# End-to-End Transmission Control by Modeling Uncertainty about the Network State

Keith Winstein and Hari Balakrishnan
*M.I.T. Computer Science and Artificial Intelligence Laboratory, Cambridge, Mass.*
{keithw,hari}@mit.edu

## ABSTRACT

This paper argues that the bar for the incorporation of a new subnetwork or link technology in the current Internet is much more than the ability to send minimum-sized IP packets: success requires that TCP perform well over any subnetwork. This requirement imposes a number of additional constraints, some hard to meet because TCP's network model is limited and its overall objective challenging to specify precisely. As a result, network evolution has been hampered and the potential of new subnetwork technologies has not been realized in practice. The poor end-to-end performance of many important subnetworks, such as wide-area cellular networks that zealously hide non-congestive losses and introduce enormous delays as a result, or home broadband networks that suffer from the notorious "bufferbloat" problem, are symptoms of this more general issue.

We propose an alternate architecture for end-to-end resource management and transmission control, in which the endpoints work directly to achieve a specified goal. Each endpoint treats the network as an nondeterministic automaton whose parameters and topology are uncertain. The endpoint maintains a probability distribution on what it thinks the network's configuration may be. At each moment, the endpoint acts to maximize the expected value of a utility function that is given explicitly. We present preliminary simulation results arguing that the approach is tractable and holds promise.

## CATEGORIES AND SUBJECT DESCRIPTORS

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design — Network communications

## GENERAL TERMS

Design, Performance

## 1. INTRODUCTION

Over the past several years, the bar for a new subnetwork technology to become a first-class part of the Internet has risen sharply. Previously, any network capable of sending unfragmented 20-byte packets could be part of the Internet. Today, the tacit requirement is that TCP perform well.

Network resource management, which can be grouped into functions performed end-to-end and those done in the net, has converged to a "fixed point" at the endpoints: TCP or TCP-compatible congestion control. As a result, in-the-net functions — for traffic engineering, packet scheduling, queue management, rate adaptation, or stochastic loss handling — must be engineered with TCP in mind.

This requirement is considerably harder to meet, not least because TCP congestion control does not have an end goal that is easy to state. Nor does TCP have well-understood dynamical behavior between many hosts contending for network resources, and there is no deployed mechanism to inform TCP about unconventional details of a network.

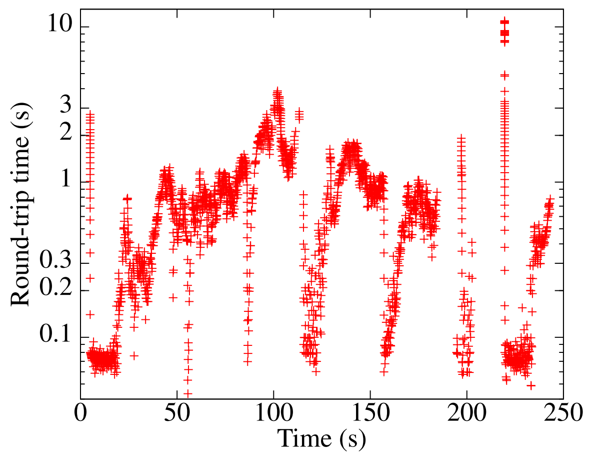
TCP generally models those behaviors of the Internet that were exhibited in the 1980s, such as gateway buffer overflow (represented in TCP's congestion window), overall round-trip time (RTT), and delay jitter (tacitly assumed to be a lightly-tailed distribution, like a Gaussian) [16]. Newer phenomena exhibited by wireless networks — such as variable bit rates, intermittency and fading, and stochastic loss not caused by congestion — are not accounted for in TCP's model.

As a result, the need to accommodate TCP is distilled to the subnetwork designer as a set of loose guidelines: e.g., hide non-congestive packet losses, limit congestive packet loss rates to no more than a few percent, prevent intra-flow packet reordering, carefully provision queue sizes to be neither too large nor too small, and pay attention to TCP dynamics in queue management and scheduling. In fact, RFC 3819 from the IETF provides 60 pages of "best current practice" advice to Internet subnetwork designers [18], much of which focuses on how to make subnetworks "play well" with TCP.

