Technical Report 2003-02-02
Internetworking and Media Communications Research Laboratories
Department of Computer Science, Kent State University
http://medianet.kent.edu/technicalreports.html

Event Model and Application Programming Interface of TCP-Interactive

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 transport protocol can greatly benefit transport friendly applications. We envision a transport mechanism, which is interactive and can provide event notification to the subscriber of its communication service. We then show a friendly adaptive MPEG-2 video transcoding scheme, which directly interacts with the transport protocol and adjusts its production rate whenever a new event is received from the transport layer. We have recently implemented this concept system. The implementation has two components-- an interactive transport protocol over FreeBSD called iTCP and, a novel symbiotic MPEG-2 full logic transcoder. We have also experimented the real system on the Active Network (ABone) using selected nodes in the U.S. and Europe. In this report we present the performance of this system and report potential dramatic improvement in time-bounded video delivery. In this report we present the application accessible event model and the application programming interface that the application and the symbiotic component can use to interface with the interactive TCP.

1. Introduction

Congestion control for time-sensitive multimedia traffic has remained a difficult problem. Most of the mechanisms for congestion control those have been proposed to date are based on delaying traffic at various network points. The more classical schemes depend on numerous variants of packet dropping in network, prioritization (graceful delay in router buffer) admission control (delaying at network egress points), etc. However, a key aspect of a vast majority of these schemes is that they introduce **time distortion** in the transport pathway of applications. Though time distortion does no harm to time insensitive traffic such as email forwarding or ftp data, but they work completely against the application if the traffic is time sensitive such as multimedia streaming or control data.

For last few years it has been felt that for multimedia applications, the applications themselves have to be more integrated in the solution. Particularly promising are the research in the new TCP friendly paradigm [KeWi00, ReHE00, SiWo98, PrCN00]. [SiWo98] presented a TCP rate-based pacing mechanism that particularly takes note of document transfer characteristics. [ReHE00] discussed a general framework where applications can control rates based on their end-to-end measurements (similar end-to-end technique is used in RealPlayer). There are also fully application level proposals. Due to the lack of convenient means to obtain network states several works suggested [BrGM99, Wolf97] sending multilevel redundant information for video. Also several other works investigated combining application specific information from several streams into one clearinghouse architectures for aggregated congestion control. For example, Congestion Manager [ABCS00, BaRS99] is a system layer component. It provisions aggregated congestion control when multiple streams from the same end-point attempt to send via a separate program called Congestion Manager (CM). [SiWo98] proposed building TCP friendly application where application relies on real-time transport protocol (RTP) mediated end-to-end measurement. CM tries to minimize congestion by coordination between multiple sending streams. [PrCN00] used multiple probing mechanics for aggregate congestion control.

There has been several promising work on network or system level issues to increase TCP friendly-ness. Though, the paradigm of 'friendly applications' almost by definition shifts a major part of the congestion management responsibility to the applications, interestingly relatively very few work exists that seriously looked into the corresponding issues that arise in an actual time-sensitive application while taking advantage of the suggested 'friendliness'. The dynamics of the two systems can lead to stability issues. Time sensitive applications themselves have substantial complexity in adapting. Rate adaptation for any advanced multimedia

```
initially, cwnd = 1 (one segment);
win_size = min (cwnd, snd_wnd);
When congestion occurs:
   ssthresh = max(win_size/2, 2);
   if congestion was due to timeout
      cwnd = 1;
   for every ACK received:
      if (cwnd <= ssthresh)
         /* perform slow start */
         cwnd =  2 * cwnd;
      else
         /* perform congestion avoidance */
         cwnd = cwnd + segment_size;
```

Figure-1. Slow Start/Congestion Avoidance algorithm (SSCA).

```
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:
   cwnd = cwnd + 1;
   transmit a new segment;

When a new ACK arrives:
   /* one RTT after retransmission –
      fast recovery */
   cwnd = ssthresh;
```

Figure-2. Fast Retransmit/Fast Recovery algorithm (FRFR).

application in general is quite complex. It requires sophisticated layer 4+ techniques. It is highly unlikely that multimedia rate adaptations for any performance coding schemes (such as MPEG) can be performed at predominantly network or system layer.

It seems that an alternate strategy for time sensitive multimedia traffic should be the multimedia knowledge enriched rate control, which can work in symbiosis with the network condition. We have recently implemented an MPEG-2 ISO-13818-2 [ISO96, KYGP01, KhYa01, KhGu1] video streaming system, and a novel interactive version of TCP called TCP Interactive. The general principle we follow is simple and intuitive. It seems an effective delay conformant solution for time sensitive traffic may be built if the original data volume can be reduced by its originator-- the application[1].

