Journal of Computer Communications
Elsevier Science, May 2005, 28 (2005), pp702-711

Interactive Transparent Networking- Modeling Examples of Snoop and WTCP Protocols

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 *–The layered rigid organization of traditional network service stack currently poses a two fold problem: (i) better solutions/improvements cannot be practically deployed in the service stack –we call this the 'Evolution' problem, and (ii) applications are becoming more selective and critically demanding for specialized services –we call this the 'Implementation Conflict' problem. The 'Programmable' and 'Active' network paradigms tried to solve both problems by allowing arbitrary custom codes to be embedded inside network layers. We propose a less radical approach in which required service state information can be pulled-up to the upper layer where 'actions' are performed by programmable components, and generated 'actions' are pushed down into the network layer. This approach relives lower network layers from housing costly custom components and addresses other practical issues like security and flexibility. We call this mechanism 'Interactive Transparent Networking'.*
In this paper, we explain the mechanism and its advantages in creating TCP-friendly applications. We also show by example how it can be used as a protocol augmentation tool by modeling two well-known protocols proposed in the literature to improve TCP performance over wireless networks: Snoop [2] and WTCP [12].

Keywords: Active networks, transparent networking, transparency service model, TCP interactive, interactive modeling.

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. These layers, the organization of the services and their specific implementations has provided a simple and systematic framework to meet the needs of first generation internet application- however now it seems the situation has changed. The same organization now is widely felt to be 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. Historically increasingly improved techniques have been found several times- and the process may continue each time speed differential increases in physical technology [4], [16]. This can be considered as the implementation's '*evolution*' problem. 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 that allow temporal and spatial data quality tradeoff) [7]. This second level problem can be considered as the implementations' '*conflict*' problem. The problem is quite fundamental.

Solutions with varied doses of radicalism have been anticipated by the researches. In one end researchers have started 'softening' up particular layers. For example TCP is now being worked out for making them more tunable (Net100 project [11]). The paradigm of *programmable networking* took the approach one step deeper- it suggested making the API of not only the protocol end-points, but also that of router software stack running inside network to be made open. This can allow more intelligent scripts to be written to configure the underlying services. The paradigm of *active networking* has moved another step further [15]. It envisions almost arbitrary custom codes to be embedded inside network software layers which can program the content and the path of data being communicated.

In this research we present another approach which may be called *interactive transparent networking*. It seems

the idea for changing and embedding custom components inside network layer software comes from two realities- first we are seeing that the data in any communication pathway is being required to be processed in custom and adaptive ways. No finite set of pre-agreed '*fixed*' protocols may ever foresee and satisfy the cases. Secondly, these customizations are very dynamic and communication case specific. Many of the triggers for the accompanying custom actions originate right inside network software. Thus, allowing programmable modules to be embedded inside '*fixed*' network software solves both the problems.

However, it seems a less radical approach of **interactive transparent networking**, which we will explain shortly, might be able to achieve similar functional capability- with perhaps much lesser network layer complexity. The idea is to make network service implementations interactively accessible. This unique feature can allow new type of application level codes to augment, alter and influence the lower level functionalities of the protocols in the same way as they are embedded inside. This results in a surprisingly low cost path for implementing and upgrading new protocols. It also enables a new generation of powerful network friendly applications.

The paper is organized as follows: in the following section we first present the interactive transparent networking paradigm then we show how protocol functionality can be modeled to prepare it for interactive transparency –we use TCP as a prime example. In subsequent sections we present three powerful case solutions to demonstrate the capabilities of this paradigm. The first one is the example of a new type of transport friendly application- a network aware video rate transcoder. This system can take advantage of protocol state information by direct event feedback and generate adaptive application response- with much lower feedback delay than possible in classical network paradigm. The second and the third are examples of custom enhancement of TCP/IP –enacted at application layer. For these later two we select two interesting TCP/IP enhancements which, until now have faced hard time finding any deployment path in standard TCP/IP world. We demonstrate that based on interactive transparent networking they can be easily implemented at application layer.

