts to some smaller effective value due to window resizing. The underlying cause might be a packet loss or a congestive delay deep inside network. In either case, the sender transport buffer builds up and results in increased communication delay. In response to the loss event, we let the subscriber adjust its generation rate to a lower generation state (we call this state the *frugal bandwidth* state). The normal operation is however by a *satisfied bandwidth* state. In any practical feedback system there is also always a reaction delay in the feedback loop. Let τ be the reaction time needed by the subscriber process to react and adjust its rate. Given the above model, the particular design problem we address is the following:

Given the bandwidth function w(t), the generation function g(t), the satisfied state bandwidth (H), and the

upper bound on the acceptable data delivery delay (d_Q), determine the best possible frugal state (generation rate and its duration) for which the bound d_Q can be ensured.

Here the delay bound d_Q is the maximum delivery delay an application can sustain between generation endpoint and delivery endpoint of the application layer.

We now further define two additional concepts important for the derivation to be presented.

4.2 Critical-delay-point inequality

Assuming the loss is detected at time t_{loss}. After the loss assume it takes t_{equal} time for the transport system to again equalize the transport bandwidth with the frugal state generation rate of the subscriber. This is the point where then window rate increases and again equals the generation rate i.e. $w(t)= g(t)$. We call this point the *even-point*.

Since the generation rate is larger than the transport rate before the even-point is reached, therefore the transport buffer will build up until the even-point is reached. The buildup will again begin to gradually decrease after the even-point. Thus the bytes entering the buffer exactly at the even-point will face maximum delay. Let this time be called critical-delay-point $t_{critical}$. Thus, if the transport buffer already has Q bytes in it (before moving to the frugal state), the buffer size at even-point is given by the left hand side of equation—(1a). Let d be the maximum acceptable delay, then the following inequality must hold. We name it *critical-delay-point* constraint:

$$\max(0,Q) + \int_{t_{loss}}^{t_{equal}} g(t) - w(t) \cdot dt \le \int_{t_{equal}}^{t_{critical}} w(t) \cdot dt$$

$$\text{Or, } \max(0,Q) + \int_{t_{loss}}^{t_{equal}} g(t) \cdot dt \le \int_{t_{loss}}^{(t_{equal}+d)} w(t) \cdot dt \qquad \text{--(1a)}$$

4.3 Recovery-Point Inequality

The bytes entering the transport buffer after the even-point will face lesser but still non-zero delay. This data too will be entering into the buffer which will be quite full. These additional bytes, those generated between the even-point and the critical-delay-point, will still populate the buffer. Therefore our ultimate goal is to take the buffer into pre-event state before returning to normal generation. Thus, the subscriber system should still continue to operate at somewhat less than satisfied state even after even-point. This extended frugality will allow remaining buffer buildup to dissipate—completely erasing the effect of the original timeout event. We define this instance of time, when the entire buildup will dissipate and return to pre-event state as the *full-recovery-point*. Let's call it the recovery time $t_{recovery}$, then the following second inequality in equation—(1b) must hold. We call it *full recovery-point* constraint.

$$\max(0,Q) + \int_{t_{loss}}^{t_{recovery}} g(t) \cdot dt \le \int_{t_{loss}}^{t_{recovery}} w(t) \cdot dt \qquad \text{--(1b)}$$

4.4 Frugal State Determination

The two inequalities respectively can provide a general solution for the level and duration of the frugal state for any general transport bandwidth and generation function. It can also predict the corresponding recovery time. Below, we solve specifically for the case where the iTCP transport control is similar to TCP (binary-back-off and additive-increase) and a piecewise step g(t). For simplicity, we assume that when a loss event is detected the window function decelerates to zero (i.e., w(tloss)=0). We first solve for a fast reacting system, where the reaction time is very small and let the buildup before subscriber reaction is Q. Let g(t) is a piece-wise step function. We further assume that the post-fault w(t) is a linear function with bandwidth acceleration m.

Let d_0 is the maximum buffer delay tolerable by the application data. Given a maximum propagation delay limit T_P, and bandwidth $w(t)$, we can say that $d_Q = d+T_P+(1/w(t))$ where d is the total delay faced by the byte entering at critical-delay-point. Since, typically $w(t)>>1$, then d can be approximated by $d = d_Q - T_P$. Let T be the time it takes the system to reach the even-point (i.e., $T = t_{equal} - t_{loss}$). The left hand side is the sum of the buildup before the TCP reacted and the data that arrived after the reaction until the even-point. The right hand side is the total data released until even-point from TCP. Then critical buffer equality (1a) can be expanded into:

$$Q + mT^2 \le \frac{1}{2} \cdot m \cdot [T + d]^2 \qquad \text{--(2)}$$

$$T^2 - 2Td - d^2 + \frac{2 \cdot Q}{m} \le 0$$

It solves to:

$$T = d \pm \sqrt{2d^2 - \frac{2Q}{m}} \qquad \text{--(3)}$$

Only positive real solutions are practical. For any given system arbitrary delay bound cannot be met. In that case both the solutions are imaginary. The model can now be used to determine the limit on the maximum acceptable delay. For the real solution the minimum delay requirement cannot be smaller than:

$$d_{\min} \geq \sqrt{\frac{Q}{m}} \quad \text{--(4)}$$

T can have two solutions. Both solutions are positive if:

$$d \leq \sqrt{\frac{2Q}{m}} \quad \text{--(5)}$$

Otherwise, only one solution is positive. From T, we can determine the frugal state bandwidth of the generator function. It should be stepped down to:

$$h = mT = md\left(1 \pm \sqrt{2 - \frac{2Q}{md^2}}\right) \quad \text{--(6a)}$$

Out of the two solutions, the best possible frugal state (the one which allows higher transmission rate in the frugal state) is:

$$h_{best} = mT = md\left(1 + \sqrt{2 - \frac{2Q}{md^2}}\right) \quad \text{--(6b)}$$

And the other solution is:

$$h_{other} = mT = md\left(1 - \sqrt{2 - \frac{2Q}{md^2}}\right) \quad \text{--(6c)}$$

The second solution, when exists, provides a second possible frugal state with lower generation rate. If this solution is taken, the data-generation allowance at frugal state will be lower. However, it will result in faster recovery.

The next question we ask is how long the system should stay in frugal state. We first derive a lower bound. This is given by the critical recovery time:

$$T_{critical} = 2T = 2d\left(1 \pm \sqrt{2 - \frac{2Q}{md^2}}\right) \quad \text{--(7)}$$

For the special case, when, the initial buildup (or reaction time) is zero, the corresponding height and duration of the frugal state is:

$$h_{best} = md(1 + \sqrt{2}) \quad \text{--(8)}$$
$$T_{critical} = 2d(1 + \sqrt{2})$$

For step $g(t)$, between the critical-point and recovery-point the system continues to be in frugal state accelerating the recovery. Corresponding recovery time is the complete duration of the frugal state. It can be determined by solving equality—(2), and is given by:

$$T_{recovery} = \frac{h}{m}\left(1 + \sqrt{1 + \frac{2Qm}{h^2}}\right) \quad \text{--(9)}$$

For the general case, when there is a buffer buildup due to the reaction delay=τ, the buildup can be estimated from the satisfied state generation rate and the reaction delay. Let H be the bandwidth satisfied state generation rate, when τ is small, Q can be approximated by:

$$Q = \tau(H - \frac{m\tau}{2}) \quad \text{--(10)}$$

The slop m can be approximated from the effective RTT and the segment size (up to the current *threshold* TCP window grows exponentially).

$$m = \frac{B_{channel}}{RTT\left(\log_2 \frac{B_{channel}}{2I} + \frac{B_{channel}}{2I}\right)} \approx \frac{2I}{RTT} \quad \text{--(11)}$$

Here $B_{channel}$ is the target channel bandwidth, I is the increment step or segment size and RTT is the round trip delay estimate used by TCP to resize its window.

