Published in the Proceedings of the International Conference on Computing, Communications and Control Technologies – CCCT'04, Austin, TX, USA. August 2004. pp. 66 – 71, Vol. 7

Protocol Modeling with Transparent Networking

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

TCP friendliness is considered as an important concept in networked based application design. However, a particular problem with current networked systems is that it becomes very difficult to device TCP friendly solutions as the fixed part of the network layer itself has been designed as a closed box. We investigate an approach in which required service state information can be pulled-up to the upper layer where various custom friendly 'actions' can be performed by programmable application components, and the generated 'actions' are pushed down into the network layer via application network interactivity. This approach enables design of a new set of network interactive TCP friendly systems. We call this mechanism 'Transparent Networking'. In this paper, we explain the transparency service model and show by example how it can be used to model two well-known protocols proposed in the literature to improve TCP performance over wireless networks: Snoop [2] and WTCP [10].

Keywords: transparent networking, transparency service model, TCP interactive, TCP friendliness.

1. INTRODUCTION

Traditional network software stack has been designed with a layered (or more accurately pseudo-layered) organization. Each layer is intended to offer a specific service to the overall task of information communication between the application end-points. Each service has a specific implementation somewhere inside these layers. This organization used to be a blessing but no longer! These layers, the organization of the services and their specific implementations now turns out to be quite rigid and frustratingly immutable. As the diversity and the sophistication of applications have grown over the decade, this fixed and layered approach of network software organization is facing two levels of difficulties. In the first level, better solutions/implementations have been found for many of the services. A classical example is TCP's timer management and congestion control. Historically increasingly improved techniques have been found several times- and the process may continue each time speed differential increases in physical technology [4, 12]. This can be considered as the implementation's '*evolution*' problem. For example currently it is very difficult to build a jitter or delay managed transport service- though it appears as quite main stream for many applications. At a second level, problem also arises when selective applications critically require specialized service, while that very service can be undesirable to other applications. TCP's retransmission based implementation of reliability is one such classic case. Though, it is conceptually possible to implement reliability in other ways (such as FEC), the choice of this particular mechanism has been found to be severely detrimental for time-sensitive elastic traffic (such as streaming audio, video) [5]. This second level problem can be considered as the implementations' '*conflict*' problem.

The *transparent networking* approach–which we will explain shortly—might be able to solve both problems with moderate effort. In this approach, instead of embedding codes within deep network layers, we propose creating mechanisms only to pull up the required service state information in the upper layers. And then the actual action can be formulated by the programmable components running in the upper layers- even at the application layer and then to create handles so that the generated actions again can be pushed down below in the network layer. This relives the lower network layers from housing costly custom components- and to readdress complex issues regarding security and resource sharing. The attraction is that the application space already has a very well developed provision to run custom codes, share resources, and handle security issues for managing multiple trust domains, etc. Much of that can be reused.

Each communication service software component within a transparent network framework can easily be interactive –and thus can tell each other about their dynamic states. As we will show this transparent networking framework is also quite lightweight. With this design principle we have recently implemented a *transparent FreeBSD*. As an instance we have also implemented an interactive version of an otherwise legacy protocol TCP. We call it *TCP Interactive* (iTCP). Previously researchers have proposed smart solutions to several of the known network problems which required custom modification within network software layers. However, despite their functional advantages- each of these creative solutions faced deadly deployment problem because these required highly individualized changes within the network layers- which was never realized. We now demonstrate how few such instances based on TCP derivatives can be easily implemented at the application level and operated on demand within this new paradigm of transparent networking. These are the Snoop protocol proposed by Balakrishnan et al. [2], and the WTCP proposed by Sinha et al. [10].

Besides enabling application layer implementation path of custom network solutions, the proposed protocol transparency also offers a second type of benefit of no lesser implication. Transport friendliness has been preached for applications for quite some time by network researchers. Unfortunately the problem has much to do with the classical implementations of network protocols which themselves are not interactive or open to applications. It is very difficult for applications to be friendly with its non-friendly counterpart. In this context, the proposed encouragement in protocol interactivity can make way for a new generation of transport friendly applications and usher dramatically efficient solutions to currently notorious problems arising from network unawareness or ignorance. A good survey on TCP friendly protocols can be found in [12].

The paper is organized as follows: in section 2 we present the general framework of our transparency service model and iTCP. In section 3 we show how the interactive scheme can be used to model two well-known protocols proposed in the literature to handle traffic on wireless networks. In section 4 we discuss performance and security issues and we conclude in section 5.

