International Journal of Future Generation Communication and Networking
Vol. 8, No. 2 (2015), pp. 137-186
http://dx.doi.org/10.14257/ijfgcn.2015.8.2.12

Evolution of TCP in High Speed Networks

Ivan Petrov and Toni Janevski

Makedonski Telekom, Kej 13 ti Noemvri, No.6, 1000 Skopje, Macedonia,
Email: ivan.petrov@telekom.mk
Ss. Cyril and Methodius University in Skopje, Faculty of Electrical Engineering and Information Technologies, Karpos 2 bb, 1000 Skopje, Macedonia,
Email: tonij@feit.ukim.edu.mk

Abstract

TCP congestion control protocols have low performances in high speed wide area networks mainly because their slow response with large congestion windows. This TCP behavior has initiated new design phase of alternative protocols that provide improved traffic utilization in high bandwidth delay product networks. The paper presents survey of various high speed sender side congestion control proposals that preserve the fundamental host to host principle. Solutions focus on variety of problems occurring in high speed environment with intention to eliminate congestion collapses and to ensure effective resource utilization. Internet data transfer does not depend only on that how TCP will utilize the network capacity, we have to stress that TCP must cooperate with existing transmitting data protocols through the same network in order to assure fair resource sharing. Part of the paper scope are state of the art high speed TCP proposals, we explore their congestion control techniques, strengths, weaknesses and we try to detect future TCP development possibilities.

Keywords: *TCP, congestion control, high speed TCP*

1. Introduction

TCP key feature is to provide reliable bi directional virtual channel communication between any two Internet hosts. It works over IP meaning that it has best effort packet delivering nature. We are not going to describe basic TCP functional design because it is well known instead we study the TCP behavior in high speed long delay networks. We should recall that TCP uses sliding window mechanism; it should be able to recover form packet losses in timely meaner, meaning that the shorter the interval between the packet transmission and loss detection, the faster TCP will recover. It is important to note that this interval should not be too short because the sender may detect permanent loss and will unnecessary retransmit the corresponding packet causing protocol overreaction and waste of network resources that may induce high network congestion. High speed TCP proposals use direct and indirect approaches of more aggressive network probing. Indirect approach use various losses and delay based models to create congestion control mechanism that will be aggressive enough in case of underutilized network and will remain gentle in case of utilized resources.

Several issues were identified as reason for poor performances of standard TCP protocols in high speed networks. Linear increase by one packet per round trip time in congestion avoidance phase is too slow and multiplicative decrease in case of loss is too drastic. Maintaining large average congestion windows at flow level require extremely small equilibrium loss probability; dynamics is unstable which leads to severe oscillations that can be reduced by accurate packet loss probability estimation and a stable flow dynamic design. At packet level, oscillation is unavoidable because TCP uses a binary congestion signal (packet loss).

ISSN: 2233-7857 IJFGCN

Proposed solutions can be arranged in three major groups: loss based, delay based and loss based with bandwidth estimation. Hybrid solutions use both, loss and delay as congestion measure. In Section 2 we observe STCP, HSTCP, BIC, H-TCP, CUBIC, Libra and Hybla as loss based high speed protocol representatives (reactive protocols); they all use packet loss as congestion measure that triggers the protocol phases. In Section 3 we analyze YeAH, Africa, Compound and Illinois as part of delay based solutions; they use queuing delay or round trip time variation to switch protocol phases, most of them use Reno when operating in slow mode combined with one of the loss based protocols during in fast mode of operation. Section 4 discuss FAST, New Vegas, TCPW-A, Log West wood+, ARENO and Fusion, proposals that use combination of Vegas, Reno and Westwood mechanisms. Section 5 concludes the paper.

In this survey we present the differences, some basic mathematical models and design parameters of mentioned protocols in order to review them with aim to detect further improvement possibilities. The paper scope presents only sender side modification mechanisms. Protocols requesting router or advanced layer modifications (Data link, Network, *etc*) are excluded from the analysis because of their complex implementation characteristics.

2. Loss Base High Speed TCP Protocols

2.1. HSTCP

HSTCP is modified protocol that tries to improve TCP behavior in high speed networks. It makes TCP additive increase and multiplicative decrease parameters to be a function of current congestion window when high congestion window size is achieved. This allows TCP to use congestion window sizes of tens of thousands of packets with realistic packet drop rates and minimal danger to the rest of the network. Protocol modification does not address quickly reaching of steady state speed, it utilize the congestion avoidance phase by modifying α and β parameters. Standard TCP protocol is defined with the following equation:

$$w \leftarrow w + \frac{\alpha}{w} \quad CA\ \ phase\ \ ,\ when\ \ ACK\ \ received$$

$$w \leftarrow w - \beta w \quad in\ \ case\ \ of\ \ drop$$

$$w \leftarrow w + \gamma \quad Slow\ \ Start$$

w , α and γ are defined in units of Maximum Segment Size (MSS). Default used values of increase, decrease and slow start parameters are $\alpha = 1$; $\beta = 0.5$ and $\gamma = 1$ respectively. Steady state TCP sending rate is given with:

$$x = \frac{\sqrt{\frac{\alpha(2-\beta)}{2\beta}}}{\sqrt{p}}$$

It is measured in packets per round trip time (ppt). When the default parameter values are used it obtains the following form:

$$x = \frac{\sqrt{1.5}}{\sqrt{p}}$$

p is the packet drop rate, it represents fraction of dropped or corrupted packets. *T* denotes the round trip time. When the protocol is in Slow Start phase instantaneous throughput *x(t)* in ppr is given with *x(t)=2t/T* , the time needed to reach speed *x* in ppr is *t=Tlog2(x)* seconds. When time out occur, the congestion window is reduced to $w = 1$ MSS and if the TCP sender is forced linearly to increase the congestion window it will need $w - 1$ round trip times to achieve the previous value of w packets. In case of *Gbit* connections the time needed to achieve the previous speed will be measured in hours. When we have single timeout the sender slow starts to back up a window of $w / 2$ and will recover in a reasonable time.

Development of high speed networks has opened new TCP chapter which improves protocol performances, it modifies the congestion avoidance phase identified as main reason of poor TCP performance in high speed networks.