2 Interactive Transparent Networking

Instead of embedding custom codes within network layer the paradigm of interactive transparent networking proposes creating mechanisms only to pull up the required service state information to 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 custom components and to re-address complex issues regarding security and resource sharing. The attraction is that the application space already has very well developed provision for housing custom codes, share resources, and handle security issues for managing multiple trust domains, etc. Much of that can be reused.

What is the organizational distinction? In contrast to creating a network software organization which can house custom code as proposed in active networking, this requires creating a network software organization which can facilitate service state information exchange. Unfortunately, the current network software components lack this facility. During the early days of design, the over emphasis of hiding network related details from user applications has resulted in extremely non-transparent legacy design.

We call a network software system which can support this new state accessibility as interactive transparent networking. 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 deployment problem because they required highly individualized change within the network layers. Most in fact could never be realized –despite their demonstrated benefit. 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. [12].

Natural concerns of any programmable networking paradigm are performance and security. Since these

State	Description
1	Normal mode communication
2	First duplicate ACK received
3	Second duplicate ACK received
4	Third duplicate ACK received – Fast Retransmission
5	Timed out (1T)
6	Fast Recovery: Send a new segment for every duplicate ACK received
7	New ACK received, shrink win to half
8	Drop win size to 1 segment
9	Slow start: Increase win exponentially
10	Congestion avoidance: Increase win linearly
11	Exponential backoff (2T)
12	Exponential backoff (4T)
13	Exponential backoff (8T)
14	Exponential backoff (16T)
15	Exponential backoff (32T)
16	Exponential backoff (64T)

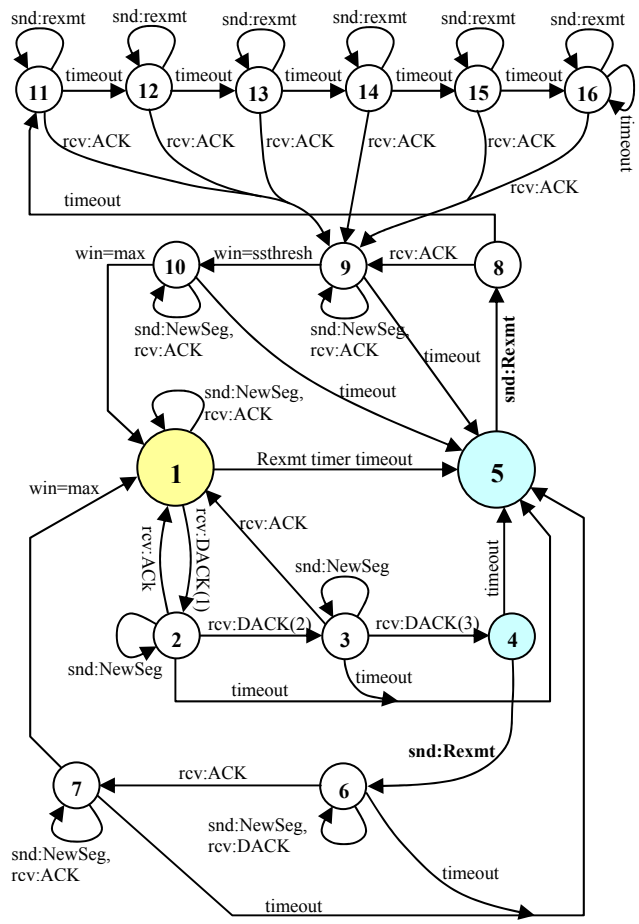

Figure-1. A finite state machine modeling the congestion control mechanism of TCP.

solutions are operating at application level, are they too slow? What happens to the security issue if network states are more accessible? In this paper we will also include discussion on how the security and resource sharing issues probably take a much manageable proportion as opposed to full active network approach.

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- is a proposal to make the very networks itself friendly toward the applications. Consequently it 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.