We contend that this "fixed point" has hampered the evolution of interesting in-the-net behavior. More importantly, these requirements have led to poor end-to-end performance in many important subnetworks (perhaps because it isn't easy to distill best practice into a succinct form). Many of today's wide-area cellular networks zealously hide non-congestive losses, and as a result can introduce delays of 10 *seconds* or more, confounding TCP (Figure 1). Home broadband networks are often provisioned with gigantic buffer sizes, an old problem now known as "bufferbloat."[1] Large buffers can enhance the performance of a single TCP flow, but at a cost to

[1]http://www.bufferbloat.net/

**Figure 1: Round-trip time during a TCP download on the Verizon LTE network in Cambridge, Mass., Oct. 14, 2011 at 3 p.m.**



latency-sensitive traffic and new TCP flows that must compete with an incumbent flow that has already filled the buffer.

We propose an alternate approach for end-to-end resource management and transmission control. Instead of writing down rules for the endpoints to follow, as TCP does, we start from the desired behavior and work backwards. The endpoint maintains a flexible model of its uncertainty about the characteristics of the network between source and destination. As the sender receives acknowledgments from the receiver, it infers whatever it may to reduce this uncertainty, e.g., about the link speed, buffer size, volume and behavior of cross traffic, amount of jitter, or even the topology of the network itself. For example, in this framework it is easy to accommodate both congestive loss and stochastic loss without becoming confused as TCP does.

At each moment, the endpoint acts to maximize a utility function that is given explicitly. When the network is uncertain, that means the endpoint takes whatever action will maximize the expected utility, taken over the distribution of possible configurations of the network. For example, the endpoint might try to achieve a fair allocation of throughput between it and cross traffic.

Although efforts have been made to reverse-engineer the utility function that TCP maximizes in steady state [20], these utility functions generally concern themselves only with throughput. In our system, the utility function is arbitrary and may also include a penalty for creating latency for other users of the network, meaning the sender will be discouraged from filling all queues between it and the destination, unlike TCP.

By separating the network model and utility function and making them first-class objects supplied to an endpoint mulling when it should send, our system's end goal can be clearly stated: we take the best possible action to maximize the supplied notion of utility, subject to the uncertainty we have about the network.

We note that the idea of writing out an explicit function to optimize in a distributed (end-to-end) way is the starting point of distributed mechanism design (DAMD) [10]; our proposal may be viewed as sharing some of the DAMD ethos, though we have not focused on issues like strategy-proof mechanisms that are central to prior DAMD work [8, 9]

We have implemented a preliminary inference engine to calculate and take this best possible action, and here present results from simulation arguing that this approach is tractable and holds promise.

## 2. Related Work

Network congestion control has been an active research area for over 25 years, with scores of papers on the topic and a variety of methods used in practice. However, most implemented schemes share the basic structure developed by Jacobson in the context of TCP congestion control (Tahoe and then Reno) [16]. In essence, all TCP variants model the entire network path using a single variable, cwnd, and use incoming ACKs to adjust this value and send out data. TCP also tracks the smoothed round-trip time (srtt) and linear deviation (rttvar) to set the retransmission timeout value, resetting cwnd to 1 segment when that happens. Much work has been done on different increase/decrease rules for cwnd within this architectural framework (e.g., NewReno [14], equation-based congestion control [12], binomial control [2], CUBIC [13], etc.). Other prior work has proposed using srtt increases to incorporate delay into window increase decisions (e.g., Vegas [6], FastTCP [26], etc.). Some other work has explicitly estimated bandwidth to optimize TCP performance over networks with wired and wireless links (e.g., Westwood [23]).

A different line of work jointly designs end-to-end control with explicit router participation, as in single-bit ECN [11], multi-bit extensions like VCP [27], and more expressive schemes like XCP [19]. Information set by routers help the end system determine whether to adjust cwnd or an explicit rate, but the approach still involves the sender modeling the entire network using a single cwnd or rate variable.