For symbiosis with the underlying transport protocol, each time a retransmission timeout event (at $t=0$), reported the *frugal state bandwidth* is determined as following.

$$\begin{aligned} g(t) &= w(t) \quad when\ \xi = 0, t = 0 \\ &= md\left(1 + \sqrt{2 - \frac{2Q}{m.d^2}}\right) when \xi = 1 \\ &= w(t) \quad at \quad t > T_{recovery} \end{aligned} \quad \text{--(12)}$$

5 Symbiosis Mechanism: The *T-ware*

The important task of gluing between the transport layer and the application unit (MPEG-2 rate transcoder) is finally performed by the symbiosis unit (*T-ware*). *T-ware* essentially executes the throttling model. It estimates the parameters required to execute the model by probing iTCP as needed and finally it provides the application the rate parameter as it requires operating in symbiosis. Below we describe its parameter estimation process and invocation operations.

5.1 Estimation of the Model Parameters from iTCP States

To be able to use the symbiosis throttling model described above, we now show how the model parameters can be estimated from the TCP state and event times made accessible by the iTCP. Namely, we want to find τ, H, RTT, and I from the TCP internal state variables now made available by iTCP.

5.1.1 Reaction Delay (τ)

The reaction time τ was approximated as following:

$$\tau = (t_{ResponseTime} - t_{EvenTtime}) + u_{rate} + u_{TCP} \quad \text{--(13a)}$$

`EventTime` is when the signal handler was invoked. The quantity u_{TCP} is a constant approximating the time taken by iTCP's kernel signaling. We assume $u_{TCP} = 0$. Thus `EventTime` is used here as an approximation of the real time when the event has occurred deep in the TCP layer. `ResponseTime` approximates the time of the real rate reduction (i.e. when the calculated h_{best} is saved to "`rate.par`" file). Quantity u_{rate} is the estimate of the rate control systems reaction time after receiving the new rate, we also assume $u_{rate}=0$.

5.1.2 Round Trip Time (RTT)

RTT is directly returned by TCP from its state variable `TCPstate->t_rtttime`. TCP implementation uses the following process to measures round trip time (*RTT*) and retransmission timer out (*RTO*). First, TCP measures the *RTT* between sending a byte with a given sequence number and receiving an acknowledgment that covers that sequence number (*M* denotes the measured *RTT)*. Afterwards, TCP updates a smoothed *RTT* estimator *R* using the low-pass filter:

$$R \leftarrow \alpha.R + (1-\alpha).M \quad \text{--(13b)}$$

Where α is a smoothing factor with a recommended value of 0.9. The smoothed *RTT* is updated every time a new measurement *M* is made. This means that 90% of each new estimate *R* is from the previous estimate and 10% is from the new measurement *M*. TCP then calculates a new retransmission timer out value (*RTO*) based on the mean and variance of the *RTT* measurement. The technique was proposed by Jacobson [5]. He used the mean deviation as a good approximation of the standard deviation since it is easier to compute. In each *RTT* measurement *M*, the following calculations are made:

$$\begin{aligned} Err &= M - A \quad \text{--(13c)} \\ A &\leftarrow A + gErr \\ D &\leftarrow D + h_{gain}(|Err| - D) \\ RTO &= A + 4D \end{aligned}$$

Where *A* is the smoothed *RTT* (an estimator of the average) and *D* is the smoothed mean deviation. *Err* is the difference between the measured value just obtained and the current *RTT* estimator. Both *A* and *D* are used to calculate the next *RTO*. The gain *g* is for the average and is set to 1/8. The gain for the deviation is h_{gain} and is set to 1/4.

5.1.3 Maximum Segment Size (*I*)

RTT is directly returned by TCP from its state variable `TCPstate->t_maxseg`. Maximum segment size MSS (we called it *I* in our model), is the largest 'chunk' of data that TCP can send to the other end. When a connection is established, each end has the option to announce the MSS it is willing to receive. When TCP sends a SYN segment, it can send an MSS value up to the outgoing interface's MTU, minus the size of the fixed TCP and IP headers. In our experiment, TCP chose MSS of 1460 bytes.

5.1.4 Satisfied State Bandwidth (*H*)

H can be calculated by finding the ratio: number of bytes transmitted so far over elapsed time since the video has started. This is estimated from two TCP state variables (*t_rtseq* and *t_iss*) and two local measurements:

$$H = \frac{(TCPstate \rightarrow t\_rtseq - TCPstate \rightarrow t\_iss) \times 8}{u_{videoStartTime} - u_{eventTime}}$$

The difference:

$$TCPstate \rightarrow t\_rtseq - TCPstate \rightarrow t\_iss$$

Between the state variables gives how many bytes have been transmitted so far. We multiply it by eight to convert it to bits since all our calculations will be in bit/second units. The $u_{videoStartTime}$ is the time when the video started; it was saved in a file by the encoder prior to sending the first frame.

5.2 *T-ware* Implementation

The Symbiosis Throttling of equation 12 is actually implemented in the loss event handler or the T-ware module. Basically, we need to calculate h_{best} and $T_{recovery}$ every time the T-ware is invoked.

The role of the signal handler was merely to catch the signal from the kernel and invoke the appropriate T-ware. To simplify things we let the encoder subscribe with the *"retransmit timer out"* event only. Figure 6a outlines a sketch of the signal handler code. After catching the SIGIO signal, it needs to know which socket generated the event (line 5) then it probes the socket to get the event number and the event handler name (line 6). Once retrieved, it forks a new child and executes the appropriate T-ware for the event type (lines 8-10). It passes to the T-ware the time when the event occurred and the socket id. Once activated, the T-ware ─shown in figure 6b─will first probe the socket to retrieve the following parameters from TCP:

`t_rtttime` (round trip time), `iss` (initial send sequence number), `t_rtseq` (sequence number being timed), and `t_maxseg` (maximum segment size). The T-ware then calculates the satisfied state bandwidth generation rate H, the reaction delay τ as explained before. Afterwards, the handler calculates m, Q, h_{best}, and $T_{recovery}$ in a straightforward manner (lines 8-11). In line 13, it stores the reduced rate h_{best} in the "*rate.par*" file which will be noticed immediately by the symbiotic encoder. Finally, the event handler starts a timer for recovery. When the timer reaches $T_{recovery}$, the recovery T-ware kicks-in and returns the encoder bit rate to the normal rate. The recovery T-ware is outlined in figure 6c.

6 Experiment and Performance Analysis

The results presented here are not simulation. We used a real implementation of iTCP and the MPEG-2 Symbiotic Transcoder. The results presented came from its live performance experiment conducted over the real video delivery sessions over the Internet to the worldwide sites. Before presenting our results first we will describe the testbed and the setup.