However, a key element in any such scheme is that the application must be notified. Unfortunately, today's transport protocols do not support any interactivity with applications. It seems such non-interactivity has been inherited from the early days of networking interface research, when the applications were simple and did not require sophistication. In this report we will show that transport interactivity can bring major benefit to high performance and demanding applications. The particular scheme we propose here has the following novel aspects compared to other recent works:

- First, we suggest an active and direct notification mechanism by the underlying transport protocol, rather than using indirect end-to-end feedback tools. If there is any congestion, we propose an interactive transport protocol, which can directly notify the application.
- To demonstrate the efficacy of the principle, we have designed a corresponding video rate transcoder system that works in symbiosis with the network. This transcoder actively participates in a custom symbiotic *exponential-back-off and additive-increase* like scheme in application layer with deep application level knowledge. (This is also one of the first to our knowledge) resulting in much more effective joint quality/delay sensitive communication.
- The resulting scheme is similar in spirit to the TCP friendly approaches. However, there is a fundamental difference in how it is done. We expect network (or system) layers to remain as simple as possible. The means and techniques for rate reduction remain with the producer application. The responsibility of the network layer is simply to pass on only selected end-point events to the applications.

As, we will show the scheme is not only intuitive and simple, but also surprisingly effective compared to many other recently proposed schemes, which involve much more complex system/network layer reorganization.

The result presented in this report is not simulation; rather report from a real implementation of the concept system that we have completed very recently. The implementation has two components-- an interactive transport protocol over FreeBSD that we called iTCP and, a novel symbiotic MPEG-2 full logic transcoder [KYGP01, KhYa01], which is capable of working in tandem with the interactive transport. The transcoding model has been developed

[1] It is interesting to note, that the idea of application and network symbiosis have been mentioned for quite some time. However, almost no study exists which has focused on it.

by closely following the MPEG-2 Test Model 5 (TM5). MPEG-2 TM-5 signifies a real video coder with substantial complexity of itself. While the detail can be found in [Mpeg00], we describe the salient part of the rate control architecture that is critical to this symbiosis in [KhGR02].

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 window of the sender and the sender can send packets only as long as 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 timeout, the congestion window is reduced to one slot and TCP reenters 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) the arrival of duplicate ACKs. Two algorithms then contribute to the TCP congestion control behavior; these are the classic algorithm of slow-start and congestion-avoidance [Jac88], and the augmentation of fast-retransmit and fast-recovery [Jac90]. Figure-1 and Figure-2 below respectively shows the relevant details of the two algorithms.

2-2. Congestion Control Events

Table-1 below 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 these congestion control events.

In Table-1, the column labeled (**SSCA**) means that the event takes place in the Slow Start/Congestion Avoidance algorithm, and the label (**FRFR**) means that the event takes place in the Fast Retransmit/Fast Recovery algorithm. These events are also presented in Figure-3 below. The graph given in Figure-3(a) shows the sequence of events of the *SSCA* algorithm and how they affect the effective bandwidth available for TCP. Figure-3(b) shows the same sequence for the *FRFR* algorithm. However, in general design we expect only a subset of the internal events that constitutes a protocol will be of interest to the subscriber application. Only a subset of the internal events is made accessible via the interface. An application instance typically subscribes even to a