New protocol design process requires number of constraints to be satisfied. The protocol should be TCP friendly; when competing with standard TCP flows all should have fair share of the bandwidth. The protocol should be incrementally deployable; it should reach high speeds reasonably quickly when in slow start; it should recover from multiple time-outs and should perform well in case of congestion, this means that congestion avoidance properties should be improved or at least the standard one should be preserved.

$$x = \frac{\sqrt{1.5}}{\sqrt{p}}$$

Basic idea of High Speed TCP is to modify the TCP response function at very low packet drop rates in order to achieve speed of 10Gbps. This modification should solve the issue of translating the response function to modified window increase and decrease parameters α and β . HSTCP does not change the Slow Start parameter γ. It sets its value at one packet meaning that *cwnd* will be doubled in slow start phase for each *RTT*.

HSTCP is compatible with existing TCP's, it uses standard increase and decrease parameters when current packet drop rate *p* is greater than some value *P* or when the current congestion window is at most W packets. It uses following switching point values *W*=31 and *P*=0.0015 given in [1-3]. Target drop rate should be chosen for a congestion window *W1*. The target *W1* in this case is chosen to be 83333 packets which corresponds of 10 *Gbps* sending rate when 1500 byte packets and a round trip time *RTT*=100 ms are assumed. Packet drop rate is calculated (*P*=10-k) for a congestion window of *W1* packets. Default target point values given in [1-3] are *W1*=83000 and *P1*=10-7. Now we have two given points that can be presented on log log scale. We can write the equation of line defined by two points. Standard TCP response function is defined and can be calculated as $w = \frac{1.22}{p^{0.5}}$, with help of predefined point values it can be drown on log log scale. The two points (*P*,*W*) and (*P1*,*W1*) give the response function of high speed TCP, when *p*<*P*. It is chosen the new response function to be linear on a log-log scale, meaning that the new response function is defined by equation of line defined with two points:

$$\log w = \frac{\log W1 - \log W}{\log P_1 - \log P}(\log p - \log P) + \log W \equiv y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

Having in mind that $S = \frac{\log W1 - \log W}{\log P_1 - \log P} \equiv \frac{y_2 - y_1}{x_2 - x_1}$ represents the slope of the line, the upper equation can be written as $\log w = \log(p^S \left(\frac{1}{P} \right)^S W)$ from where $w = p^S \left(\frac{1}{P} \right)^S W$ in case when *(P,W)* are set at (0.0015,31) and *(P1,W1)* are set at (10-7,83000) we receive the following *S* value *S*=-0.82, if we sustain the values in the last equation we will obtain the new response function $w = \frac{0.15}{p^{0.82}}$. Graphical presentation of *wHSTCP* and *wAIMD_TCP* when *p* is in range of [10-7, 0.0015] is given at Figure 1.

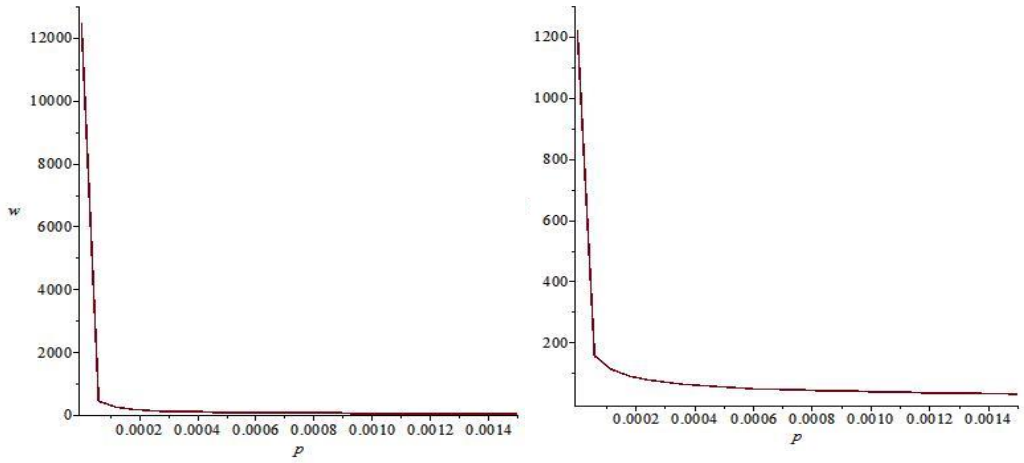


Figure 1. Graphical Presentation of *wHSTCP* (left) and *wAIMD_TCP* (right) when *p* is in Range [10-7, 0.0015]

Increase and decrease parameters are set to be functions of current window size *w* when the congestion window receives values larger than the predefined threshold of *W* packets. This means that for congestion window of *W1* packets increase and decrease parameters are chosen to satisfy the following equation:

$W_1 = \frac{\sqrt{\frac{\alpha(w)(2 - \beta(w))}{2\beta(w)}}}{\sqrt{P_1}}$ and $\alpha(w)$ will satisfy the following $\alpha(w) = W_1^2 P_1 \frac{2\beta(w)}{(2 - \beta(w))}$ this is the case when $\alpha(w) = 72$ and $\beta(w) = 0.1$ and *(P1,W1)* are set at (10-7,83000). This means that the decrease after drop is 10% and the increase is just under 0.1% per round trip time in absence of drop.

When $w \le W$ and $p \ge P$, HSTCP is defined with:

$$\alpha(w) = 1 \qquad when \quad w \le W$$

$$\beta(w) = 0.5 \qquad for \quad w \le W$$

The response function is $w = \frac{\sqrt{\frac{\alpha(w)(2-\beta(w))}{2\beta(w)}}}{\sqrt{p}}$ ppr.

HSTCP goal is to choose $\alpha(w)$ and $\beta(w)$ parameters so that the response function will give *W* for a packet drop rate *p(W)=P* and *W1* for a packet drop rate *p(W1)=P1*, when $w>W$, we can express *p* to be a function of *w*, from the upper equation we obtain:

$$\log(p(w)) = (\log(P_1) - \log(P))\frac{\log(w) - \log(W)}{\log(W_1) - \log(W)} + \log(P)$$

$$p(w) = w^m \left(\frac{1}{W}\right)^m P; \quad m = \frac{\log(P_1) - \log(P)}{\log(W_1) - \log(W)}; \; p(w) = \frac{0.099}{w^{1.22}}$$

Decrease parameter $\beta(w)$ is defined with $\beta(W) = 0.5$ and $\beta(W_1) = B$, when $w \geq W$ $\beta(w)$ is let linearly to vary as log of *w* according the following equation (equation of line on log log scale, *β,w*):

$$\beta(w) = (B - 0.5)\frac{\log(w) - \log(W)}{\log(W_1) - \log(W)} + 0.5$$

Increase parameter $\alpha(w)$ can be computed using standard TCP equation $\alpha(w) = \frac{w^2 2\beta(w)p(w)}{2-\beta(w)}$.