6.1 The ABone Testbed

We wanted to run the experiment on the real Internet environment. This required running the symbiotic transcoder, a sender equipped with iTCP transport protocol, and a set of players on remote hosts around the world. We could have done this manually by conventional methods to reach a number of remote nodes worldwide. But this would have required extensive overhead to setup the testbed and maintain it. Furthermore, this will not be flexible or practical if we wanted to switch to a new set of remote nodes. Therefore, we decided to use the ABone. The ABone, developed under the DARPA ANI program forms a virtual network infrastructure on which a growing set of active network components can be tested and experimentally deployed. ABone is an operational network and provides an Internet wide network of routing as well as processing capable nodes. Providers can contribute confederation of computing capable nodes. Independent application involving multiple trust domains can be securely launched and executed. It also specifically allows new transport protocol components to be remotely deployed. ABone nodes are available from Europe, Asia and North America. Individual nodes are contributed and managed locally and independently by the contributing site administrators. However, the administrators do not have to manage the remote users. Researchers can remotely install and execute programmed components on any collection of these nodes via the ABone backbone management and control backplane being a part of a centralized user pool. The codes are distributed via an enlisted set of Trusted Code Servers (TCS), which help authenticating them prior to distribution. The security domains are handled by the backplane control system. The backplane is being maintained by the ABone Coordination Center (ABOCC) at ISI at the University of Southern California. ABone status can be monitored live from the ABOCC web site [3]. In addition to the iTCP machine we have a cluster of 10 registered ABone nodes in our lab at Kent State University (mk00 - mk09 .maunakea.medianet.kent. edu). Four of these nodes run on FreeBSD and the rest run on Linux. At the time of our experiment (Nov. 2003), there were 24 Linux nodes, 5 Solaris nodes, and 12 FreeBSD nodes registered on the ABone.

For our experiment we simply sent our video player to one of the ABone's trusted code server at (http://bro.isi.edu/KENT). Then we configured and registered our iTCP machine (kawai.medianet.kent.edu) as a primary node on the ABone to run iTCP and the symbiotic transcoder. The server remained in a traditional (non active) node. The ABone allowed the automatic loading of the sessions on designated machines worldwide.

6.2 Experiment Setup

This experiment describes the performance of an MPEG-2 ISO/IEC13818-2 (176x120) resolution video encoded with base frame rate of 2 Mbps at main profile.

Figure 7 illustrates deployment setup. The video server runs on a classic TCP machine (*manoa*) and feeds the video stream into the transcoder, which runs on the iTCP active node (*kawai*). To create some forced congestion in the experiment we also run a congestion injector program on a first-mile active gateway router (*lahaina*). The injector creates congestion bursts. Figure 8 shows the congestion injector. It allows the *duration*, and the *interval* between bursts to be programmed for three consecutive bursts. During a congestion burst the router will simply disrupt its routing table by removing the entry that leads to the player machine. When the burst time is over, the router restores the routing table back to normal. In our experiment, we used 3 three-second bursts at 10 second intervals. A three-second burst usually triggers 1 to 2 retransmit timer out events depending on the player's location. We ran the players on selected remote ABone node. We repeated the experiment on four ABone nodes, two in the US and two in Europe. Some general network conditions observed of the four target nodes are shown in table 2. The player and transcoder units were enhanced to collect detail frame arrival, and delivery measurements.

6.3 Impact on Video Frame Delay

For a detailed comparison we have performed several sets of experiment. These are: `iOPT`, `iEXP`, `iOFF`, and `Classic` modes.

- In the `iOPT` (optimal backoff) mode, we activated the throttling model described above to calculate h_{best} and $T_{recovery}$ with every loss event. We challenged the system to provide the frames at guaranteed *d=6, 4,* and *2* seconds.
- As a base case we also repeated the experiment with the same congestion schedule in classical mode where the interactive and symbiotic rate adaptation features were turned-off and the entire system ran in classical TCP mode. We call it `Classic` mode.
- For comparison we also included the case of `iEXP` used in our previous work to demonstrate the effectiveness of iTCP. It is a non-optimized simple heuristics-based symbiosis which performs a lazy binary back-off scheme for the generation rate. The method adapts but it can not provide QoS guarantee as with the throttling model. Detail of this simple scheme is in [9]. With the `iEXP` (exponential backoff) mode, we used a predetermined reduction ratio ($\alpha = 0.35$) and multiplied that with current bit rate to calculate the frugal state bit rate, we also used a fixed recovery time of 4.0 seconds.
- We also repeated the experiments in another mode called `iOFF` for overhead estimation. The mode is similar to classic TCP. No symbiosis is performed. But the event subscription mechanism remains active. This will be explained later.

In all the iTCP enabled runs (`iOPT`, `iEXP` and `iOFF`), the transcoder subscribes with iTCP for the retransmission timer out event.

In the experiment, we took frame-wise detail event trace of the first 750 frames of the video at both sending and receiving ends. For a given discard threshold time in the receiving end we also traced which frame was successfully received or not at the MPEG-2 player. As explained earlier, we traced four transport aware cases (`iOPT` with three values of delay tolerance *d=2, 4,* and *6*, and `iEXP`) and two transport unaware cases (`iOFF` and `Classic`) please, return to table 3 for details on the four running modes.

Now we show the dramatic impact of iTCP's interactivity based symbiosis. In figure 9 we plot the delay experienced by the video frames in terms of frame arrival time at the player for the four modes mentioned above. In addition, we also show the ideal expected frame delivery time—`Expected` in the figure—based on linear generation rate. As can be seen iTCP outperformed classical TCP; after each congestion burst, the unaware cases (`Classic` and `iOFF`) continuously fell behind. The delay built up and it could hardly recover. This is evident by the step jumps in the delay line. The TCP aware cases also suffered some step buildup, but it was much smaller and it could recover after few seconds due to the rate retraction.

In table 4 we present the frame delay and acceptance ratio comparison for the whole stream. The table shows the performance for three choices of delay tolerance *d=2, 4,* and *6* seconds. For each value of *d* we traced the four running modes (`iOPT`, `iEXP`, `iOFF`, and `Classic`) and recorded the average delay in seconds that each frame has experienced and the frame acceptance ratio at the receiving player ABone nodes. It can be clearly noticed that iTCP/aware modes achieved low delays and high acceptance percentages while the unaware/classic modes suffered from higher delays and lower acceptance percentages. Clearly iTCP's T-ware mechanism allowed the application to use sophisticated optimization techniques to optimally control the temporal qualities of its traffic.

6.4 Symbiotic Rate Control

In the next set of experiments we will present the internals of the symbiosis mechanism in more detail. Figure 10 depicts the symbiotic frame rate transcoding that occurred due to the joint rate specification at the rate control logic at the symbiosis unit and in the transcoder for each frame. In the figure we show four plots for the four target ABone nodes. Each plot represents the iOPT mode run for the delay tolerance case *d=4*. Table 6 presents the actual values of h_{best} and $T_{recovery}$ that controlled the frugal mode operation as calculated by the symbiosis controller T-ware after being activated by each one of the three loss events created in the experiments.