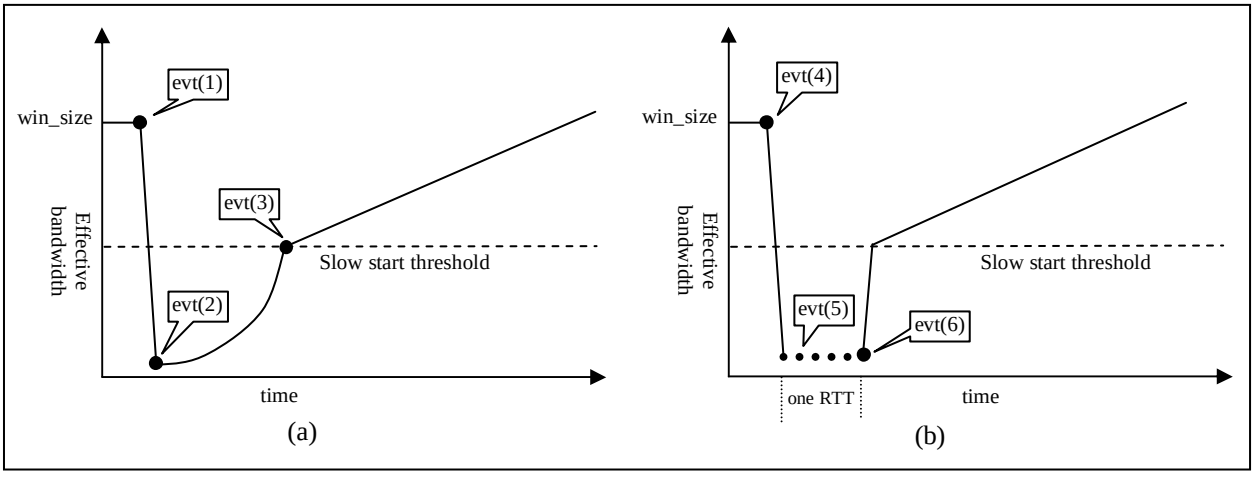

Figure-3. Effective bandwidth changes due to TCP congestion control internal events.

subset of these accessible events. In Table-1 the column (**Sub**) shows the subscribable events in our design.

3. Interactive TCP Model

3-1. Event Subscription

Figure-4 below shows the conceptual model of our interactive protocol. Once it establishes a TCP connection, the user process starts by binding the TCP kernel with a set of chosen events from Table-1 using a subscription API that extends the standard socket API. The model allows one user process to subscribe with multiple sockets and with different set of events per socket. The socket itself can support notification service for multiple subscribing processes. Each individual subscription is handled individually in the socket layer even if the same process made two or more subscriptions.

Each subscription binds the subscribed event with a predetermined user-supplied *Event-Handler*. The Event-Handler will be invoked as a user process when the subscribed event occurs in the kernel space. Also, the subscription mechanism itself is dynamic; it allows the subscribing process to subscribe to new events or cancel subscription (unsubscribe) to previously subscribed events at any time during the lifetime of the TCP connection.

3-2. Event Notification

The basic scenario of the event notification mechanism proceeds as follows: An entity called *Event-Monitor* runs in the iTCP kernel space and monitors all subscribed events for every socket (2).

Assume at some point event `(evt)` occurs in socket `(sock)`. The Event-Monitor sends a signal to the *Singnal-Handler* (3a), and at the same time it writes the socket descriptor of the socket `(sock)` in the process structure `proc{}` of every process that subscribed with this socket. Also, it marks all subscriptions of event `(evt)` in the socket `(sock)` as outstanding and need to be handled (3b).

When it receives a signal, the Signal-Handler must know first which socket generated the event. To do this, it uses the probing API to read the socket descriptor of the socket `(sock)` from the process structure. Then, it uses the probing API to access the socket `(sock)` and get the relevant information about the outstanding subscription of event `(evt)`. The information retrieved includes the event type and the name of the Event-Handler (4a,b). At this point the Signal-Handler is ready to invoke the appropriate *Event-Handler* (5).

3-3. Event Handling

There are *n* Event-Handlers, enough to satisfy all subscriptions made by the user process. Event-Handlers are usually small programs supplied by the user. One Event-Handler 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. reduce outbound bit rate to the transport layer).

Our probing API allows the Event-Handler to probe

Table-1. TCP's 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

additional information about the state of the TCP connection (6a,b). This information includes some parameters from the TCP control block such as: send window size (`snd_wnd`), congestion-control window size (`snd_cwnd`), threshold for slow-start (`snd_ssthresh`), current retransmit value (`t_rxtcur`), and round-trip time (`t_rtttime`). The Event-Handler can use some of these parameters to calculate a new sending rate that guarantees certain delivery time bound of the traffic. The model allows the Event-Handler to probe the Kernel any time to get the updated values of these parameters, and it allows the network administrator to restrict access to kernel data by each Event-Handler.

3-4. API

In Figure-4 we show two kinds of API that extends the socket API; subscription API which allows the application to subscribe with the interactivity service and the probing API which allows the application to probe the transport layer and retrieve relevant information about a subscribed event or about the TCP connection state in general. Table-2 shows the complete set of API system calls that we designed to support the iTCP model. For each system call we list its prototype, the level of its caller (user or system), its potential caller (application, signal handler, or event handler), and a brief description about its functionality. Some of these functions are designed for the network administrator (root process) to manage the subscription process by granting priority levels and access permissions for the user process.