Relative fairness between HSTCP and regular TCP given with the ratio $F = \frac{w_{HSTCP}}{w_{TCP}} = \frac{0.12}{p^{0.32}}$ when *p* is in range of [10-7, 1] is presented in Figure 2.

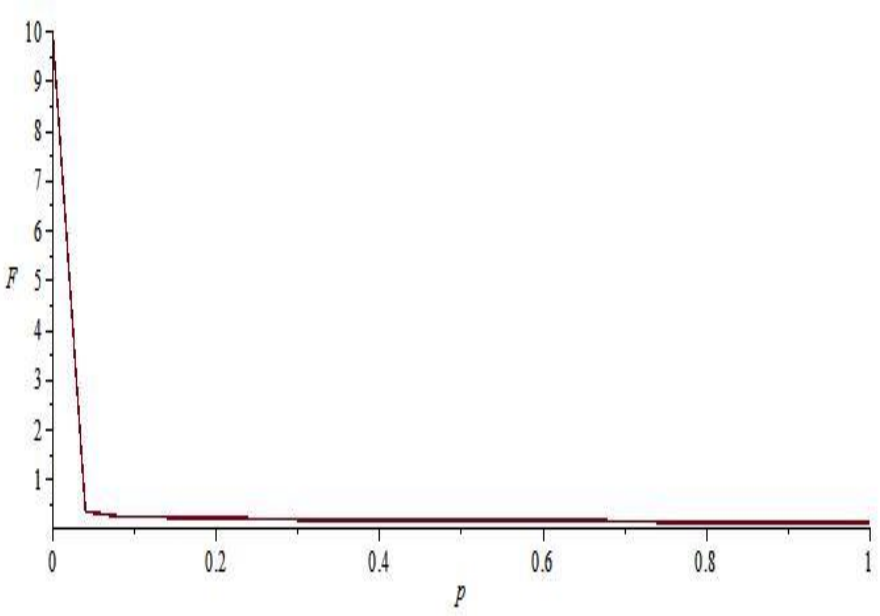


Figure 2. Presentation of Relative Fairness F Defined as wHSTCP /wAIMD_TCP when p Receives Values in Range of [10-7, 1]

2.2. Scalable TCP

Scalable TCP is designed to be incrementally deployable and behaves identically as traditional TCP stacks when small windows are sufficient. It is a simple sender side modification of TCP congestion window update algorithm. Altering of congestion window adjustment algorithm can help improving agility with large windows. Scalable algorithm is defined as follows:

$$w \leftarrow w + 0.01 \qquad CA\ phase\ ,\ for\ each\ \ ACK$$

$$w \leftarrow w - 0.125\ w \qquad drop$$

Scalable window update gives the algorithm name. In case of 1 *Gbps* connection, round trip time of 200 ms and packet size of 1500 bytes the algorithm provides original rate recovering after transient in less than 3 seconds as explained in [4] meaning that it provides better high speed network utilization (in networks with speeds greater than 100 *Mbps* and round trip times values above 50 ms).

Each source and destination network pair is identified with route *r*. End to end route dropping probability in time *t* is denoted with *p(t)*. *w* and *T* denote the congestion window and round trip time of connection route *r*. Having this in mind the generalized form of Scalable TCP can be written as:

$$w \leftarrow w + \alpha \qquad CA\ phase\ ,\ for\ each\ \ ACK$$

$$w \leftarrow w - \beta w \qquad drop$$

α and β are constants in range of $[0,1]$. Usually standard TCP packet loss recovery time is proportional with connection window size and the round trip time while Scalable TCP has packet loss recovery time proportional only with the connection round trip time (MIMD basic equations); this is identified as a main reason why Scalable outperforms traditional TCP protocols. α and β values are constrained by implementation and deployment conditions. It is clamed that values of 0.01 and 0.125 are chosen in order to be achieved best bandwidth allocation properties like flow rate variance, convergence properties and stability. Response function of generalized Scalable TCP for small end to end drop rates is given with:

$$w \approx \frac{\alpha}{\beta}\frac{1}{p}$$

Graphical presentation of *wstcp* when *p* is in range of [10-7, 0.0015] is given in Figure 3.

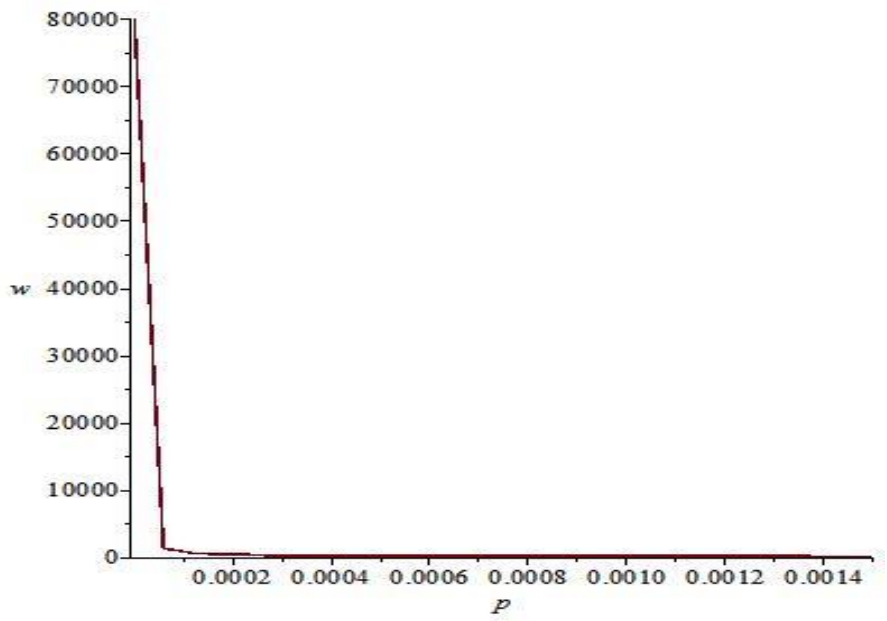


Figure 3. Graphical Presentation of wstcp

If we compare it with traditional response TCP function for small end to end drop rates given as:

$$w \approx \frac{\sqrt{1.5}}{\sqrt{p}}$$

we can notice that multiplicative factors *p* are different.