TCP congestion control was not designed with an explicit optimization goal in mind, but instead allows overall network behavior to emerge from its rules.

TCP congestion control assumes that all losses are caused by congestion, and much work has been done in hiding non-congestive stochastic losses from the sender using link-layer or subnetwork-layer methods [1, 15, 22]. At an extreme, some existing systems (some cellular wireless data networks) seem to over-buffer and over-zealously retransmit packets at the lower layers, leading to delays as high as multiple seconds. Such networks badly degrade the performance of a mix of short and long flows, and exhibit high latencies. Some research has looked at end-to-end techniques that attempt to distinguish congestion from non-congestive losses [5, 25, 3, 4, 7] using loss labeling, but thus far it has been unclear whether that is possible to do reliably.

Our approach models the network path as a combination of multiple non-deterministic idealized elements that explicitly capture things like the link speeds, the stochastic loss rate, buffer sizes, cross traffic, and other parameters. This model is known as a partially observable Markov decision processes (POMDP), for which there is a wealth of literature in the operations research and artificial intelligence communities that we benefit from [17, 24]. To our knowledge, applying this approach to network resource management is novel.

Our proposal is inspired by recent work on developing FII, a framework for evolvable network architectures [21]. FII incorporates an end-system API that decouples applications from the current socket layer. In a similar vein, we argue that end-to-end transmission control should be decoupled from rigid assumptions on the behavior of the underlying subnetworks.

## 3. MODEL-BASED TRANSMISSION CONTROL

On today's Internet, TCP's estimated quantities often do not correspond to actual parameters of the network. For example, the congestion window reflects both the available gateway buffer size in the network before buffer overflow will cause segment loss, but also segment loss that occurs for any other reason. On a network that includes a conventional FIFO queue, the round-trip time is not intrinsic to the network; it also depends on how fast the endpoints are sending and whether they have chosen to fill up the queue.

The RTT variation depends on TCP's own behavior and that of cross traffic, and is often not lightly-tailed as TCP's retransmission rules implicitly assume. TCP maintains only average estimates of these parameters and does not attempt to quantify their uncertainty. Additionally, TCP's model doesn't capture many behaviors of wireless networks, such as stochastic loss, intermittent connectivity, and changes in link speed.

By contrast, in our formulation the endpoint models the network explicitly as a nondeterministic automaton, as a combination of idealized elements. Because real networks do consist of human-engineered machinery and modelable natural phenomena (such as wireless fading), we believe such an explicit model of the network may be realistic.

The sender maintains a probability distribution of the possible states that the network could be in. At every step, the sender makes a decision about whether to send a packet immediately, or to schedule a transmission for some specific time in the future. It finds the decision that will maximize the expected value of an instantaneous utility function, by simulating the consequences of each possible timeout on the distribution of possible network states.

Separating the model from the transmission controller and making it an adjustable parameter will improve the evolvability of end-to-end transmission control, because new subnetwork technologies would no longer need to contort themselves to meet TCP's assumptions. If the model is rich enough, a new subnetwork might find itself as simply a different configuration of an already-modeled distribution of possible networks. A model that captures most of the behavior of a real network may be sufficient, without requiring each element of a real network to be represented in the model. If a new subnetwork exhibits truly novel behavior, that behavior could be modeled as a new network element and the endpoints' model extended.

Our approach consists of four parts: the model of the network itself, a sender that simulates possible network states to decide when best to transmit, an instantaneous utility function that the sender is trying to optimize, and a receiver.

### 3.1 Modeling the Network

The model is built as a language of network elements, corresponding to idealized versions of data structures and phenomena that occur in real networks:[2]

BUFFER: A tail-drop queue, whose unknown parameters are the size of the queue and its current fullness.

THROUGHPUT: A throughput-limited link, operating at a particular speed in bits per second.

DELAY: An unknown delay.

LOSS: Stochastic loss, independently distributed for each packet at a particular rate.

JITTER: A delay of a certain amount, introduced to randomly-selected packets with a particular probability.

PINGER: An isochronous sender of cross traffic at a particular rate.