2. TRANSPARENT NETWORKING SERVICE MODEL

In the logical sense, we envision every network layer as a service provider for the layer above it, and as a service user of the layer below it. For example, TCP layer provides its services to the socket layer above and uses the services of the IP layer below. In the interactive model, adjacent layers interact through a well-defined, clean interface. The interface supports four distinctive operations: (a) *subscribe,* (b) *signal,* (c) *probe*, and (d) *modify*.

An upper layer subscribes to target events in the layers below. Typically, a subscriber layer should be interested in certain events that might occur in the target layer, and by subscribing to events, the subscriber wishes to be notified when any one of the events has occurred. In real practice however, only a subset of the target layer events are subscribable. The transparency service is obliged to fulfill the subscriber wishes as far as it abides to the security restrictions and access privileges imposed by the super user.

In the next section we demonstrate the four operations of the transparency service by a general scenario.

A General Transparency Methodology

In the general framework, a Central Handler *(CH)* serves all signals from all layers. It maintains a list of all subscription instances for all subscribers and serves all probing and modification requests. Figure-1 demonstrates the service model through a simple case scenario where layer *L* wishes to subscribe for event *e* in layer *Q* below. Notice that layers *L* and *Q* are separated by layer *P* to emphasize that subscriber and target layers are not necessarily direct neighbors. *L* makes a *Subscribe* call (1) that includes: Subscriber *L*, Target *Q*, subscribed event *e*, and the Transientware (*T-ware*) module *T*. By making this call, layer *L* is basically telling *CH*: Whenever event *e* happens in layer *Q*, please activate module *T*. Since *CH* maintains subscription information of all subscribers, it adds this subscription to its database. Next, *CH* forwards the subscription request to the target layer *Q* (2). Also, *Q* adds a subscription instance for (Layer *L*/Event *e*) to its internal state. When event *e* happens, a signal is sent to the *CH* (3) which in turn probes *Q* (4) to get the event information. When it receives the event information, *CH* searches its database for the appropriate *T-ware* module that matches the instance (*L, Q, e*) and invokes it (5). Once invoked, the *T-ware* module can access relevant parts of the internal state of layer *Q* and possibly make changes to it as specified by the protocol being implemented. The *T-ware* module can use the *probe* or *modify* operations (6) to access/update the internal state of *Q* in accordance with its access privileges.

iTCP

In the past few years, we have been developing iTCP, a real interactive transport protocol based on the transparency service model. We let TCP track congestion control related events like '*retransmission timer time out*' event and '*third duplicate ACK*' event. Actually, both events signify packet loss and cause TCP to trigger a congestion control procedure. We extended the standard socket API with subscription and probing system calls to enable demanding applications to use the transparency service.

Figure-2 depicts the basic architecture of iTCP. Upon opening the socket, an adaptive application may bind a T-ware module to a designated TCP event by subscribing with the kernel. The binding is optional; if the application chooses not to subscribe, the system defaults to the silent mode identical to TCP classic. The logical sequence of operations follows the general transparency framework described in the previous section and proceeds as follows: (1, 2) subscribe, (3a, 3b) send a signal, (4a, 4b) probe kernel for event type, (5) invoke appropriate T-ware module to serve the event, (6a, 6b) probe/modify internal state, and (7) means that a T-ware module can also access or modify certain state variables in the user application if necessary.

We have experimented iTCP with elastic video traffic by allowing an adaptive video transcoder [7] to intercept the video stream and modify the generation bit rate based on feedback signals from iTCP. When the network is congested (a '*timer out*' event has happened) iTCP triggered a T-ware module which ordered the transcoder to reduce its generation bit rate. When congestion is dissolved, iTCP triggers another T-ware module which ordered the transcoder to recover and return to normal bit rate.

The scheme proved to be truly TCP friendly and have shown substantial gain in video performance metrics like frame-wise end-to-end delay and referential jitter. More information on iTCP and related experiments can be found in [5, 8, 6].