Legacy window size (*lwnd*) and legacy loss rate (*pl*) defined as maximum packet loss rate needed to support windows larger than *lwnd* are defined. Standard TCP connections can not use effectively large windows, larger than certain size (*lwnd*) because of limited send buffer memory and the limited amount of socket receive. Scalable TCP uses traditional TCP algorithm if $cwnd \leq lwnd$ and the scalable congestion window algorithm when $cwnd > lwnd$. It is important to notice that drop rate constraints are similar, in case of congestion when the drop rate is higher than *pl*, Scalable TCP connections use the traditional TCP algorithm and for drop rates lower than *pl*, legacy connections will have a window of at least *lwnd* packets, this enables Scalable TCP connections to receive larger windows than legacy connections but legacy connections on the other side will never starve for bandwidth.

lwnd is a policy decision value. Continuous and decreasing response curve must pass through the point (*pl*,*lwnd*) which gives the following constraint

$$lwnd = \frac{\alpha}{\beta}\frac{1}{p_l} = \frac{\sqrt{1.5}}{\sqrt{p_l}}$$

$$\frac{\alpha}{\beta} = p_l lwnd \approx \sqrt{1.5\, p_l}$$

The coefficient of variance for the instantaneous sending rate is given with

$$CoV\ (x_r) = CoV\ \left(\frac{w}{T}\right) \approx \sqrt{\frac{\beta}{2}}$$

When $P_r \to 0$, β should be chosen as small as possible in order to reduce the instant rate variation. It is sensible the algorithm to have smaller rate variation than traditional TCP meaning that $\beta \le \frac{1}{2}$. STCP responds to congestion events at most once per round trip time. It is necessary that the window expansion and contraction cycle lasts longer than the round trip time. Using packet loss recovery time of Scalable TCP and noticing that β is the only free variable we obtain the following constraint:

$$\beta > 1 - \sqrt{\frac{1}{1.5\,p_l}}$$

If we have in mind the general form of MIMD algorithm, *w* changes and in order to provide faster convergence the source should reduce it's sending rate after receiving feedback by ½ in less than $\frac{\log\left(\frac{1}{2}\right)RTT}{\log\left(1-\beta\right)}$ seconds. As a response of sudden increase of available route capacity when $P_r \to 0$, the time taken by the source to increase its sending rate by factor of 2 is $\frac{\log\left(2\right)RTT}{\log\left(1+\alpha\right)}$ seconds.

Convergence properties suggest β and α to be chosen as large as possible in order to provide fast convergence. This conflicts the desire to keep the instantaneous rate variation small which requires β to be small. For heterogeneous round trip times, the generalized STCP algorithm is locally stable around its equilibrium point

$$\alpha < \frac{p_i(\hat{x}_i)}{\hat{y}_i\, p_i'(\hat{x}_i)} \quad \forall\, i \in I$$

$\hat{x}_i$ represents equilibrium rate of each link, $p_i(x)$ is loss probability of link *i* for arrival rate x and I represents set of all links. The fairness of STCP with AIMD TCP can be expressed as $F = \frac{w_{STCP}}{w_{TCP}} = \frac{0.066}{p^{0.5}}$ or graphically presented at Figure 4 in case when *p* receives values in range of [1e-10, 1]

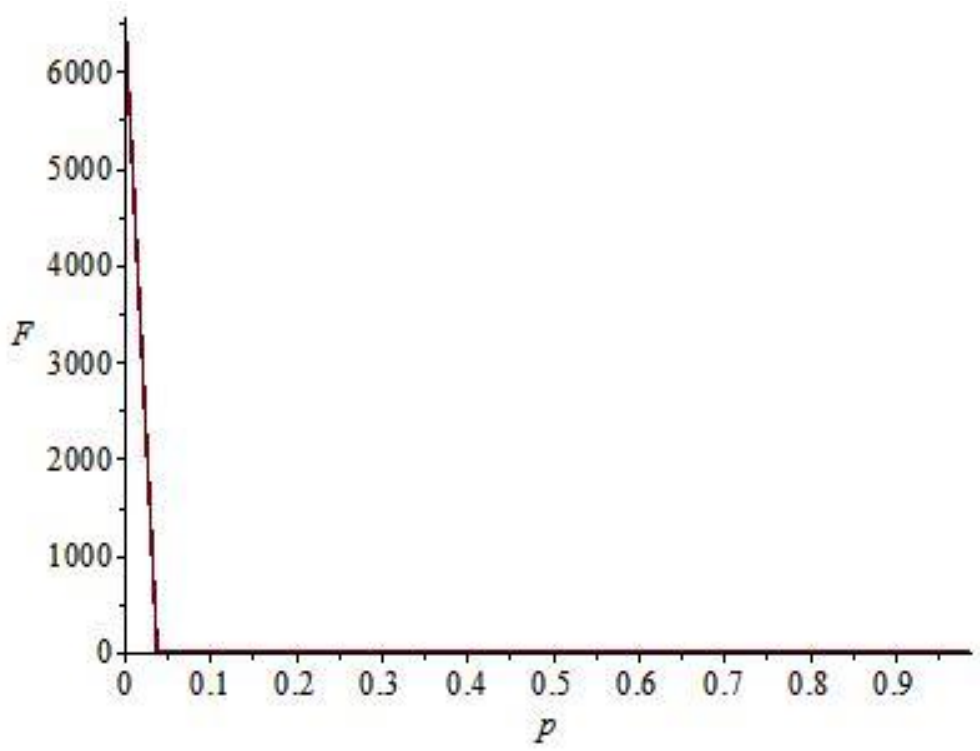


Figure 4. STCP Fairness when p Receives Values in Range [1e-10, 1]