In each plot of figure 10 we see the target bit rate and the retraction ratio as specified by the symbiosis controller, and the resulting outgoing actual frame rate generated by the transcoder. The timer out events (in this case there are 3 timeout events) reported by iTCP resulted in the symbiosis unit to modify the rate according to the optimal backoff symbiotic rule (equation-12). Though, the precise MPEG-2 generation rate varied widely from frame to frame to accommodate the frame type, but the general trend followed the specified target. Table 5 provides the overall stream compression due to symbiotic adaptation for the entire stream (`iOPT` and `iEXP` cases), as compared to the normal non-symbiotic cases (`iOFF` and `Classic` cases). In the `Classic` and

iOFF cases, there were no adaptation (thus retraction =1). Compared to this both iOPT and iEXP reduced the overall delivered bits about 83-95%.

However, it is interesting to note that iEXP without its optimization logic, operated more aggressively and compressed more (for example iEXPs' 85% vs. iOPTs' 91% in *d=2*). In comparison iOPT operated more confidently (reduced less bits), yet it achieved higher temporal quality (average delay is 2.6 sec vs. 0.5 sec for same cases).

6.5 Observation at Application Level:

In the above experiments we illustrated how the symbiosis mechanism worked from the video transport protocol (MPEG-2) and the network transport protocol TCP layers beneath it. In this plot we will illustrate how this mechanism appears from the very top—at the application layer itself. An application receives and delivers uncompressed frames. The performance metric this end-system uses is the temporal and spatial quality difference between the transmitted and the reproduced uncompressed video frames. The underlying MPEG-2 transport protocol and the network layer TCP together provides the transport. The specific compression, windowing etc. and other detail mechanisms are external techniques to the end systems.

In figure 11 each frame is plotted as a point in the video quality/frame delay plane. The figure shows four plots for the four ABone nodes, and each plot represents three running modes (iOPT with *d=4*, iEXP, and Classic). As can be seen from the region of the three QoS distributions, in TCP-classic, although frames have been generated with SNR quality ranging between 18-40 dB, but many of these frames suffered long delay and were lost in transport. In contrast, the interactive iOPT mode can deliver all the frames with guaranteed delay when the bulk of the frames had 10-32 dB quality.

It is interesting to note that the iEXP mode achieved the same tradeoff, but since it took a non-optimized and thus more aggressive approach in symbiotic rate reduction the quality suffered more and recorded values as low as 7 dB. Fundamentally, what iTCP has offered is a qualitatively (as opposed to the quantitative improvements offered by any unaware solution) new empowering mechanism, where the catastrophic frame delay can be traded off for acceptable reduction in SNR quality.

6.6 Interactivity Overhead

The dramatic advantage in application level performance came at a cost since the event tracking mechanism added some overhead. We were also curious to find out the overhead of the event mechanism. To track the overhead, we recorded the total data transmission time under the three conditions (iOPT, iOFF, and Classic). The left most bar of figure 12 plots the transport time for the optimal interactive mode where we activate both event delivery and symbiosis. To observe the overhead of the event service, in the iOFF mode we used the iTCP implementation, however, we stopped the symbiotic reduction so the transport layer handled the same amount of data. As expected the overall transmission time increased in all three cases. However, in the third column (Classic mode) run we turned off the interactive service altogether and thus we saved its overhead and lost its benefit. As can be seen, the slight increase in the event delivery overhead was vastly offset by the application level technique. The advantage the application gained from the event delivery was much bigger than the overhead.

7 Conclusions

In this paper, we have presented a case of video rate symbiosis mechanism in line with current advances in TCP friendly systems. We have presented the case through a clean 'interactive' generalization of the classical TCP, and a novel implementation of a symbiotic MPEG-2 transcoder. We collected the results of our experiment by running real video session experiments on the Internet on the ABone testbed. In the previous discussion we have demonstrated the case of quality conformant congestion control for elastic video traffic. The approach exposed the overall advantage of network 'friendly' applications. However, it also departs significantly from the mainstream TCP friendly systems that have been suggested recently in two senses; First, it does not add any new major component in the network software structure. One of the principal strength of the proposed scheme is its relative simplicity at the network layers, yet, its effectiveness. It only expects some form of interactivity directly from the concerned network protocol as a general interface feature. Thus, there is no expectation of (or conflict with) additional services (such as combined congestion control from multiple applications). Secondly, the applications do not have to be designed dependent on other auxiliary indirect probing tools or network utilities, nor it excludes their use when available. Current TCP dynamics are highly optimized for various network dynamics such as fair queuing and congestion control scenarios. The proposed interactivity does not alter any dynamics in the network side and thus those optimizations. Its actions are directed completely at application side.

Nevertheless, the approach adds lesser but yet some complexity in the network layer. The augmentation of

the notification feature increases the normal mode delay of TCP slightly. The actual cost depends on the intensity of the binding between events and T-wares. However, as shown by the results with a prudent design the impact on the network level transfer rate (based on lower layer measurement), if any, can be widely surpassed by the gain made at the application layer. However, an interesting safeguard of this scheme is that a wrong design will only affect the application at fault and will have no negative effect on the network dynamics. However, the proposed interactivity is not an alternate to other network level schemes, rather is a complimentary scheme. Some of the information measured by the auxiliary tools suggested by other approaches might be already available (or are being estimated/tracked) at lower layers anyway. At least this is the case with TCP congestion. The direct protocol interactivity we propose thus seems to be the logical path that can avoid potential duplication of efforts.

There are profound and fundamental advantages of event based interactivity. The proposed interactivity opens up a new horizon in implementing advanced adaptive optimization for applications and middleware. The event based instant knowledge of the network states, enables sophisticated and efficient solutions to various network impairment problems—which cannot be otherwise realized via classical closed network layer design.

It is further interesting to note that besides enabling a new class of adaptive applications, the proposed interactivity can have important implication on network layers as well even when mediated by application level T-ware components. Indeed, it can play a critical role in cross-layer optimization. We have demonstrated such optimization where, the slow handoff of Mobile IP (MIP) can be substituted by an infrastructure-free end-to-end mobility solution called IPMN [20]. The scheme uses event-based interaction between several network layers mediated by T-wares. It offers blazingly fast event based handoff and much simplified transport (no tunneling delay) than MIP.

We have also recently demonstrated that interactivity can even be used for protocol extension. Researchers are regularly finding numerous otherwise excellent extensions and improvements to existing protocols. Unfortunately, in networking there is hardly any deployment path especially for the case of lower level protocols. We have recently shown how two such well known extensions—which did not find realization—WTCP and SNOOP can be easily realized at the application layer via protocol interactivity [19]. Again, we would like to emphasize that even the original TCP protocol called for interactivity (RFC007, RFC793). The proposed work thus can be considered as a renewed investigation to this overlooked but probably one of the most profound features of TCP/IP networking that original designers envisioned.

8 References

[1] Andersen D., Bansal D., Curtis D., Seshan S., and Balakrishnan H., "System Support for Bandwidth Management and Content Adaptation in Internet Applications," Proc. of OSDI'00, Oct. 2000, San Diego, CA.

[2] Balakrishnan H., Rahul H., and Seshan S., "An Integrated Congestion Management Architecture for Internet Hosts," Proc. of ACM SIGCOMM, Cambridge, MA, Sep 1999. pp.175-187.