INTERMITTENT: Connects input and output only intermittently, and switches from connected to disconnected according to a memoryless process with particular interarrival time (mean-time-to-switch).

SQUAREWAVE: Regularly alternates between connected and disconnected with a certain period.

ISENDER: A sender that follows our approach by maintaining a model of the network and scheduling transmissions to maximize the expected utility.

RECEIVER: The ISENDER's destination, which conveys the time of each packet received back to the ISENDER.

Some of these elements are non-deterministic. For example, when a packet is sent to LOSS, the element both loses the packet with probability $p$, and sends it on with probability $1-p$.

The network elements can be combined in various ways:

SERIES: Connects two network elements and sends the output of one to the input of the other.

DIVERTER: Routes packets from one source (such as the cross traffic) to one network element, and all other traffic to a different element.

EITHER: Sends traffic either to one element or another, switching with a specified mean-time-to-switch.

By combining these elements arbitrarily, it is possible to model more complicated networks. For example, a PINGER followed by one or more JITTERs can model a non-isochronous source of cross traffic. Multiple queues may be chained. Some cross traffic may be present only on some links but not others, and then only intermittently. The entire connection may die occasionally. Buffer sizes and throughputs can vary over time.

### 3.2 Sender's transmission behavior

The ISENDER is the object that implements our approach. It has two jobs: (1) maintain a model of the network configuration with specified uncertainty and (2) send packets according to a schedule that maximizes the expected utility.

[2]Figure 2 illustrates how these components can fit together in a real network.

The first task is accomplished using standard probabilistic techniques. Initially, the ISENDER has substantial uncertainty about the configuration of the network. There may be a THROUGHPUT whose link speed was drawn uniformly from a wide distribution. Each BUFFER may have a total size drawn from some distribution, with some unknown initial fullness. The rate of cross traffic and amount of jitter are also assumed to have been drawn from some probability distribution. We refer to this initial probability distribution as the "prior."

The ISENDER currently uses simple rejection sampling to maintain this probability distribution on the possible network configuration. At any point in time, it maintains a list of all possible configurations of the network and their corresponding probability.

Every time it receives an ACK from its RECEIVER or its timer (set below) expires, the ISENDER receives an event and wakes up. It simulates each of the possible network states since the last wakeup to see what results they would have produced at their simulated RECEIVER.

Any state that produces results inconsistent from what actually happened is removed from the list, and the probabilities of all remaining configurations are increased so that they still sum to unity. This process can be viewed as the sequential application of Bayes' theorem.

A nondeterministic element may "fork" the model into two possibilities; e.g., when LOSS receives a packet, it forks the model into a case where the packet is lost and one where it is sent. Both configurations are then simulated further.

Such forks do not generally lead to an unbounded explosion, because their effects do not linger forever. A packet that flowed through LOSS will eventually arrive at its destination and leave the network. Eventually, the two possible states of the network may become identical and can be compacted back into one state.

When a possibly-lost packet has gone into a buffer and has affected the timing of future packets, this "compaction" may not occur for some time. By contrast, if stochastic loss is assumed to occur only at the "last mile" of the link — after any buffers or throughput-limited links — then the consequences of stochastic loss do not linger.

Because network configurations are continually forking and because there may always be some ambiguity about the true state of the network, the number of network configurations is generally not pared down to one. However, we have found that the ISENDER can usually quickly pare down the prior to a smaller list of possibilities as it homes in on a good estimate of the network parameters.

This rejection-sampling approach is limited computationally; we have found that maintaining more than a few million possible discrete channel configurations is impractical. A more sophisticated scheme and scalable would use the approximate techniques of Bayesian inference that have been developed in the literature of POMDPs, such as Markov-chain Monte Carlo and belief compression, to represent continuous priors.

The ISENDER's second job is to, at every step, take the action that maximizes the expected value of the utility. This "action" may be one of two options: (a) "send now" or (b) "sleep until time $t$." (We assume the sender will always send packets of uniform length.)