The proposed interactive and transparent networking requires three fundamental steps. The first step towards interactive transparent networking is to capture the internal operation of a target protocol in some form of event/state based language. The classical protocols did not require this formalization. But a fundamental step towards transparency is to obtain the functional model. We call this stage protocol's *operational modeling*. The second step is to provide subscription facility so that a set of events and states identified in the operational model can be accessible to service subscribers like network or application components by some uniform interface. The third step is to create appropriate operating system facilities so that custom application

layer modules can efficiently interact with these transparent network service components as designed based on the events as programmed. The following three subsections illustrates these steps.

2.1 Protocol's Operational Modeling

In figure-1 we have modeled the congestion control mechanism of TCP-Reno as a finite state machine. Actually, all transitions in this diagram occur inside the '*Established'* state of the general connection management finite state machine of TCP [14]. The lower part of the diagram represents the '*Fast Retransmit/Fast Recovery*' Algorithm [5] which is invoked whenever packet loss is detected by receiving three duplicate ACKs, and the upper part represents the '*Slow Start/Congestion Avoidance*' Algorithm [6] which is invoked –in addition to initial bandwidth discovery— whenever packet loss is detected by a retransmission timer '*timeout'* event. The upper part also includes the '*Exponential backoff*' mechanism of TCP which reacts to multiple consecutive timeout events. In the diagram we can see 15 arrows labeled '*timeout*' and they all represent a retransmission timer '*timeout'* event. Eight of these '*timeout*' arrows lead to state (5) which means: the retransmission timer has timed out once, retransmit lost segment, drop window size down to 1 segment, and invoke the slow-start procedure. The remaining seven '*timeout*' arrows are all part of the exponential backoff sequence. We brought up this statistics to highlight the importance of the '*timeout*' event in TCP dynamics and its major role in causing a state transition in almost every state in the diagram.

Interactive transparent networking service makes such *events* and *state transitions* accessible by demanding subscribers –e.g. user application. In fact, our transparency service also allows the subscriber to modify the internal state of the protocol. In general, the transparency service user (e.g. application level user program) and the target protocol (e.g. TCP) 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 protocols below. Typically, a subscriber layer should be interested in certain events that might occur in the target protocol, 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 few sections we describe the interactivity architecture and demonstrate with a general scenario the four operations of the transparency service.

2.2 Interactivity Architecture

The general architecture of the centralized interfacing is given in Figure-2. All network layers including the application layer send their subscription requests (S_{app}, S_n, S_{n-1}, .. S_1) to a *Central Handler (CH)*. The *CH* also receives all event signals (K_n, K_{n-1}, ..., K_1) from all layers. Notice that the application layer does not send any signals since it is only a service user (it does not provide a service for a layer above it). The *CH* also activates the appropriate event handler (T-ware code) to serve events in each layer.

This scheme obviously incurs little overhead in terms of signaling cost. But the *CH* can be a bottleneck if several events happen simultaneously. If we assume that the CH

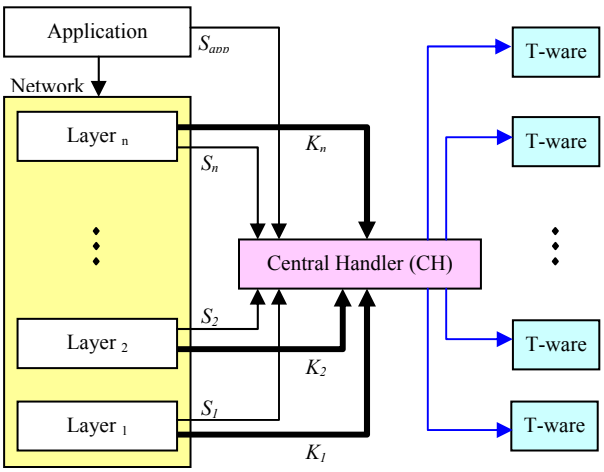

Figure-2. Multi-layer event subscription and signaling architecture with a centralized handler.