3. PROTOCOL MODELING EXAMPLES

In this section we briefly describe two well-knows protocols; Snoop protocol [2] and WTCP protocol [10]. They are among many other schemes proposed in the literature to improve TCP performance over wireless links. Then, we show how they can be modeled with our transparent networking scheme through API and

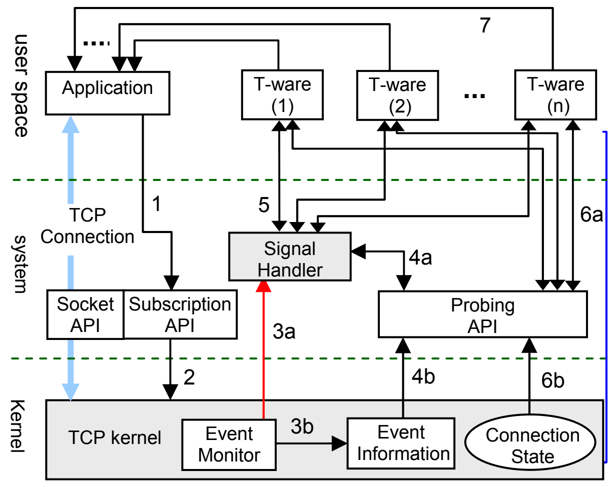

Figure-2. The TCP-interactive extension and API.

signaling extensions. This transparent modeling offers two potential benefits; (i) it becomes much easier to implement and deploy the proposed protocol on a real network, and (ii) new extensions or alternative algorithms –as application level T-ware modules—can be experimented with the new protocol without changing the underlying infrastructure. For example, a protocol like WTCP which was intended to improve TCP performance over wireless links can also be augmented with extra T-ware modules to add TCP friendly features.

3.1. TCP Performance over Wireless Networks

Wireless networks have certain characteristics that are not handled properly by regular TCP such as high bit error rate (BER) and long disconnections due to handoffs or bad reception. When a packet is lost, regular TCP assumes that it is due to congestion and will always trigger congestion control procedures at the fixed host. However, in a wireless environment, radio transmission errors or handoffs can also cause packet loss. This will result in significant reductions in throughput that can severely degrade overall performance. A good survey on proposed protocols for improving TCP performance over wireless networks can be found in [4, 1, 3].

3.2 Snoop Protocol

The Snoop protocol introduced a module, called Snoop, at the base station that monitors every packet that passes through in both directions. The Snoop module maintains a cache of TCP packets sent from the fixed host that have not yet been acknowledged by the mobile host. A packet loss is detected either by the arrival of duplicate acknowledgment or by a local timeout. To implement the local timeout, the module employs its own retransmission timer. The Snoop module retransmits the lost packet if it has it in the cache. Thus, the base station hides the packet loss from the fixed host, therefore avoiding its invocation of an unnecessary congestion control mechanism. Figure-

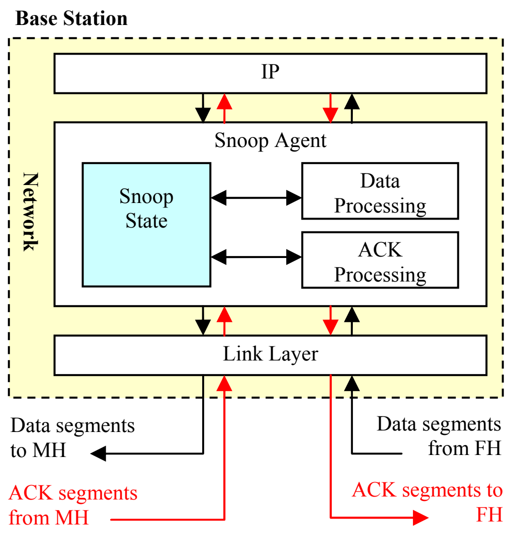

Figure-3. Conventional Snoop protocol mechanism.

3 describers the basic architecture of the Snoop protocol and figure-4 shows the interactive version of Snoop.

The scheme represents part of the snoop protocol that handles one direction of the traffic only (Data segments from FH to MH and ACK segments from MH to FH). The snoop protocol uses a different technique to handle traffic on the other direction, but it can be easily modeled with the interactive framework in a similar fashion. The model shown in figure-4, assumes that data segments are cached in the network as in conventional Snoop for performance reasons.

Whenever the *Interactive IP* layer receives a Data Segment from the FH, or an ACK segment from the MH, it sends a signal (software interrupt) to the application layer. A *Signal Handler* is immediately invoked to serve the signal upon its arrival. Based on the event type (Data or ACK received), the *Signal Handler* invokes the appropriate T-ware module either the *Data Handler* or the *ACK Handler*.

The *Snoop Agent* is a process that runs in the application layer. Its main role is to initialize and maintain the *Snoop State*, subscribe with the interactive service, and setup the Signal Handler. Afterwards, most of the work is done by the T-ware modules. The Snoop State is similar to the one used in the conventional snoop protocol. The Data Handler handles the (*Data segment received*) event. It implements the *Data processing* algorithm of the snoop protocol. The ACK Handler handles the (*ACK segment received*) event. It implements the *ACK processing* algorithm of the snoop protocol. Both algorithm are describes in detail in [2].