Conceptually, then, the approach is simple: when the ISENDER wakes up, it makes a list of strategies including sending immediately and at every delay up to the slowest rate the ISENDER could optimally send. We evaluate the consequences of each strategy on each possible network configuration, and choose the strategy that maximizes the expected value of the utility.

If the RECEIVER notifies the ISENDER before $x$ seconds have passed that it has received some data, the sender will be woken up early and will reevaluate the best decision. Thus, the sender is really deciding at each step either to send, or to establish a timeout of a duration that maximizes utility.

## 3.3 Utility

This approach leads to some subtleties about how utility is defined. The simplest approach would be to credit the sender with some amount of utility for every byte that is received in the future. However, we wish to consider only the consequences of one sending decision at a time, looking forward only a finite number of seconds, and calculate the *instantaneous* utility of each packet.

This is defined as the packet size in bits, divided by $e^{\tau}$, where $\tau$ is the number of milliseconds in the future when the packet will be received. This has the effect of nearly linearly rewarding throughput — the accumulated instantaneous utility of a stream of packets will correspond almost linearly to the actual throughput for any realistic bitrate, since $\sum_{t=0}^{\infty} e^{-t/(1000r)} \approx 1000r + 0.5$ for $r > \frac{1}{100}$ packets per second.

The sender considers the utility to itself — which it can observe, since it receives its own packets — and also to any cross traffic, which it only observes in simulation. The utility function can optionally penalize latency experienced by the cross traffic, e.g. if the cross traffic is a video conference or other delay-sensitive traffic, which will discourage the sender from filling up a queue. The utility function may include a parameter varying the relative value of cross traffic compared with our own.

The sender does not need accumulate the total instantaneous utility under each possible delay forever — only until the consequences of each hypothetically sent packet have ceased to linger and every possible set of network configurations has become identical again, irrespective of delay decision. It chooses the delay that produces the highest accumulated utility over that interval and sleeps until that point.

We stress that the sender's algorithm need not be executed in real time. For a particular model and distribution of possible states, there will be a policy that can be computed in advance that prescribes the utility-maximizing behavior.

## 3.4 Receiver behavior

The RECEIVER accumulates packets and wakes up the SENDER for each one, notifying it of the received time and sequence number of the packet. In our preliminary experiments, we consider the sender and receiver clocks to be synchronized and the return path to be lossless. In the future, clock skew may need to be incorporated into the model as a parameter to be estimated.

### 3.5 Limitations

One important limitation at this stage is that we have not yet experimented with any networks that contain more than one ISENDER, or any network elements performing TCP. The question of an ideal sender's interaction with TCP, or with itself — whether starting with the same or different assumptions, and whether optimizing the same or a different utility function — will be of great importance to the practical realization of our approach.

Our current network elements are sparse — we will need to add some allowance for non-FIFO scheduling, active queue management, multipath intra-flow routing, and other real-life phenomena.

Finally, in initial experiments the network model represents the entire round-trip path between source and destination and back to source. We have modeled the return path as lossless and instant. In general, both paths will need to be modeled.

## 4. RESULTS

We have implemented the above design in C++ and embedded the ISENDER in an event-driven network simulation and tested it in a variety of configurations.

The sender reaches a predictable, ideal result in simple configurations, such as a single ISENDER connected to a queue, drained by a throughput-limited link. It begins tentatively if it is not sure of the link speed and initial buffer occupancy. Once it has inferred those parameters, it simply sends at the link speed from there on out.

If cross traffic is present and the utility function penalizes induced latency to other traffic, then the ISENDER drains the buffer before sending at the link speed.

We present here the results of a more complicated model, shown in Figure 2. We simulate isochronous cross traffic of an unknown speed, fading in and out intermittently. Both the sender and cross traffic feed into the same tail-drop buffer of unknown size, which is drained by a link of unknown speed. The packets then undergo stochastic loss with an unknown rate before arriving at their corresponding receivers. The sender only hears acknowledgments from its own receiver.