2.3. BIC

TCP friendliness and bandwidth scalability where subject of observation of initially developed high speed protocols. *RTT* unfairness is important parameter that should be addressed during protocol design phase. Interesting problem occur when flows with different *RTT's* are competing; result is unfair bandwidth flow share. This issue is common for drop tail routers in high speed networks where packet loss can be highly synchronized. Fine tuning of key parameters, for example congestion windows and network interface buffers can solve this issue. Jumbo packet options and multiple TCP connections enhance the bandwidth utilization. Guaranteeing TCP friendliness and bandwidth scalability with one fixed window increase rate is challenging. Adaptive schemes of window varying growth rate in dependence of network conditions should be used. BIC protocol combines additive increase and binary search increase mechanisms. HSTCP and STCP have *RTT* unfairness issue when multiple flows with different *RTT's* are competing for the same bottleneck bandwidth. *RTT* unfairness is defined as flow windows ratio in terms of their *RTT* ratio. Usage of *RED* routers reduces *RTT* unfairness. *RTT* unfairness can be described with congestion window growth; we can note that larger window increases faster than smaller window. Synchronized loss is other issue that occurs simultaneously across multiple competing flows. Probability that a window losses at least one packet increases exponentially. Loss event probability in case of uniform distribution of random loss probability p, is given with $1 - (1 - p)^w$. Small router side loss can cause loss events across multiple large windows flows. Synchronized loss encourages *RTT* unfairness because loss events are more uniform across different window sizes if the size is larger than a certain limit, large windows with short *RTT* can always grow faster than small windows with long *RTT*. AIMD protocols scale the bandwidth share by increasing its additive increase factor and provide linear *RTT* fairness but are not TCP friendly. HSTCP and STCP are scalable under low loss rates and TCP friendly under high loss rates but are not *RTT* fair. BIC TCP is designed to be scalable, to provide *RTT* fairness,

TCP friendliness and convergence. Let w_i denote the window size just before a loss event and RTT_i denote RTT of the flow i. i will denote the interval between two consecutive loss events during steady state.

We can recall that two HSTCP flows with α and β increase, decrease parameters given as window size function can be presented as:

$$w_1 = w_1(1 - \beta(w_1)) + \alpha(w_1)\frac{t}{RTT_1}$$

$$w_2 = w_2(1 - \beta(w_2)) + \alpha(w_2)\frac{t}{RTT_2}$$

With substitution of $\alpha(w) = \frac{2w^2\beta(w)p(w)}{2 - \beta(w)}$ we obtain $w = \frac{0.15}{p(w)^{0.82}}$ and $\frac{w_1}{w_2} = \left(\frac{2 - \beta(w_2)}{2 - \beta(w_1)}\frac{RTT_2}{RTT_1}\right)^{4.56}$

SCTP flows can be expressed in the following form:

$$w_1 = w_1(1 - \beta)(1 + \alpha)^{\frac{t}{RTT_1}}$$

$$w_2 = w_2(1 - \beta)(1 + \alpha)^{\frac{t}{RTT_2}}$$

It is interesting to notice that these equations do not have solution meaning that SCTP does not have convergence point indicating that shorter RTT flow consumes all of the bandwidth while the longer RTT flow drops down to zero. Response function of congestion control protocol describes the sending rate expressed as a function of packet loss rate. It gives information about protocol scalability, convergence, RTT fairness and TCP friendliness. Protocol defined with response function $\frac{x}{p^y}$ where x and y are constants and p is a loss event rate has RTT unfairness proportional to $\left(\frac{RTT_2}{RTT_1}\right)^{\frac{d}{1-d}}$, d increases the response function slope and assures RTT unfairness increase. Response function slope presented in a log log scale determines RTT protocol unfairness. TCP friendliness can be defined with crossing point where protocol response function crosses the TCP response function. HSTCP and STCP behave like standard TCP below the cross point (point where their response functions cross the TCP one). Above the cross point, HSTCP and STCP run their own "scalable" protocols, ideally consuming unutilized bandwidth left by TCP flows. The protocol becomes TCP friendly if the response function crosses TCP function at as low as possible loss rate (to the left of the X axis) because below that point the protocol behaves as TCP. Moving the cross point to the left increases the response function slope, this is the case when scalability has to be maintained but at the same time it hurts RTT fairness. An ideal protocol would have response function that will cross the TCP response function at lowest rate as possible and under lower loss rates its slope would be close to the one of the AIMD protocol.

BIC TCP consists two logical/functional parts: binary search increase and additive increase.

2.3.1. Binary Search Increase

Two values are defined, $W_{\min}$ is estimated as a flow window size when packet loss is not detected, $W_{\max}$ receives predefined value. Binary search increase can be implemented between $W_{\max}$ and $W_{\min}$. Target window size represents the midpoint between these two values. If a packet loss occurs during target point window increment then the current window can be treated as a new maximum and the reduced window size after a packet loss can be the new minimum, midpoint between these new values becomes the new target. This is a reasonable approach since the network incurs loss around the new maximum and not around the minimum; it is obvious the new target point to be in the middle of the defined values. When the target window is reached and if packet loss is not detected during increment phase it becomes the new minimum and new target window is calculated. The process continues until the difference between the maximum and the minimum falls below the preset threshold, named as minimum increment $S_{\min}$. This technique is called binary search increase; it allows more aggressive bandwidth probing when the difference between the current window size and the target window is large; it assures less aggressive probing as the current window size gets closer to the target window size. The number of lost packets usually is proportional with the last increment value before loss happens; this indicates that binary search increase phase reduces the packet losses. Protocol increase function is logarithmic. Concave response function is the main benefit of this technique usage.

2.3.2. Additive Increase

Binary search increase is combined with additive increase strategy in order to ensure faster convergence and *RTT* fairness. When the distance between the midpoint and the current minimum is too large, increasing directly to the midpoint might add too much network stress. Maximum increment $S_{\max}$ is defined. When the distance between the midpoint and the current minimum is larger than $S_{\max}$ then in the next *RTT* the congestion window is increased by $S_{\max}$ until the distance becomes less than $S_{\max}$ when the window increases directly to the target point. After large window reduction this strategy linearly increases the congestion window value.