Both Data Handler and ACK Handler need to interact with the IP layer and to access the Snoop state; they use the special interactive API to (i) probe the IP layer and *Read* relevant header parameters from the TCP segment

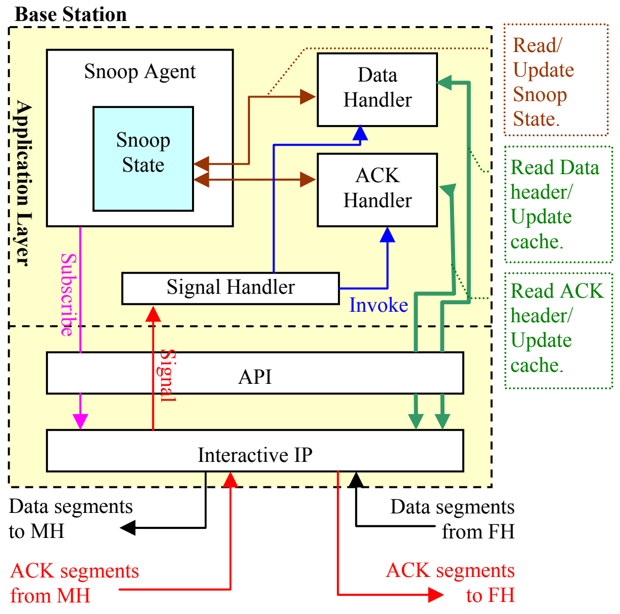

Figure-4. The interactive version of Snoop.

that has just arrived and (ii) to update the cache of TCP segments. The Data Handler adds segments to the cache and the ACK Handler clears the cache or part of it as decided by their respective algorithms. We assume that both handlers have full access to the Snoop State; they can read and update state variables as necessary.

3.3 WTCP

Wireless Transmission Control Protocol (WTCP) is specifically designed for wireless wide area networks. WTCP is based on the following two key principles: (i) it uses rate-based rather than window-based transmission control, i.e., it does not use ACKs for self clocking, and (ii) it uses the ratio of the inter-packet separation at the receiver and the inter-packet separation at the sender as the primary metric for rate control rather than using packet loss and retransmit timeouts.

WTCP uses a heuristic based on the average per-packet separation to distinguish congestion losses from random losses. In this heuristic, the receiver initially predicts that all losses are non-congestion losses. The following example from the WTCP original paper [10] explains the main concept of this heuristic: consider that packets i and j were received ($i < j$), but packets $i+1$ … $j-1$ were all lost. In this case the receiver computes the average inter-packet separation for each of the lost packets as:

$$perPktSep \leftarrow \frac{recvTime_j - recvTime_i}{j-i}$$

Where $recvTime_i$ is the time at which the last bit of packet i arrive. If the value of *perPktSep* is close to the measured inter-packet separation at the receiver (i.e. within the band [average − K • mean deviation, average + K • mean deviation], where K is a constant), then the receiver predicts that the losses were all random losses. Otherwise, the receiver predicts that there was at least one congestion loss, and the sending rate is reduced.

The basic scenario of the WTCP's rate-based scheme is shown in figure-5. The receiver computes the desired sending rate via its rate control mechanisms, and notifies this rate to the sender in the ACK packets. ACKs, thus, carry both reliability information (SACK) and rate control information. The sender monitors the reception of ACKs, and adjusts its rate accordingly. It also monitors the ACKs to tune the ACKing frequency, which it then notifies to the receiver in future data packets.

In figure-6 we show WTCP modeled with the interactive scheme. Here, we moved most of the processing to the application layer as T-ware modules, i.e., the rate control algorithm on the receiver (MN) and reliability algorithm on the sender (FH). The interactive extension provided the necessary API that allows TCP to trap events on both ends. On the MN, when a new packet is received, this event triggers the (inter-packet time computation) T-ware module, which calculates new timers and updates the internal state of WTCP. When it is time to perform the periodic update, this event triggers the (sender rate heuristic) T-ware module to calculate a new rate for the sender. The updated rate is transmitted to the sender through the API. On the sender side, when an ACK packet is received, two T-ware modules are activated, since the ACK packet carries both ACK and SACK information. The (SACK processing) module discovers holes in the transmitted packet sequence, i.e., discovers lost packets, and issues retransmission request through the API. The (ACK monitoring module) calculates a new ACKing frequency rate based on the current transmission rate and the internal state and sends the updated rate to the receiver periodically.