[3] Berson S., Braden B., and Ricciulli L., "Introduction to the ABone," Feb. 2002, available at http://www.isi.edu/abone/DOCUMENTS/ABarch/.

[4] Briceño H., Gortler S. and McMillan L., "NAIVE--network aware Internet video encoding," Proc. of the 7th ACM International Conference on Multimedia, Oct. 1999, Orlando, FL, pp. 251-260.

[5] Jacobson V. and Michael J. Karels, "Congestion Avoidance and Control," Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.

[6] Jacobson V., "Modified TCP Congestion Avoidance Algorithm," end2end-interest mailing list, April 1990.

[7] Ke J. and Williamson C., "Towards a Rate-Based TCP Protocol for the Web," Proc. of the 8th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecomm. Systems, 2000.

[8] Khan J., Zaghal R., and Gu Q., "Rate Control in an MPEG-2 Video Rate Transcoder For Transport Feedback based Quality-Rate Tradeoff," PV2002, Pittsburgh, PA, April 2002.

[9] Khan J., Zaghal R., and Gu Q., "Dynamic QoS Adaptation for Time Sensitive Traffic with Transientware," IASTED WOC'03, Banff, Canada, July 2003.

[10] Khan J. and Zaghal R., "Event Model and Application Programming Interface of TCP Interactive," Technical Report 'TR2003-02-02', Feb. 2003.

[11] Khan J. and Patel D., "Extreme Rate Transcoding for Dynamic Video Rate Adaptation," 3rd Int. Conference on Wireless and Optical Communication WOC 2003, Banff, Canada, July 2003, pp. 410-415.

[12] Khan J. and Gu Q., "Network Aware Symbiotic Video Transcoding for Instream Rate Adaptation on Interactive Transport Control," IEEE NCA'01, Oct. 2001, Cambridge, MA, pp.201-213.

[13] Pradhan P., Chiueh T. and Neogi A., "Aggregate TCP Congestion Control Using Multiple Network Probing," Proc. of the 20th International Conference on Distributed Computing Systems, ICDCS 2000.

[14] Raman S., "A Framework for Interactive Multicast Data Transport in the Internet," Ph.D. thesis, UC-

Berkeley, May 2000.

[15] Rejaie R., Handley M., and Estrin D., "Architectural Considerations for Playback of Quality Adaptive Video over the Internet," Proc. of the IEEE ICON 2000.

[16] Sisalem D. and Wolisz A., "Towards TCP-Friendly Adaptive Multimedia Applications Based on RTP," Proc. of the 4th IEEE Symposium on Computers and Communications, 1998.

[17] Stevens W. R., "TCP/IP Illustrated, Volume 1: The Protocols," Addison-Wesley, 1994.

[18] Wolfinger B., "On the potential of FEC algorithms in building fault-tolerant distributed applications to support high QoS video communications," Proc. of the sixteenth annual ACM symposium on principles of distributed computing, 1997, pp. 129-138.

[19] Khan J. and Zaghal R., "Interactive Transparent Networking-Modeling Examples of Snoop and WTCP Protocols," Journal of Computer Communications, Spring 2005, Elsevier, http://www.sciencedirect.com/

[20] Zaghal R., Davu S., and Khan J., "An Connection Oriented Mobility – Performance Analysis with Voice Traffic," IEEE Wireless and Optical Communications, WiOPT05, 3rd IEEE Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, Riva Del Grande, Italy, April, 2005, pp.219-228.

[21] Floyd S., Handley M., Padhye J., and Widmer J., "Equation-Based Congestion Control for Unicast Applications," August 2000. SIGCOMM 2000.

[22] Handley M., Floyd S., Pahdye J., and Widmer J., "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448, Proposed Standard, January 2003.

[23] Ahmed T., Mehaoua A., Boutaba R., and Iraqi Y., "Adaptive Packet Video Streaming Over IP Networks: A Cross-Layer Approach," IEEE Journal on Selected Areas in Communications, Vol.23, no.2, February, 2005, pp.385-401.

[24] Wang R., Yamada K., Sanadidi M. Y., and Gerla M., "TCP with sender-side intelligence to handle dynamic, large, leaky pipes," IEEE Journal on Selected Areas in Communications, 23(2):235-248, 2005.

```
initially, cwnd = segmentSize (1 segment);
ssthresh = 65535 bytes;
win_size = min (cwnd, snd_wnd);
When congestion occurs, do:
        ssthresh = max(win_size/2, 2);
        if congestion was due to timeout
                cwnd = segmentSize;
        for every ACK received:
                if (cwnd <= ssthresh)
                  cwnd += 1;
                else
                  cwnd += 1/cwnd;
```

Figure 1. Slow Start/Congestion Avoidance (SSCA) mechanism.

```
When a 3rd duplicate ACK is received:
   ssthresh = max(2, min(cwnd, snd_wnd)/2);
   Retransmit missing segment;
   cwnd = ssthresh + 3;

Each time another duplicate ACK arrives, do:
   cwnd = cwnd + segment_size;
   transmit a new segment;

When a new ACK arrives, do:
   cwnd = ssthresh;
```

Figure 2. Fast Retransmit/Fast Recovery (FRFR) mechanism.

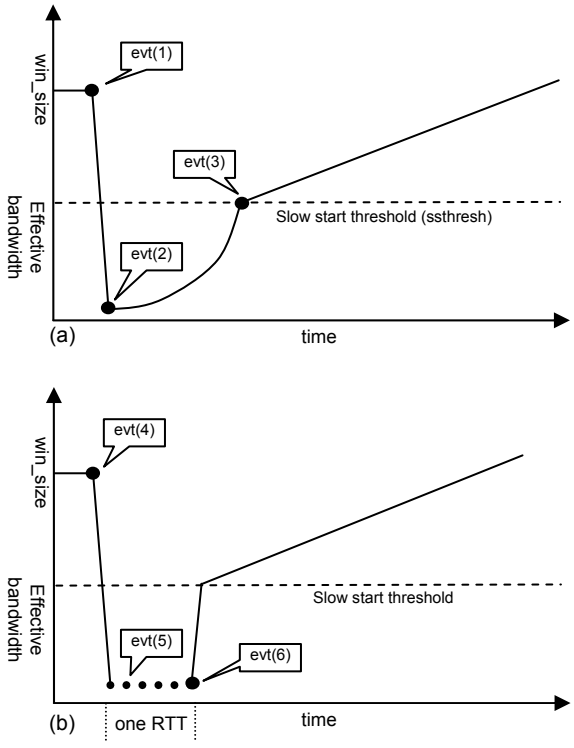

Figure 3. Effective bandwidth changes due to TCP congestion control internal events.