2.3.3. Slow Start

When the window grows past the current maximum, the maximum is unknown. Binary search increase sets its maximum to the default value; it is usually large constant while the current window size is set to be the minimum. In this case the target midpoint can be very far, if the target midpoint has very large value then the window is linearly increased by the maximum increment. Slow start is activated to probe the bandwidth up to *Smax* value. If *cwnd* is the current congestion window size than in each *RTT* round the increase steps are *cwnd+1, cwnd+2, cwnd+4, ..., cwnd+ Smax*. This technique is chosen because the window size is around its saturation point but the maximum is unknown. This phase safely probes the available bandwidth in the slow start until it becomes safe to increase the window by $S_{\max}$, we have to note that after exiting the slow start phase protocol switches to binary increase.

2.3.4. Fast Convergence

In binary search increase it takes $\log(d) - \log(S_{\max})$ *RTT* rounds to reach the maximum window value after window reduction. In case of two flows with different window sizes and same *RTT*, the larger window will be decreased more when protocol is in multiplicative decrease phase of operation. Window increases with log step causing the larger and smaller window to reach their maxima fast and almost at the same time. The smaller window takes small amount of the larger window before the next window reduction happens. In order to improve this behavior the binary increase phase is modified. After window reduction the new maximum and minimum are set, we can denote them as $\max\_win_i$ and $\min\_win_i$ for the flow *i*. If the new maximum is less than the previous one the window is in downward trend so the new maximum is readjusted to be the same as the new target window and the target is readjusted, after this operation normal binary increase is used. This mechanism is called fast convergence.

2.3.5. BIC Response Function

Loss event happens at every *1/p* packets, congestion epoch is defined to be a period between two consecutive loss events. $W_{\max}$ Denote the window size just before a loss event, after a loss event the window size decreases to $W_{\max}(1-\beta)$. It is defined that BIC switches additive increase to binary search increase when the distance from the current window size to the target window is less than $S_{\max}$. The target window is the midpoint between $W_{\max}$ and the current window, BIC switches between those two increases when the distance from the current window size to $W_{\max}$ is less than $2S_{\max}$. Let *N1* and *N2* be the numbers of *RTT* rounds of additive increase and binary search increase respectively.

$$N_1 = \max\left(\left|\frac{\lceil W_{\max}\beta \rceil}{S_{\max}}\right| - 2, 0\right)$$

The total amount of window increase during binary search increase can be expressed with $W_{\max}\beta - N_1 S_{\max}$. If we assume that this is divisible by $S_{\min}$ then N_2 can be obtained

$$N_2 = \log_2\left(\frac{W_{\max}\beta - N_1 S_{\max}}{S_{\min}}\right) + 2$$

In additive increase the window grows linearly with slope $1/S_{\max}$ and the total number of packets during additive increase can be obtained with

$$Y_1 = \frac{1}{2}\left(W_{\max}(1-\beta) + W_{\max}(1-\beta) + (N_1 - 1)S_{\max}\right)N_1$$

Binary search increase grows logarithmically

$$Y_2 = W_{\max}N_2 - 2\left(W_{\max}\beta - N_1 S_{\max}\right) + S_{\min}$$

Total *RTT* number in congestion epoch is *N=N1+N2* and the total number of packets in an epoch is *Y=Y1+Y2*. The loss event happens at every *1/p* packets so *Y* can be expressed

as *Y=1/p*. Above equations help $W_{\max}$ to be expressed as a function of *p*. Detail equation presentation is given in [5]. In this section will present only the main conclusions.

BIC Sending rate is proportional with $\frac{1}{p^{d}}$, when *1/2 < d <1*. As the window size increases, *d* decreases from *1* to *1/2*. For a fixed β, when the window size is small, the sending rate is a function of $S_{\min}$ and when the window size is large it is a function of $S_{\max}$. When the window is small, the protocol is TCP-friendly, when the window is large, it is more *RTT* fair and provides higher sending rate than TCP. This can be achieved by adjusting $S_{\min}$ and $S_{\max}$.

RTT fairness can be examined with the following when $W_{\max}\beta \le 2S_{\max}$ and *w1/w2* $\frac{w_1}{w_2} = e^{\left(\frac{1}{RTT_1}-\frac{1}{RTT_2}\right)t\ln(2)}$ can be expressed with , *t* increases with the total bandwidth. The peak is achieved when $W_{\max}\beta = 2S_{\max}$, in this case it becomes linearly proportional to RTT_2 / RTT_1. Exponential *RTT* fairness can be problematic in case of larger windows but under small windows synchronized loss is less frequent so it is believed that the unfairness can be managed to be low. The coefficient setting is given in [5].

2.4. HTCP

HTCP is another approach that tries to improve TCP behavior in high speed/long distance networks. It is already shown that flows may take unacceptably long time to recover their window size after congestion event occurs in high speed long distance networks. One approach is the current congestion window value to be used as *BDP* path indicator. AIMD parameter α can be increased as function of *cwnd* that assures additive increase algorithm directly to scales with the bandwidth delay product. This approach is similar with the one used by HSTCP and STCP proposals, they increase the multiplicative decrease factor β which highly influence and increase the flow aggressiveness. Adjusting AIMD parameters for a particular flow based on the *cwnd* flow state creates a fundamental network asymmetry that can lead to undesirable network behavior. TCP should address adequately several issues like friendliness with legacy TCP, efficiency, fairness and responsiveness.

2.4.1. Network Analysis

Let $w_i(k)$ denote the *cwnd* value of flow i immediately before backoff. k indexes the congestion events. Let $\alpha_i(k)$ and β be the AIMD increase decrease parameters of the i th flow, let T be the interval between congestion events k and $k+1$. $T(k)$ is measured in seconds. $\alpha_i(k)$ is defined as:

$$\alpha_i(k) = \frac{w_i(k+1) - \beta w_i(k)}{T(k)}$$

it represents effective increase rate in packet/s. $\alpha_i(k)$ is often approximated with *1/RTT* where RTT_i is the round trip time of flow i.

$$w_i(k+1) = \beta w_i(k) + \alpha_i(k) T(k)$$

in case of two flows i and j we have

$$w_i(k+1) - w_j(k+1) = \beta\left(w_i(k) - w_j(k)\right)$$

in case of *n* congestion events we can write the following equation

$$w_i(k+n) - w_j(k+n) = \beta^n\left(w_i(k) - w_j(k)\right)$$

Because $\beta < 1$ it indicates that w_i, w_j must eventually convergence to the same value. The number of flow congestion epochs needed to assure converge is determined by β. β determines the number of congestion epochs needed for convergence but the duration of the congestion epochs depends on both β and α. Increasing β and/or α reduces the congestion epoch duration. Adjusting β to reduce the congestion epoch duration, results with a corresponding (exponential) increase of the congestion epochs number before convergence. It is known that asymmetry is created by *RTT* unfairness which leads directly to increasing rate differences that provide additional unfairness.