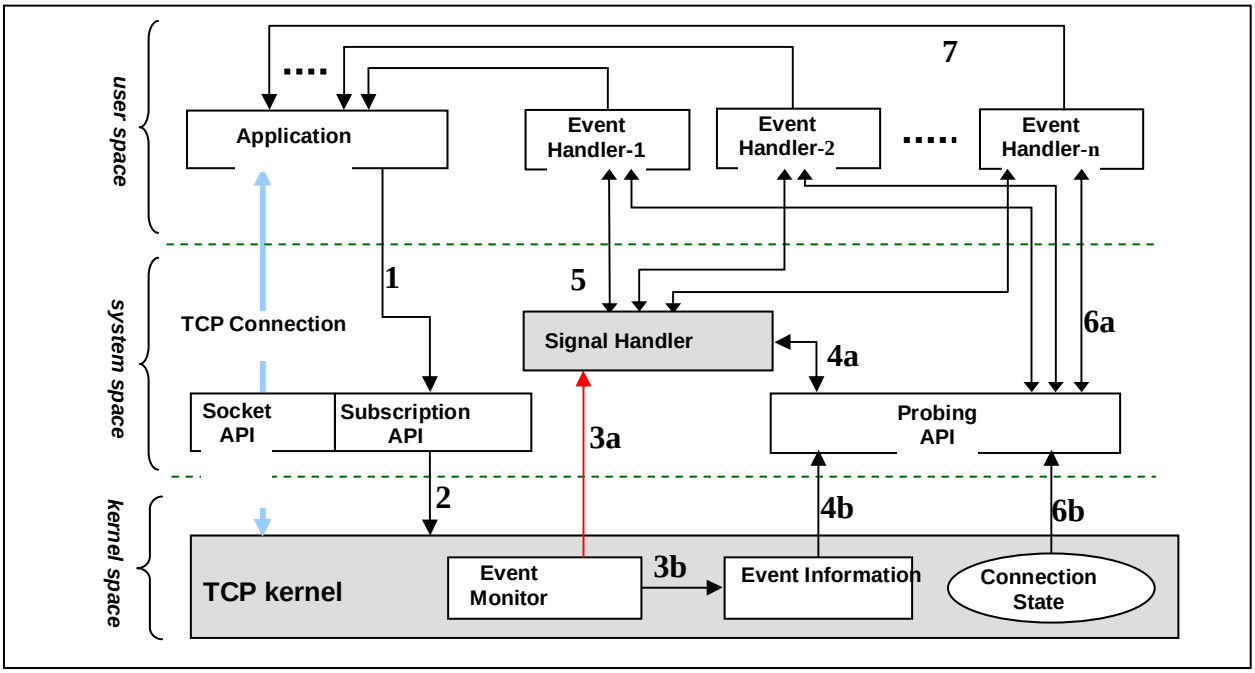

Figure-4. The TCP-interactive extension. The added registration API allows demanding applications to subscribe to events and probe additional event data.