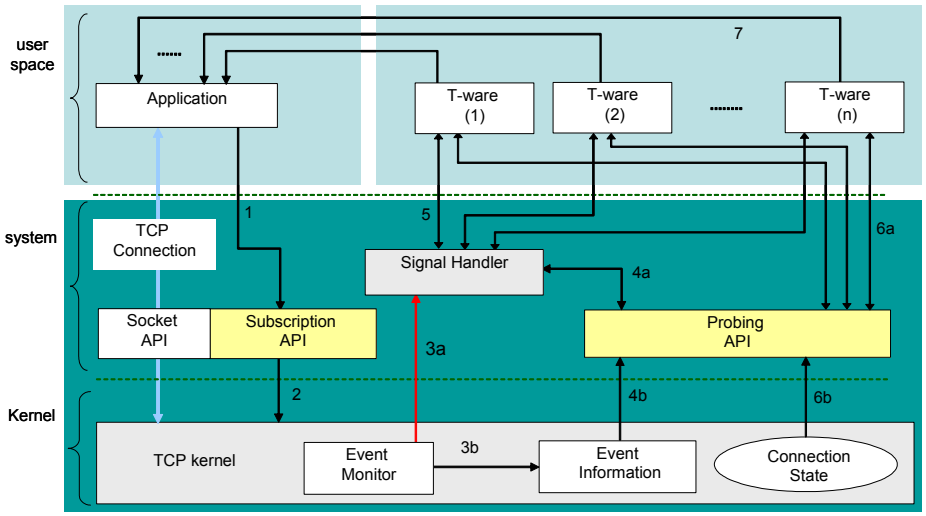

Figure 4. The iTCP extension and API.

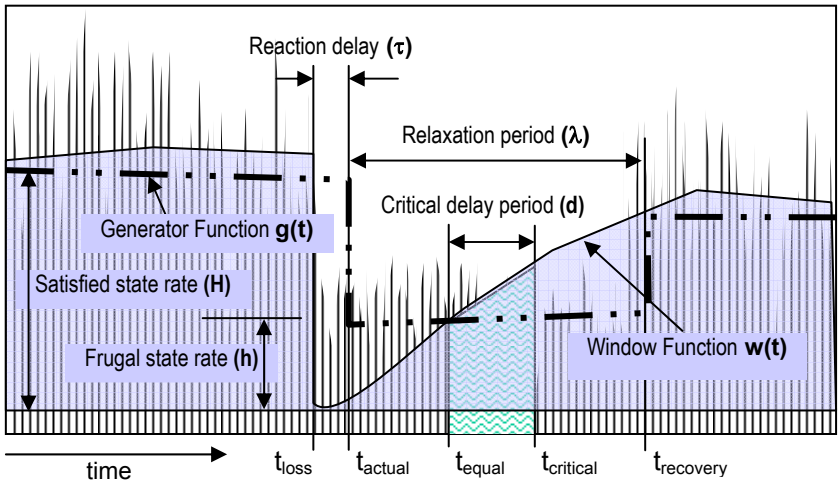

Figure 5. Symbiosis throttling model.

```
1:  SignalHandler(signum){
2:    struct evtSubInfo *handInfo;
3:    if (signum == SIGIO){
4:      gettimeofday(eventTime);
5:      s = GetSockid();
6:      ProbeEvtInfo(s, handInfo);
7:      if (!(child = fork())){
8:        execl(handInfo->handler, s, eventTime);
9:        exit(0);
10:     }//end if
11:   }//end if
12: }//end SignalHandler
```

(a)

```
1: Loss-T-ware(socket s, eventTime){
2: struct connState *TCPstate;
3:       probeSocket (s, TCPState);
4:       fscanf(timeFile, "%ld", videoStartTime);
5:       H = (TCPState->t_rtseq - TCPState->t_iss)*8
               / (videoStartTime - eventTime);
6:       gettimeofday(respTime);
7:       responceDelay = respTime - eventTime;
8:       m = 2*(TCPState->t_maxseg)*
                      8 / TCPState->t_rtttime;
9:       B=responceDelay * (H -(m*responceDelay)/2);
10:      h_best = m*d* (1 + sqrt(2-(2*M/m*d)));
11:      T_recovery = (h_best/m) *
                     (1 + sqrt(1+(2*B*M)/(h*h)));
12:      ratefile = fopen("rate.par", "w");
13:      fwrite(h_best, ratefile);
14:      StartRecoverTimer();
15: }//end LossTware
16: }
```

(b)

```
1:  Recovery-T-ware(signum){
2:     if (signum == SIGALRM){
3:        waitTimecount++;
4:        if ( waitTimecount && !rateOK &&
                        (waitTime>T_recovery)){
5:              ratefile = fopen("rate.par", "w");
6:              fwrite(originalRate, ratefile);
7:              rateoK = 1;
8:        }//end if
9:     }//end if
10: }//end RecoveryTware
```

(c)

Figure 6. Pseudo code of the Signal handler, the Event handler, and the Recovery handler.

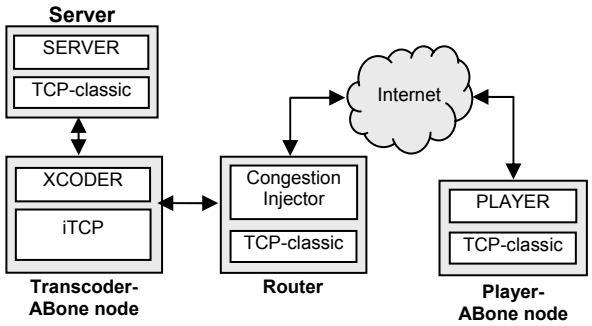

Figure 7. Experiment setup. The congestion injector creates consecutive timed bursts of congestion.

```
int bursts = 3;
int burstTime[]={3, 3, 3};
int interBurstTime[]={10, 10, 0};
sleep(10);
for (i=0; i<bursts; i++){
   remove entry from routing table;
   sleep(burstTime[i]);
   return entry to routing table;
   sleep(interBurstTime);
}
```