The scenario shown in figure-6 assumes that applications on both endpoints should subscribe with their respected events. We also assume that a signal handler on each end manages all signaling and activates T-ware modules.

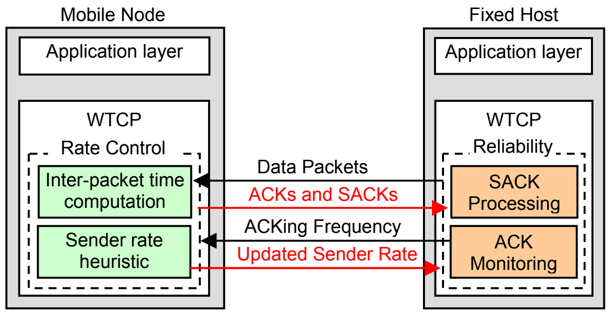

Figure-5. Conventional WTCP protocl.

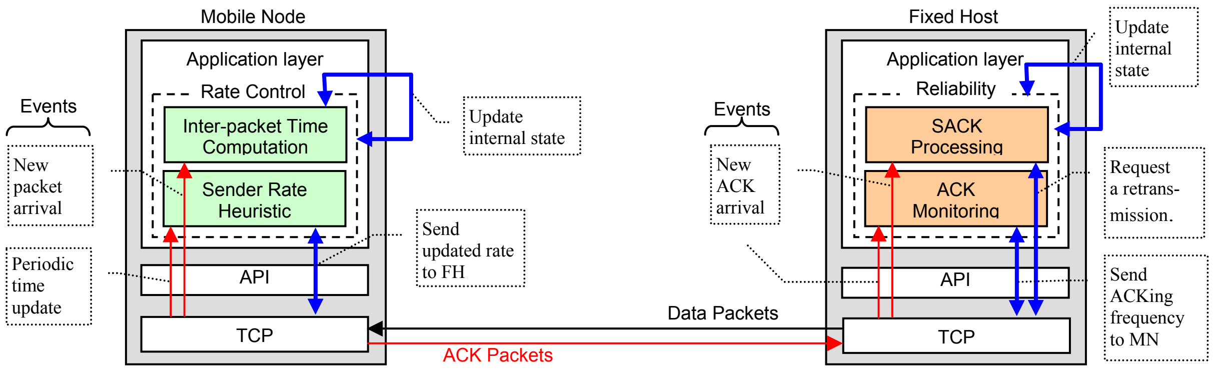

Figure-6. The interactive version of WTCP.

Scenario	Classic Snoop	Interactive Snoop (iSnoop)
Error-free, handoff-free wireless link	$SNOOP_{free} = N_{DAT}\ (C_{DAT} + U_n) + N_{ACK}\ (C_{ACK} + U_n)$	$iSNOOP_{free} = Sub + N_{DAT}\ (S + H + C_{DAT} + U_i) + N_{ACK}\ (S + H + C_{ACK} + U_i)$
Error-prone link with BER_x = 1 error / x Mbytes	$SNOOP_{free} + (T / BER_x)$	$iSNOOP_{free} + (T / BER_x)$
Handoff every n seconds	$SNOOP_{free} + C_{hoff}\ (8 \cdot T / n \cdot R)$	$iSNOOP_{free} + C_{hoff}\ (8 \cdot T / n \cdot R)$

Table 2. Algebraic overhead cost of Snoop and iSnoop in three scenarios of wireless link properties.

Name	Meaning
C_{ACK}	Overhead cost per ACK segment
C_{DAT}	Overhead cost per Data segment
N_{ACK}	Number of ACK segments
N_{DAT}	Number of Data segments
U_n	Update State/Cache cost in normal mode
U_i	Update State/Cache cost in interactive mode. We assume that $U_i > U_n$ since U_n might involve making a system call.
Sub	Subscription cost
S	Software Interrupt '*Signal*' cost
H	Signal Handler cost
R	Retransmit cost
T	Total transfer size (Mbytes)
C_{hoff}	Handoff cost
R	Wireless link bit rate (Mbps)