Level	Pot. Caller	Description
`void GetEvents (int *NumOfEvents, evtInfo *EventList[]);`		
User	User process	Retrieve the complete list of available events in the TCP kernel. Retrieve **`evtInfo{}`** structure for each event in the list.
`int SubscribeEvt (int sock, int evt, char *e_hand);`		
User	User process	Subscribe with the socket **`(sock)`** for one event type **`(evt)`** to receive a signal when it occurs. Add a **`subInstance{}`** structure to the **`evtList`** linked list in the subscribed socket.
`int UnsubscribeEvt (int sock, int evt);`		
User	User Process	Unsubscribe a previously subscribed event. Afterwards, no signal will be sent when this event occurs. Remove the **`subInstance{}`** from the **`evtList`** linked list in the subscribed socket.
`int GetSockid (void);`		
System	Signal Handler	Get the descriptor of the socket that sent the signal when the subscribed event had occurred. This is necessary since a process can subscribe to many sockets, and the *Signal Handler* needs to know which socket triggered the event.
`int ProbeEvtSubInfo (int sock, struct evtSubInfo *info);`		
System	Signal Handler	Get the *number* and the *Event Handler* of the event that has just occurred in the socket **`(sock)`**.
`int ProbeSocket (int sock, struct connState *conn);`		
User	Event Handler	Probe the socket **`(sock)`** to retrieve the *current state* of the TCP connection is the **`connState{}`** structure.
`int GetSubPerm (int sock, int evt, int *perm);`		
User	User Process	Get the current access permission string **`(perm)`** for the event **`(evt)`** subscribed with socket **`(sock)`**. Get three flags: (*Read*, *Write*, and *Subscribe*) for two levels (*System* and *User*).
`int GetSubPriority (int sock, int evt, int *priority);`		
User	User Process	Get the *Priority Level* of the event **`(evt)`** subscribed with socket **`(sock)`**. Return 1 in **`(Priority)`** priority is lowest, or 3 priority is highest.
`int GetHandlerPerm (int sock, int evt, int *mask);`		
System	Root Process	Get the *Connection Access Mask* **`(mask)`** for the event **`(evt)`** subscribed with socket **`(sock)`**. The returned value in **`(mask)`** specifies which fields in the **`connState`** structure are accessible by the *Event Handler* and which fields are not.
`int SetSubPerm (int sock, int evt, int perm);`		
System	Root Process	Set a new access permission string **`(perm)`** for the event **`(evt)`** in the socket **`(sock)`**. The integer **`(perm)`** should specify three flags: (*Read*, *Write*, and *Subscribe*) for two levels (*System* and *User*).
`int SetSubPriority (int evt, int priority);`		
System	Root Process	Set a new *Priority Level* for the event **`(evt)`** in the socket **`(sock)`** by assigning a value to **`(priority)`**.
`int SetHandlerPerm (int sock, int evt, char *e_hand, int mask);`		
System	Root Process	Set a new access mask **`(mask)`** for the event **`(evt)`** in the socket **`(sock)`**. The integer **`(mask)`** should specify which fields in the **`connState`** structure are accessible and which fields are not.
`int GetEvtState (int evt, int *state);`		
User	User Process	Get the *Subscription State* of the event **`(evt)`**. Return *zero* in **`(state)`** if the given event is subscribable or *one* otherwise.
`int GetEvtWinEffect (int evt, int *effect);`		
User	User Process	Get the effect of the event **`(evt)`** on the *effective bandwidth* of the TCP connection. Return +1 in **`(effect)`** if evt opens the sliding window, -1 if it closes the window, or 0 if it has no effect on the window.
`int DelEvent (int evt);`		
System	Root process	Set the *deleted* flag in the **`evtInfo{}`** structure to *true*. Afterwards, this event will be ignored in any subsequent system call.
`int AddEvent (int evt);`		
System	Root Process	Reset the *deleted* flag in the **`evtInfo{}`** structure to *false*. Afterwards, this event will be reported in any subsequent system call.

Table-2. iTCP extension of the socket API.

4. Conclusions

In this report we have discussed the event model and the API of the TCP-interactive. This work is based on the FreeBSD-release 4.3 implementation of TCP. The event model traced those relevant events that occur when the TCP kernel invokes a congestion control mechanism. Some of these events signify packet loss due to network congestion like the *'retransmission timer out'* event and the *'third duplicate ACK'* event, while other signify recovery from congestion like *'a new ACK received'* event. The TCP-interactive traces those events and the connection state associated with them. To serve the application, we have designed a comprehensive set of API system calls. The API can be classified into three types; *(i)* extension to the socket API to subscribe and bind event with event-handler, *(ii)* probing API to enable the application to probe TCP kernel for event and connection data, and *(iii)* maintenance API for the network administrator to modify/set access permissions and event parameters.

This report only discusses the event model and the API of the TCP-interactive. It has been extensively tested for video streaming over ABONE and has demonstrated dramatic improvement in video QoS performance. The results can be seen in forthcoming technical documents.

The work has been supported by the DARPA Research Grant F30602-99-1-0515.

References

[ABCS00] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan, System Support for Bandwidth Management and Content Adaptation in Internet Applications, Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, OSDI 2000, October 23-25, 2000 San Diego, California.

[AlPa99] Allman, Paxson, et al. TCP Congestion Control, RFC 2581, April 1999.

[BaRS99] Balakrishnan, H., Rahul, H., and Seshan, S., An Integrated Congestion Management Architecture for Internet Hosts," Proc. ACM SIGCOMM, Cambridge, MA, September 1999.pp.175-187.

[BrGM99] Hector M. Briceño, Steven Gortler and Leonard McMillan, NAIVE--network aware Internet video encoding, Proceedings of the seventh ACM international conference on Multimedia, October 30 - November 5, 1999, Orlando, FL USA, Pp 251-260.

[BrOP94] Brakmo, L.S., O'Malley, S.W., and Peterson, L.L, TCP Vegas: New Technique for Cogestion Detection and Avoidance, Proc. SIGCOMM' 94 Conf. ACM, pp-24-35, 1994.