The ISENDER is initialized with a prior that includes, as one possibility, the true value of most of the parameters. The prior represents a discretized uniform distribution over the following ranges:

**Figure 2: Network model used in experiment**

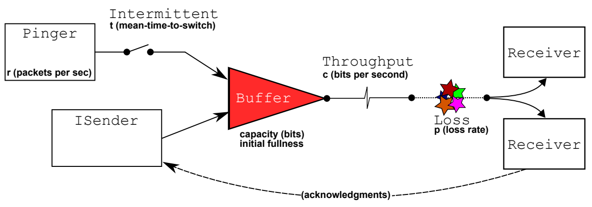
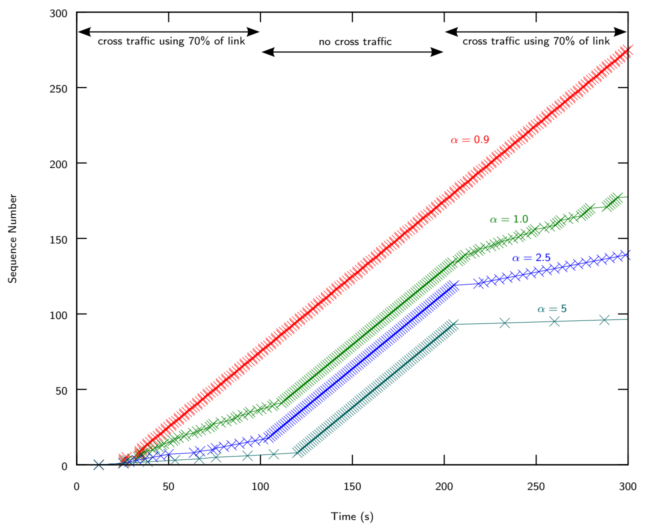


**Figure 3: Results of varying priority to cross traffic**



| Parameter | Prior belief | Actual |
|---|---|---|
| $c$ (link speed) | $10,000 \leq c \leq 16,000$ | 12,000 |
| $r$ (packets per second) | $0.4c \leq r \leq 0.7c$ | $0.7c$ |
| $t$ (mean time to switch) | 100 s | *n/a* |
| $p$ (loss rate) | $0 \leq p \leq 0.2$ | 0.2 |
| buffer capacity (bits) | $72,000 \leq x \leq 108,000$ | 96,000 |
| initial fullness | $0 \leq x \leq$ buffer capacity | 0 |

The true network can carry one 1,500-byte packet per second and stochastically loses 20% of all packets. Although the ISENDER assumes the PINGER switches on and off intermittently according to a memoryless process, in reality we switch deterministically every 100 seconds and observe the results. The ISENDER does not see the PINGER's traffic directly or receive acknowledgments; it has to infer probabilistically whether the PINGER is sending.

Our utility function is our own instantaneous throughput, times some multiple $\alpha$ of the throughput achieved by the cross traffic. By increasing this $\alpha$, we should see the ISENDER become more deferential to cross traffic.

The results are shown in Figure 3. The ISENDER behaves as desired. Irrespective of $\alpha$, the sender starts out slowly when it is uncertain of the channel parameters. As it narrows down the possible channel configurations, it begins to send at the optimal rate for its notion of utility.

During the period that the cross traffic is not sending, the ISender always sends at the exact link speed. (By the time the cross traffic shuts off 100 seconds in, the sender has figured out all the parameters of the channel, including the fact that the cross traffic occupies 70% of the link and the 20% packet loss.) The effect of $\alpha$ is to change the timidity with which the ISender reaches the conclusion that the cross traffic has switched off, and to adjust the sending rate when competing with the cross traffic.

When $\alpha < 1$, the sender has no reason to defer to the cross traffic and sends at the (discovered) link speed. When $\alpha = 1$, it senses the presence of the cross traffic and the fact that it occupies 70% of the link, and fills in the rest of the link. These calculations are not perfect because the ISender itself is operating under 20% packet loss. When $\alpha > 1$, the sender becomes more and more deferential to the cross traffic.

Except for the case when $\alpha < 1$, the ISender never causes a buffer overflow, since this would hurt overall utility by dropping one of the other sender's packets. For the sender who prioritizes his own packets above the cross traffic, the best decision is to flood out all of the other sender's packets.