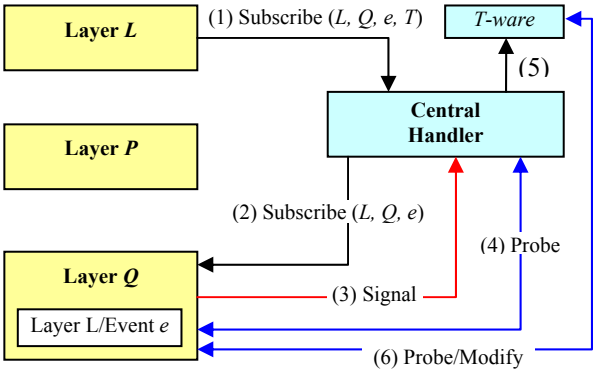

Figure-3. The sequence of operations in a general interactive service model scenario.

uses a FCFS queue, some events may experience some delay before they can be served.

2.3 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-3 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-war*e 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.

2.4 Interactive-TCP (iTCP)

In the past few years, we have been developing iTCP, a real interactive transport protocol based on the transparency service model. We installed a handler in TCP to track congestion control related events: '*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. Table-1 shows a sample of iTCP's API extensions. To see the complete list, please refer to our technical report in [10].

Table-1. Selected API functions of iTCP

Level	Caller	Description
`void GetEvents (int *NumOfEvents, evtInfo *EventList[]);`		
User	User process	Retrieve the complete list of available events in the TCP kernel. Retrieve **`evtInfo{}`** struct for each event in the list.
`int SubscribeEvt (int sock, int evt, int T-ware);`		
User	User process	Subscribe with the socket **`sock`** for event type **`evt`**. Register handler **`T-ware`** for this event. This call will add a **`subInstance{}`** structure to the **`evtList`** 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`** 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 several sockets, and the *Signal Handler* needs to know which socket triggered the event.
`int ProbeEvtInfo (int sock, struct evtInfo *info);`		
System	Signal Handler	Get the *number* and the *Handler* name 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.

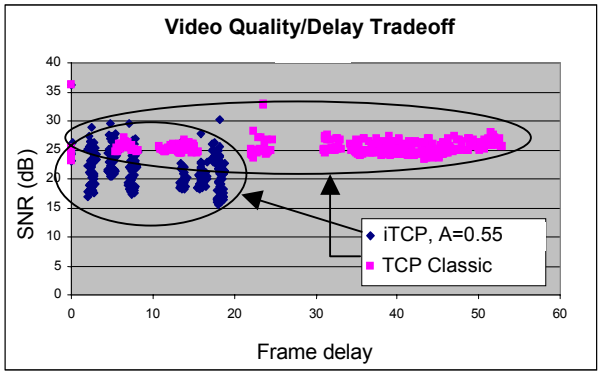

Figure-5. The two clusters show the quality, delay tradeoff offered by the iTCP. The iTCP dramatically reduced frame delivery delay by controlled trade-off of the SNR quality.

Figure-4 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.

3 A Network-Friendly Application

3.1 Video Transcoding with iTCP

In this section now we provide an example of a network friendly application. This is a video rate transcoder [9] which can automatically, and in real- time adjust the video rate to match the dynamic changes in link capacity. The system can work on elastic video traffic and perform highly sophisticated rate/quality adaptation based on custom perceptual video compression theory. It can respond in millisecond resolution. The idea of video rate adaptation has been proposed before- but unfortunately- on traditional network the difficulty is to sense the network speed. As being application unfriendly, the network protocols do not provide any feedback to applications about its state of service. To circumvent the problem within the TCP-friendly application paradigm some researchers have proposed end-to-end measurement based sensing using RTP [13]. Unfortunately, the round trip delay involved in RTP does not allow formulation of adaptive response in