[GuTe98] Wetherall, Guttag, Tennenhouse, "ANTS: A Tool kit for Building and Dynamically Deploying Network Protocols", IEEE OPENARCH'98, San Francisco, April 1998. Available at: http://www.tns.lcs.mit.edu/publications/openarch98.html

[ISO96] Information Technology- Generic Coding of Moving Pictures and Associated Audio Information: Video, ISO/IEC International Standard 13818-2, June 1996.

[Jac88] Jacobson, V, "Congestion Avoidance and Control", Proc. SIGCOMM' 88, Conf, ACM, pp-214-329, 1988.

[Jac90] Jacobson, V., "Modified TCP Congestion Avoidance Algorithm," end2end-interest mailing list, April 30, 1990.

[KeWi00] Jun Ke and Carey Williamson / University of Saskatchewan, Towards a Rate-Based TCP Protocol for the Web, Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000.

[KhGu01] Javed I. Khan, Q. Gu, Network Aware Symbiotic Video Transcoding for Instream Rate Adaptation on Interactive Transport Control, IEEE International Symposium on Network Computing and Applications, IEEE NCA' 2001, October 8-10, 2001, Cambridge, MA, pp.201-213

[KHFH96] Keesman, Gertjan, Hellinghuizen, Robert Hoeksema, Fokke Heideman, Geert, Transcoding of MPEG bitstreams Signal Processing: Image Communication, Volume: 8, Issue: 6, pp. 481-500,

September 1996,

[KYGP01] Javed I. Khan, Seung Su Yang, Qiong Gu, Darsan Patel, Patrick Mail, Oleg Komogortsev, Wansik Oh, and Zhong Guo Resource Adaptive Netcentric Systems: A case Study with SONET-a Self-Organizing Network Embedded Transcoder, Proceedings of the ACM Multimedia 2001, October 2001, Ottawa, Canada, pp617-620

[KhY01] Javed I. Khan & S. S. Yang, Resource Adaptive Nomadic Transcoding on Active Network, International Conference of Applied Informatics, AI 2001, February 19-22, 2001, Insbruck, Austria, [available at URL http://medianet.kent.edu/, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet] (accepted).

[KhGR02] Javed I. Khan, Qiong Gu and Raid Zaghal, Symbiotic Video Streaming by Transport Feedback based quality rate selection, Proceedings of the 12th IEEE International Packet Video Workshop 2002, Pittsburg, PA, April 2002, http://www.pv2002.org .

[KPOY01] Javed I. Khan, Darsan Patel, Wansik Oh, Seung-su Yang, Oleg Komogortsev, and Qiong Gu, Architectural Overview of Motion Vector Reuse Mechanism in MPEG-2 Transcoding, Technical Report TR2001-01-01, Kent State University, [available at URL http://medianet.kent. edu/ technicalreports.html, also mirrored at http:// bristi.facnet.mcs.kent.edu/medianet] January, 2001]

[Mpeg00] MPEG-2 Test Model 5, [URL: http://www.mpeg.org/MPEG/MSSG/tm5/, last retrieved December 25, 2000]

[PeDa00] 19 L. L. Peterson and B. S. Davie. Computer Networks, 2nd edition, Morgan-Kaufmann, 2000.

[PrCN00] Prashant Pradhan, Tzi-cker Chiueh and Anindya Neogi Aggregate TCP Congestion Control Using Multiple Network Probing, Proceedings of the The 20th International Conference on Distributed Computing Systems, ICDCS 2000. 2000.

[ReHE00] Reza Rejaie,Mark Handley, Deborah Estrin, Architectural Considerations for Playback of Quality Adaptive Video over the Internet, Proceedings of the IEEE International Conference on Networks (ICON'00), 2000.

[SiWo98] Dorgham Sisalem, Adam Wolisz, Towards TCP-Friendly Adaptive Multimedia Applications Based on RTP, Proceedings of the The Fourth IEEE Symposium on Computers and Communications, 1998.

[Tene96] Tanenbaum, A. Computer Networks, 3rd edition, Prentice Hall, 1996.

[TSSW97] Tennenhouse, D. L., J. Smith, D. Sincoskie, D. Wetherall & G. Minden., "A Survey of Active Network Research", IEEE Communications Magazine, Vol. 35, No. 1, Jan 97, pp 80-86

[Wolf97] Bernd E. Wolfinger, On the potential of FEC algorithms in building fault-tolerant distributed applications to support high QoS video communications, Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing , 1997, Pages 129 – 138