Figure 8. Congestion Injector mechanism

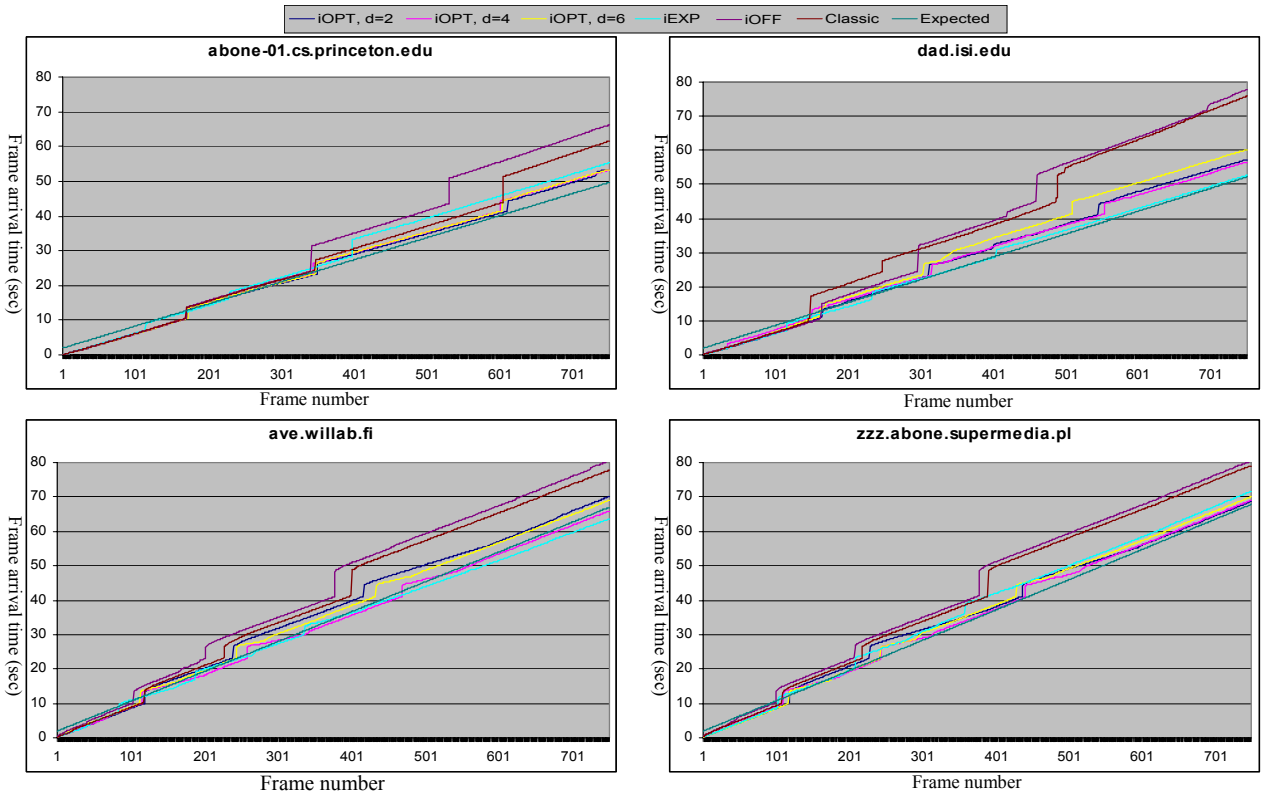

Figure 9. Frame arrival delay on the four ABone nodes.

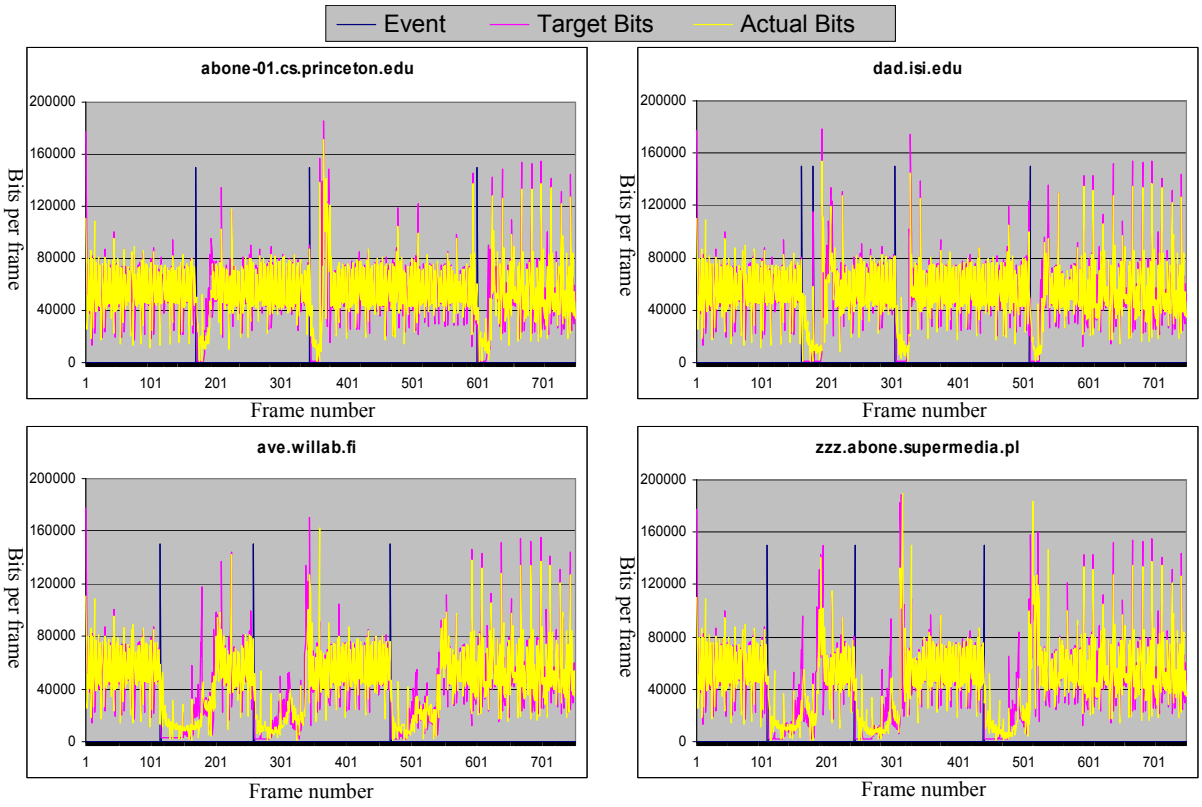

Figure 10. Symbiotic Rate Reduction on the four ABone nodes. The plot shows the case of 'iOPT' mode with delay tolerance d=4.

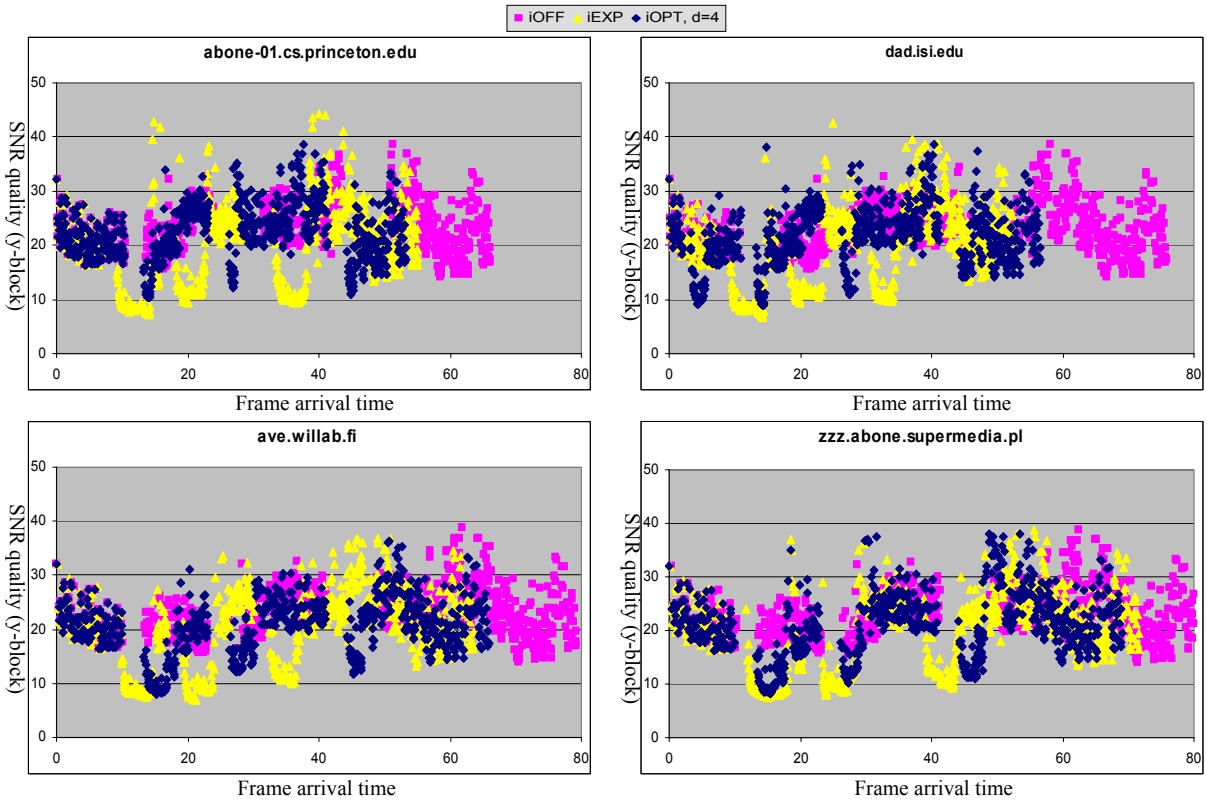

Figure 11. Frame Arrival time and frame SNR quality tradeoff for three running modes. X-axis frame arrival time, y-axis SNR quality (Y-block).