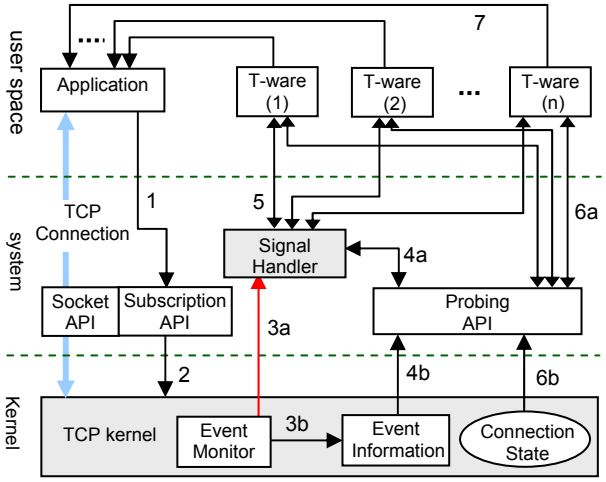

Figure-4. The TCP-interactive extension and API.

millisecond resolution. With iTCP the solution becomes interestingly quite easy. For this purpose we have developed a T-ware driven perceptual video transcoder.

3.2 The Scheme

The video transcoder works in symbiosis with iTCP as follows: when the network is congested (a '*timer out*' event has happened) iTCP triggers a T-ware module which orders the video transcoder to reduce its generation bit rate. When congestion is dissolved, iTCP triggers another T-ware module which orders the transcoder to recover and return to normal bit rate. The first T-ware used the '*probe*' operation to retrieve recent state parameters from TCP (e.g. '*round trip time*' and '*congestion window size*') and used them to calculate a new generation bit rate. Figure-xx shows the result

3.3 Performance

The scheme proved to be effective and have shown substantial gain in video performance metrics like frame-wise end-to-end delay and referential jitter. Figure-5 shows a sample of iTCP performance which highlights the tradeoff offered by iTCP. Here, each video frame is plotted as a point in the video quality/frame delay plane. As can be seen from the region of the two QoS distributions, in classical TCP, although frames have been generated with SNR quality ranging between 22-29 dB, but many of these frames were lost in transport, and were never delivered. In contrast, the proposed iTCP (with a bit rate reduction ratio A=0.55) delivered all the frames with 15-17 delay guaranteed at 15-26 dB quality. Fundamentally, what T-ware solution has offered is a qualitatively (as opposed

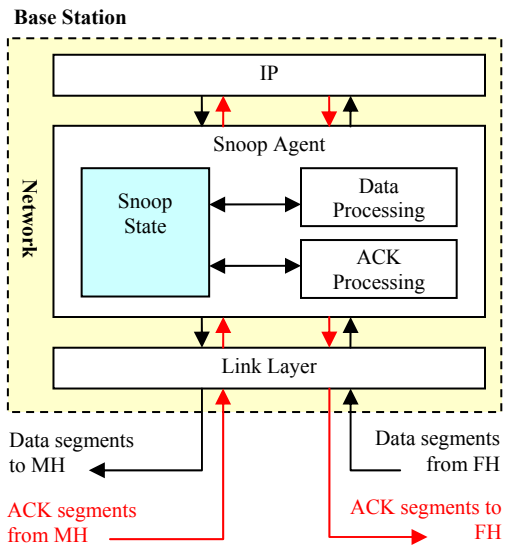

Figure-6. Conventional Snoop protocol mechanism.

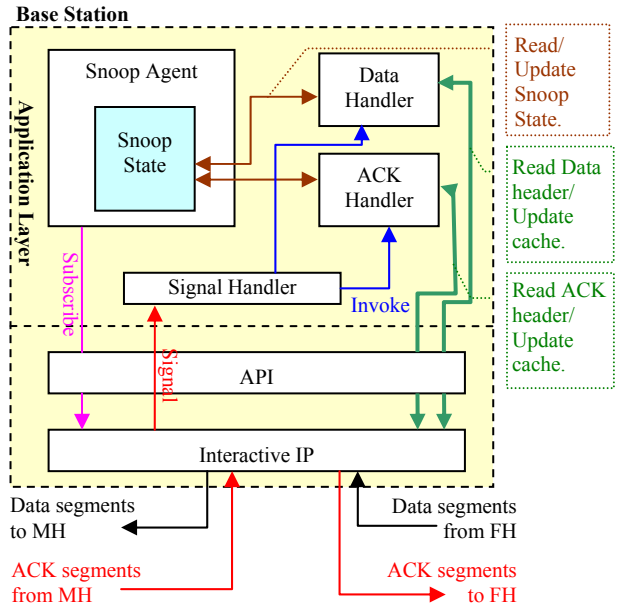

Figure-7. The interactive version of Snoop.