## 5. Conclusion

This paper makes two contributions. The first is an argument that the requirement that subnetworks be capable of running TCP well hampers the effective integration of new network technologies into the Internet. The second is a proposal for end-to-end transmission control that departs from TCP to overcome this problem. The proposed architecture models the various features of a network path explicitly and associates probabilities with the different states the path can be in. At each moment, the endpoint acts to maximize the expected value of a utility function that is given explicitly. It has the ability to adapt to new technologies (which can be introduced without making any assumptions about the ends) by being able to discover their operational properties dynamically.

Our simulation results, though on a simple network configuration, still demonstrate that TCP-like behavior can be derived from first principles in a network against unknown, intermittent cross traffic, even in the presence of 20% packet loss. Because the utility function and model are arbitrary and the sender simply does the best thing possible given the information it has, this approach shows promise for being able to flexibly evolve endpoint behavior in response to changing subnetwork technologies and user preferences.

Much work remains to be done to demonstrate the utility of these ideas. More complex network paths have to be handled, as mentioned in §3.5. The rejection-sampling approach is not as scalable as other approaches, as mentioned earlier. Many more extensive simulations and experiments need to be done to convince the community of these ideas. Our hope is that this paper has intrigued and convinced the reader that this line of work is a promising architectural and technical direction worth discussing and refining.

## 6. Acknowledgment

This work was funded by NSF award CNS-1040072.

## References

[1] H. Balakrishnan, S. Seshan, and R. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), Dec. 1995.

[2] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *INFOCOM*, 2001.

[3] D. Barman and I. Matta. Effectiveness of loss labeling in improving TCP performance in wired/wireless networks. In *ICNP*, 2002.

[4] D. Barman and I. Matta. A Bayesian approach for TCP to distinguish congestion from wireless losses. In *WiOpt*, 2004.

[5] S. Biaz and N. Vaidya. Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. In *ASSET*, 1999.

[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*, 1994.

[7] S. Cen, P. Cosman, and G. Voelker. End-to-end differentiation of congestion and wireless losses. *EEE/ACM Trans. on Networking*, 11(5):703–717, 2003.

[8] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.

[9] J. Feigenbaum, R. Sami, and S. Shenker. Mechanism design for policy routing. In *PODC*, 2004.

[10] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *DIAL-M*, 2002.

[11] S. Floyd. TCP and Explicit Congestion Notification. *CCR*, 24(5), Oct. 1994.

[12] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM*, 2000.

[13] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.

[14] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *SIGCOMM*, 1996.

[15] IEEE P802.1-93/20b0. *Draft Standard IEEE 802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*.

[16] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM*, 1988.

[17] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[18] P. Karn, C. Bormann, G. Fairhurst, D. Grossman, R. Ludwig, J. Mahdavi, G. Montenegro, J. Touch, and L. Wood. Advice for Internet Subnetwork Designers, 2004. RFC 3819, IETF.

[19] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *SIGCOMM*, 2002.

[20] F. P. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[21] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, et al. Architecting for innovation. *CCR*, 2011.

[22] A. Larmo, M. Lindstrom, M. Meyer, G. Pelletier, J. Torsner, and H. Wiemann. The LTE link-layer design. *IEEE Comm. Mag.*, 47(4):52–59, 2009.

[23] S. Mascolo, C. Casetti, M. Gerla, M. Sanadidi, and R. Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *MobiCom*, 2001.

[24] N. Meuleau, K.-E. Kim, L. P. Kaelbling, and A. R. Cassandra. Solving POMDPs by Searching the Space of Finite Policies. In *Intl. Conf. on Uncertainty in Artificial Intelligence*, 1999.

[25] N. Samaraweera. Non-congestion packet loss detection for TCP error recovery using wireless links. *IEEE Comm.*, 146(4):222–230, 1999.

[26] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. on Networking*, 14(6):1246–1259, 2006.

[27] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. *IEEE/ACM Trans. on Networking*, 16(6):1281–1294, 2008.