Table 1. Cost and link parameters for Snoop and iSnoop

4. Performance Issues

4.1. Overhead Cost

The transparency model implementation of both protocols adds some extra cost to the original scheme as a result of the added signaling and system calls overhead. Here, we show an abstract comparison of both interactive and conventional schemes of the Snoop protocol. In table-1 we show several quantities that define cost variables and wireless link characteristics. The first column in table-2 shows the estimated cost incurred by deploying the Snoop protocol for three scenarios: (1) error-free, handoff-free wireless link, (2) error-prone link with BER = 1 error for each *x* Mbytes, and (3) a moving mobile node that triggers a handoff every *n* seconds. The second column represents the interactive version of Snoop. In the first scenario (a reference case) iSnoop added overhead came from *Sub*, *S*, *H*, and U_i - U_n. Actually, in real practice these added costs should be very small (almost negligible). For example *Sub*, *H*, and U_i all involve running a small system call and OS context switch cost (around 10 *ns* on a 2 GHz machine). Besides the reference case, the other two scenarios were identical in both protocols. The same kind of analysis holds for the WTCP case, but we don't include it here for space limitations.

4.2. Security and Practice

The added small overhead cost can be justified for many practical gains allowed by the interactive model. As we mentioned earlier, since T-ware modules run in the application space, they will enjoy a well developed provision tuned to run custom codes, share resources, and handle security issues. Actually, the security issue is of great importance in such engagement. Running the Active modules inside the network raises many security concerns that usually require complex techniques to maintain acceptable security level and stability within the network domain. Moving these modules up to the application layer makes security management a much easier task. Actually, Subscriber applications and T-ware modules can only access internal network state through the API extension. Therefore, by imposing the appropriate access restrictions on each party, we can guarantee certain security level. Furthermore, since the API extensions are implemented as system calls, we can simply extend the OS security model and reuse available OS facilities like memory management and resource sharing to achieve even better performance. These characteristics make the transparency model an attractive and a practical choice to implement and deploy many useful protocols which thus far had been only simulated or tested on a small-scale controlled testbed.

4. Conclusion Remarks

We have particularly chosen two 'original source' examples for demonstrating an implementation path via transparent networking. But this is not to endorse them. The purpose is to show that these and many other solutions which required modification within network layers can be implemented at application layer as well with transparent networking. For the above two, please note because of their basic usefulness researchers have subsequently proposed several improved variants [1, 4]. The proposed transparency via interaction and triggered T-ware deployment will provide them implementation paths as well. In fact, since T-ware modules operate at the application layer it will be much easier to switch from one to another improved one.

5. References

[1] F. Anjum, and L. Tassiulas, "Comparative Study of Various TCP Versions Over a Wireless Link With Correlated Losses," IEEE/ACM Transactions On Networking, Vol. 11, No. 3, June 2003.

[2] H. Balakrishnan, S. Seshan, and R. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," ACM Wireless Networks, Vol. 1, 1995.

[3] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R.H. Katz, "A comparison of mechanisms for improving TCP performance in wireless networks," ACM SIGCOMM Symposium on Communication, Architectures and Protocols, Aug. 1996.

[4] H. Elaarag, "Improving TCP Performance over Mobile Networks," ACM Computing Surveys, Vol. 34, No. 3, Sep. 2002, pp. 357–374.

[5] J. Khan, R. Zaghal, and Q. Gu, "Symbiotic Streaming of Elastic Traffic on Interactive Transport," IEEE ISCC'03, Antalya, Turkey, July 2003.

[6] J. Khan and R. Zaghal, "Jitter and Delay Reduction for Time Sensitive Elastic Traffic for TCP-interactive based World Wide Video Streaming over ABone," Proc. of the 12th IEEE-ICCCN 2003, Dallas, Texas, Oct. 2003, pp.311-318.

[7] J. Khan and D. Patel, "Extreme Rate Transcoding for Dynamic Video Rate Adaptation," 3rd Int. Conference on Wireless and Optical Communication WOC 2003, Banff, Canada, July 2003, pp410-415.

[8] J. Khan and R. Zaghal, "Event Model and Application Programming Interface of TCP Interactive," Technical Report (TR2003-02-02), February 2003.

[9] Net100, "http://www.net100.org" The Net100 Project-Development of Network-Aware Operating Systems, 2001.

[10] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," Proceedings of ACM Mobicom'99, Seattle, WA, pp. 231–241.

[11] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A Survey of Active Network Research," IEEE Communications Magazine, Vol. 35, No. 1, pp80-86. Jan. 1997.

[12] J. Widmer, R. Denda, and M. Mauve, "A survey on TCP-friendly congestion control," IEEE Network, vol. 15, pp. 28-37, May-June 2001.