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. More on iTCP and related experiments can be found in [7], [10], and [8].

The scheme has a number of characteristics. First it avoids duplication of effort; Estimates (such as round trip delay or jitter) – which are the building block for adaptive response are often anyway kept by the communication protocol's local end-point. Unfortunately, in classical networking though those are there but are not accessible to applications; rather, applications have to re-estimate them by applying their own mechanism. The interactive transparency in underlying iTCP made this local information readily available.

For some application this artificial boundary- which process should and should not have access to these estimates have even further negative consequences. Consider the issue of response time; End-to-end application layer measurements may take time and quite often have to wait additional cycle before estimate is complete. While solutions demonstrated in this example, which depends on a simple function call to a local end-point of the involved protocol—the same information can be obtained much faster –almost at the very time when packet loss has been detected. However, it is worth clarifying that if an adaptive response requires a custom estimate- which is not kept by the involved protocols by its own- have to be re-estimated anyway by other means even if interactive transparent version of the protocol is used.

4 Protocol Extension and Augmentation

In this section we briefly describe two well-knows protocols; Snoop protocol [2] and WTCP [12]. They are among many other schemes proposed in the literature to improve TCP/IP performance over wireless links. Then, we show how they can be implemented with our transparent networking scheme through API and signaling extensions.

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 [1],[3], and [4].

4.1 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-6 describers the basic architecture of the classic Snoop protocol and figure-7 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-7, assumes that data segments are cached in the network as in conventional Snoop for performance reasons.

Assuming that the 'Snoop Agent' shown in figure-7 has subscribed with the 'Interactive IP' protocol for two

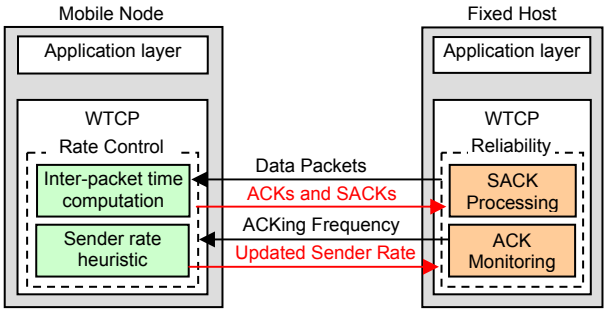

Figure-8. Conventional WTCP scenario.

events: an ACK received from MH event (evt_ACK_MH), and data segment received from FH event (evt_DAT_FH). Whenever any one of these two events occur, a signal (software interrupt) is sent 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 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.

4.2 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 [12] 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.

Table-2. iWTCP internal events

Event	Node	Meaning
1	MN	New packet has arrived at MN.
2	MN	Periodic update timer expired. Time to send updated rate to sender.
3	FH	New ACK has arrived at FH.

Table-3. Cost and link parameters for Snoop and iSnoop

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)

The basic scenario of the WTCP's rate-based scheme is shown in figure-8. 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-9 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