If we define $z_i = \frac{w_i}{\alpha_i}$ as normalized congestion window (by the increase rate) and if we consider two existing flows in the network we can write the following:

$z_i(k+1) = \beta z_i(k) + T(k)$ and $z_j(k+1) = \beta z_j(k) + T(k)$ form where we can obtain $z_i(k+1) - z_j(k+1) = \beta\left(z_i(k) - z_j(k)\right)$ in equilibrium we have $z_i = z_j$ or presented in other form as $\frac{w_i}{w_j} = \frac{\alpha_i}{\alpha_j}$ we should recall that $\alpha_i \approx \frac{1}{RTT_i}$ if we substitute this value in the previous equation we get the following $\frac{w_i}{w_j} = \frac{RTT_j}{RTT_i}$. During network equilibrium we have $w_i = \frac{\alpha_i T(k)}{1-\beta_i}$ and for two flows *i* and *j* we can write the ratio $\frac{w_i}{w_j} = \frac{\alpha_i(1-\beta_j)}{\alpha_j(1-\beta_i)}$ which indicates that fairness is ensured when the ratio $\frac{\alpha_i}{1-\beta_i}$ is the same for all flows. Standard TCP fairness requirement when default parameter values are used is given with $\alpha_i = 2(1-\beta_i)$ (substitute values of 1 and 0.5 in the previous equation)

This analysis is valid for any network type where the flows update their congestion windows in accordance with $w_i(k+1) = \beta w_i(k) + k_i f(T(k))$ where *k* represents a constant and $f(T(k))$ is any function that remains the same for all flows (ensures symmetry between flows).

This analysis can be extended for unsynchronized flows.

We will generalize the AIMD algorithm in case when the increase parameter is elapsed time function since the last occurred congestion event. Δ will note the elapsed time in seconds since the last congestion event. The increase parameter is noted as $\overline{\alpha}(\Delta)$. Compatibility with the existing AIMD protocols can be obtained if the following increase parameter constraints are fulfilled

$$\overline{\alpha}(\Delta) = \begin{cases} 1 & \Delta \le \Delta^L \\ \overline{\alpha}^H(\Delta) & \Delta > \Delta^L \end{cases}$$

Δ^L is the switching threshold parameter that assures switching from standard to novel increase algorithm function $\overline{\alpha}^H(\Delta)$. Increase $\overline{\alpha}$ is measured in *packets/RTT*.

cwnd will be updated as follows

$$w \leftarrow w + \frac{\overline{\alpha}(\Delta)}{w} \quad for \ each \ ack$$

.

$\overline{\alpha}^H(\Delta)$ Defines the rate at which bandwidth is acquired; when it receives value of 1 the standard AIMD algorithm is recovered.

HTCP chooses increase function $\overline{\alpha}^H(\Delta)$ that will provide reasonably small congestion epoch duration as the path bandwidth delay product increases.

The congestion epoch duration is defined with solution of the following equation

$$\alpha(k)T(k) = w(k+1) - \beta w(k) \text{ where } \alpha(k) = \frac{\int_0^{T(k)} \overline{\alpha}(t)dt}{T(k)RTT}.$$

It is important to notice that α has straightforward impact on the network behavior, while backoff impact factor is more complicated. Backoff parameter choice has impact on congestion epoch duration and on the number of congestion epochs before convergence. The backoff factor impacts significantly the throughput efficiency and network queue requirements within the network.

In case of congestion, which means that the buffer queue is full and the network bottleneck is operating at link capacity, the bottleneck link throughput is given with

$$R(k)^- = \sum_i^n \frac{w_i(k)}{T_i + \frac{q_{\max}}{B}} = B$$

B is the link capacity and $q_{\max}$ is the bottleneck buffer size, T_i is the round trip time experienced by the $i-th$ source when the bottleneck queue is empty. $\beta_i(k) = \beta_i$ if the

flow i experiences loss at the $k-th$ congestion event otherwise $\beta_i(k)=1$. The data throughput is given with

$$R(k)^{+}=\sum_{i}^{n}\frac{\beta_i(k)w_i(k)}{T_i}$$

under assumption that the bottleneck buffer empties. Maximum throughput can be achieved by equating the rates $R(k)^{-}$ and $R(k)^{+}$ which results with the following constraint

$$\beta_i \geq \frac{T_i}{T_i+\frac{q_{\max}}{B}}=\frac{RTT_{\min .i}}{RTT_{\max,\,i}}$$

For given β_i the choose of $q_{\max}$ can be done such that $R(k)^{-}=R(k)^{+}$ for all k. For any given size of $q_{\max}\geq BT_i$ can be set

$$\beta_i=\frac{RTT_{\min,\,i}}{RTT_{\max,\,i}}$$

for all i, ensuring $R(k)^{-}=R(k)^{+}$ for all k.

Backoff factor (β) increasing in order to improve efficiency decreases the network responsiveness and link utilization. Decreasing of the backoff factor increases the responsiveness but decreases the network efficiency (especially when queues are small).