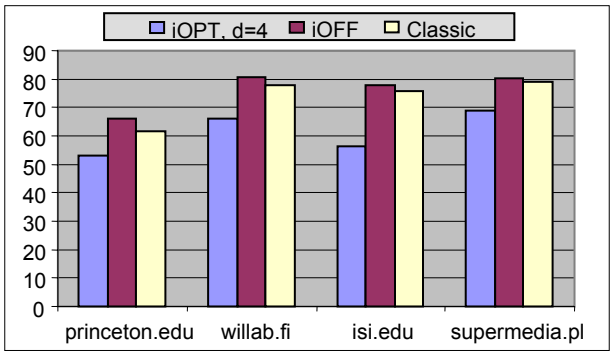

Figure 12. Overhead of the interactivity service.

Table 1. TCP Congestion Control Internal Events

Event	Meaning	Description	SSCA	FRFR	Sub
1	Retransmission timer timed out	Possibly congested network or the segment was lost.	X		X
2	A new ACK was received	Increment snd_cwnd either exponentially (if less than sstheresh) or linearly otherwise.	X		
3	snd_cwnd has reached the slow start threshold ssthresh	Switch incrementing snd_cwnd from exponential to linear.	X		
4	A third duplicate ACK was received	A segment was probably lost, perform fast retransmit.		X	X
5	A fourth (or more) duplicate ACK was received	One segment has left the network; we can transmit a new segment.		X	
6	A new ACK was received	Retransmitted segment has arrived at the destination and all out of order segments buffered at the receiver are acknowledged.		X	X

Table 2. Target ABone player nodes

Target ABone node	Country	RTT measurment				Number of hubs
		min	average	max	mean deviation	
ave.willab.fi	Finland	0.16355	0.16606	0.16647	0.798	24
zzz.abone.supermedia.pl	Poland	0.14705	0.14844	0.15701	3.023	23
abone-01.cs.princeton.edu	USA	0.03945	0.04002	0.04524	1.319	17
dad.isi.edu	USA	0.06548	0.06572	0.06610	0.186	19

Table 3. Experiment control flags and running modes.

Control flag	Effect				
iTCP	Turns on/off the interactivity service.				
EVENT	Turns on/off the event notification service.				
SYMB	Turns on/off the symbiosis feature of the transcoder. When this flag is set, the signal handler invokes the event handler to reduce the bit rate of the decoder. Otherwise, the signal handler just records the event type and time.				
OPT	Means (OPTimal mode). Used to choose between two modes of Symbiotic rate reduction (i) optimal backoff mode which uses the symbiosis throttling model described in section 4. or (ii) exponential backoff mode which uses a preset retraction rate and duration.				
Running mode	**Control Flags**				**Comments**
	iTCP	EVENT	SYMB	OPT	
iOPT	ON	ON	ON	ON	Full interactivity. Use the optimal backoff symbiosis throttling.
iEXP	ON	ON	ON	OFF	Full interactivity. Use the exponential backoff symbiosis throttling.
iOFF	ON	ON	OFF	X	Subscribe, report event, but do not change bit rate. Used to measure overhead.
Classic	OFF	X	X	X	Turn off all interactivity features.

Table 4. Average frame delay and acceptance ratio

mode		princeton.edu		isi.edu		willab.fi		supermedia.pl	
		Average Delay	Accept Ratio	Average Delay	Accept Ratio	Average Delay	Accept Ratio	Average Delay	Accept Ratio
d=2	iOPT	0.518	0.797	2.018	0.415	2.504	0.319	1.38	0.692
	iEXP	2.613	0.529	-0.015	1	-1.239	1	2.411	0.284
	iOFF	6.279	0.455	10.82	0.197	8.752	0.155	8.485	0.133
	Classic	3.047	0.461	10.957	0.217	6.615	0.273	8.485	0.147
d=4	iOPT	0.897	0.976	2.029	0.737	-0.641	1	0.727	1
	iEXP	2.613	0.529	-0.015	1	-1.239	1	2.411	0.777
	iOFF	6.279	0.455	10.82	0.197	8.752	0.293	8.485	0.277
	Classic	3.047	0.805	10.957	0.395	6.615	0.299	8.485	0.291
d=6	iOPT	0.883	1	3.974	0.679	1.623	1	1.387	1
	iEXP	2.613	0.997	-0.015	1	-1.239	1	2.411	1
	iOFF	6.279	0.455	10.82	0.329	8.752	0.295	8.485	0.277
	Classic	3.047	0.805	10.957	0.395	6.615	0.535	8.485	0.52

Table 5. Percentage of total bits delivered for each mode

	princeton.edu		isi.edu		supermedia.pl		willab.fi	
	Target Bits	Actual Bits	Target Bits	Actual Bits	Target Bits	Actual Bits	Target Bits	Actual Bits
iOPT, d=2	0.912	0.913	0.898	0.901	0.886	0.886	0.929	0.929
iOPT, d=4	0.966	0.966	0.892	0.897	0.814	0.826	0.789	0.791
iOPT, d=6	0.975	0.98	0.94	0.945	0.87	0.88	0.867	0.874
iEXP	0.843	0.843	0.835	0.835	0.862	0.862	0.86	0.86
iOFF, Classic	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 6. h_{best} and $T_{reovery}$ statistics for three ABone nodes.

d(sec)	event	princeton.edu		willab.fi		supermedia.pl		isi.edu	
		h_{best}	$T_{recovery}$	h_{best}	$T_{recovery}$	h_{best}	$T_{recovery}$	h_{best}	$T_{recovery}$
2	e1	512240	1.520135	1443311	15.68562	1333511	14.49148		
	e2	491954	1.459965	1292875	14.04808	1223663	13.29792		
	e3	496552	1.476599	1309004	14.22661	1257601	13.66691		
4	e1	279963	0.851075	602184	6.551785	665086	7.228908		
	e2	261526	0.792414	564819	6.145352	584324	6.355469		
	e3	259565	0.788820	604674	6.579283	602115	6.550372		
6	e1	186117	0.573669	486808	5.301540	467211	5.085250		
	e2	173629	0.525550	419186	4.557723	401019	4.368733		
	e3	172046	0.539606	307244	3.343913	411801	4.480751		