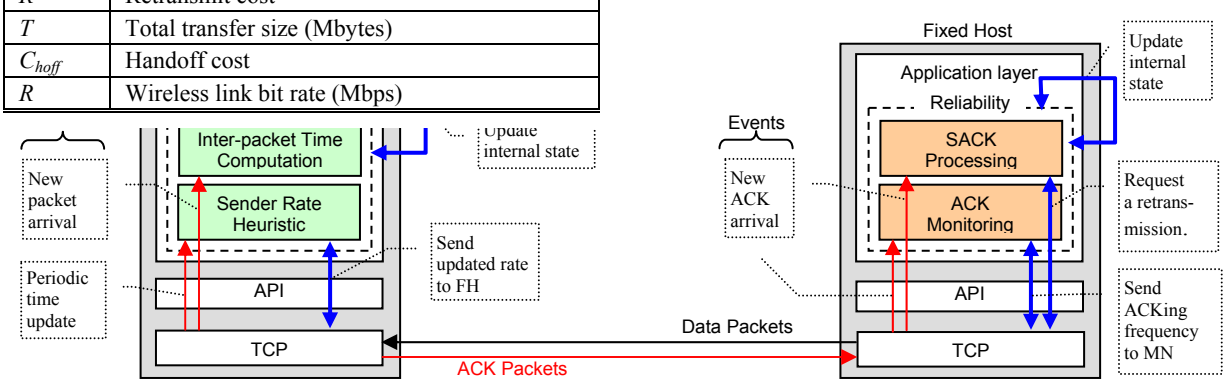

Figure-9. The interactive version of WTCP (iWTCP).

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-9 assumes that applications on both endpoints should subscribe with their respected events. The three events are summarized in table-2. We also assume that a signal handler on each end manages signaling and activates T-ware modules.

5 Performance Issues

5.1 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-3 we show several quantities that define cost variables and wireless link characteristics. The first column in table-4 shows the estimated cost incurred by deploying the Snoop protocol for three wireless link 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. 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.

To get a more realistic figure about interactivity overhead cost we performed a simple experiment on iTCP. We used `getrusage()` –a utility in FreeBSD—to collect statistics about system resources used by one invocation of the Signal Handler and a T-ware module (see figure-4) on 1.6 GHz processor. The information returned by `getrusage()` is shown in table-5. Both routines made four system calls from the iTCP API set which took 5.645 ms. The total time (system + user time) need by both routines is 22.582 ms. This includes all the time needed from the moment the signal handler catches a SIGIO signal reporting iTCP event, until the T-ware module finishes serving the event. Another interesting observation is the number of signals received –the signal handler has caught 149 signals! This is because the signal handler was programmed to catch all SIGIO signals sent by the kernel, and these happen to be many since the TCP kernel sends a SIGIO signal whenever new data is ready to be picked by a receiving application or when TCP is ready to accept new data from a sending application. Actually from these 149 signals only *one* signal was iTCP related (i.e. reported a subscribed event) and the rest were simply ignored. In a future, fine-tuned implementation this shortcoming can be remedied by creating a special purpose signal for interactivity use only. Nevertheless, the time overhead (22.58 ms) is still very small if weighted against gain in performance.

5.2 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

Table-4. Algebraic overhead cost of Snoop and iSnoop in three scenarios of wireless link properties.

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-5. System resources usage of one invocation of iTCP signal handler and T-ware.

Resource	Usage
User time used	16.937 (ms)
System time used	5.645 (ms)
Number of context switches	510
Number of signals received	149

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 managements 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.

6 Concluding Remarks

The interactive transparent networking can offer two potential benefits, (i) it becomes much easier and practical to implement and deploy protocol enhancement on a real network, and (ii) new extensions or alternative algorithms –invoked 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. The proposal does not call for any functional change in the existing protocols. It only requires making them more accessible and adding event notification service. Thus, interactive transparent networking remains fully compatible with its classical counter part. It also does not change the communication dynamics- and thus will have little impact on the various optimization technique built on them in various existing protocols.

An interesting formal question is if there is functional equivalence between the paradigms of active network and the proposed interactive transparent networking? Any system which can be realized on active network- can it also be realized on the proposed paradigm where software components in network layer are interactive? The formal answer is yet to be established, but the overlap seems quite strong.

We have particularly chosen two 'original source' examples for demonstrating an implementation path via transparent networking. But this is not to endorse them. 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.

7 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] V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.

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

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

[8] 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.

[9] 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.

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

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

[12] 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.

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

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

[15] 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.

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