Flow backoff factors adjustment can lead to unfairness between competing flows. Fairness can be restored by adjusting α_i at $\alpha_i=2(1-\beta_i)$ which leads to $\beta_i=\frac{RTT_{\min,\,i}(k)}{RTT_{\max,\,i}(k)}$

In case of multiple bottlenecks the equations will receive the following form

$$R(k)^{-}=\sum_{i=1}^{n}\frac{w_i(k)}{\sum_{j=1}^{m}(T_i+\frac{q_j(k)}{B_j})}=B$$

m represents the number of links at which packets are queued. $q_j(k)$ is the j link queue occupancy and B_j represents the bandwidth of the link j. The backoff factor can be written in the following form

$$\beta_i(k) = \frac{T_i}{T_i + \sum_{j=1}^{m} \frac{q_j(k)}{B_j}} = \frac{RTT_{\min,\ i}}{RTT_{\max,\ i}}$$

After back off we have

$$R(k)^+ = \sum_{i}^{n} \frac{T_i}{T_i + \sum_{j=1}^{m} \frac{q_j(k)}{B_j}} \frac{w_i(k)}{T_i} = B$$

Taking in consideration previous analysis H-TCP algorithm can be defined as

$$w \leftarrow w + \frac{\overline{\alpha}}{w} \qquad on\ each\ \ ack \qquad \overline{\alpha} \leftarrow 2(1-\beta)\overline{\alpha}(\Delta)$$

$$w \leftarrow w - \beta w \qquad drop \qquad \beta \leftarrow \frac{RTT_{\min}}{RTT_{\max}} \quad \beta \in [0.5, 0.8]$$

where

$$\overline{\alpha} = \begin{cases} 1 & \Delta \le \Delta^L \\ 1 + 10\left(\Delta - \Delta^L\right) + \left(\frac{\Delta - \Delta^L}{2}\right)^2 & \Delta > \Delta^L \end{cases}$$

Δ represents the time in seconds since the last congestion event, $\Delta^L = 1s$ and $\frac{RTT_{\min,}}{RTT_{\max,}}$ is the ratio of minimum and maximum flow *RTT*. Increase rate can be effectively invariant with round trip time. *RTT* unfairness between competing flows is mitigated when *RTT* scaling is employed. $\overline{\alpha}$ is presented at Figure 5 as a function of time in seconds since the last congestion event when $\Delta^L = 1s$. (in our scenario it receives only positive values).

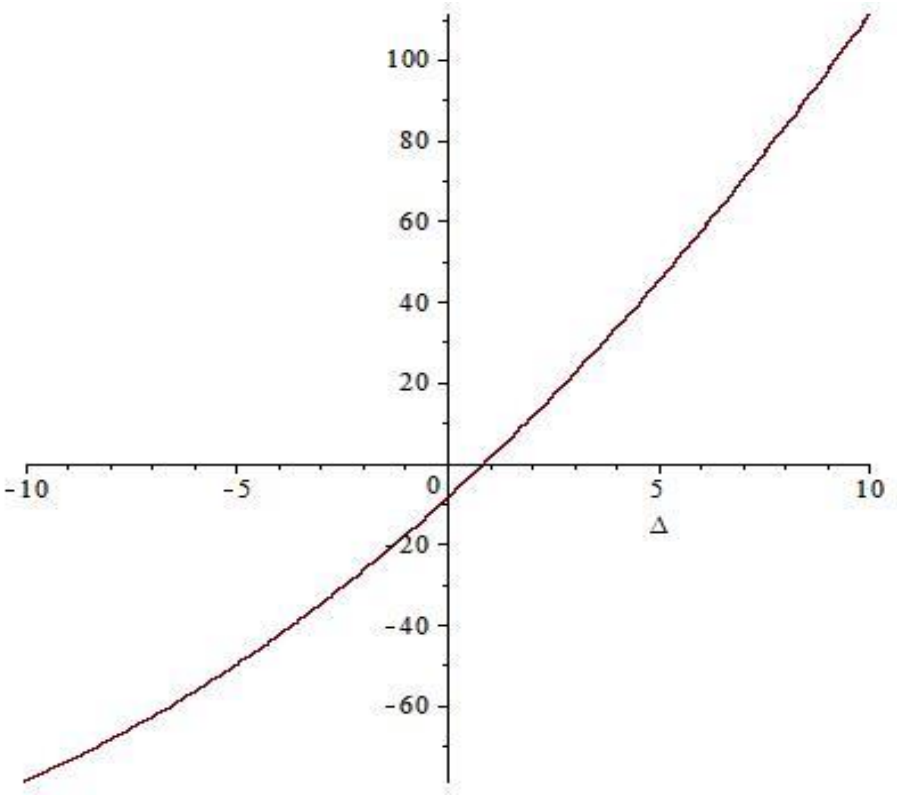


Figure 5. Graphical Presentation of $\overline{\alpha}$

2.5. TCP Cubic

It is claimed that Cubic presents enhanced BIC protocol version, which simplifies BIC window control; improves its TCP friendliness and *RTT* fairness. Cubic window growth is governed by cubic function in terms of elapsed time since the last occurred loss event. BIC growth function can be too aggressive under short *RTT* or in case of low speed networks. The protocol name originates from window growth function given with the following equation:

$$W_{cubic} = C\left(t - K\right)^3 + W_{\max}$$

C represents scaling factor, *t* represents elapsed time since last window reduction event, *Wmax* is the window size just before last window reduction and *K* receives the given form $K = \sqrt[3]{\frac{W_{\max}\ \beta}{C}}$ where β is multiplication decrease factor applied for window reduction at time of loss. *K* is obtained when *Wcubic* is compared with *Waimd* in time of congestion event when *t* receives null value.

$$w_{AIMD} = w_{\max}\left(1 - \beta\right) + \alpha\ \frac{t}{RTT}$$

Characteristic for Cubic algorithm is that the window grows very fast upon window reduction but when it gets closer to *Wmax* it slows down the growth. Around *Wmax* value window increment becomes zero, above that value the window starts to grow slowly, after that it starts probing for more and more bandwidth as it moves away from *Wmax*. This

approach enhances protocol stability and increases network utilization. Scalability is assured with fast window growth when away from *Wmax*. Cubic assures intra protocol fairness among competing flows. Good *RTT* fairness is assured because the window growth is dominated by *t*. Linear *RTT* fairness is provided because competing flows with different *RTT* have same *t* value after synchronized packet loss. Fairness and stability are enhanced by *Smax* parameter used for window increase when current window value is away from *Wmax*. This enables linear growth when the window size is away from *Wmax*. Linear window increase during Cubic operation is real time dependant under short *RTTs*. Linear *RTT* increment is smaller and stays constant in real time. It is important to notice that real-time window increase enhances protocol friendliness. Window growth function of *RTT* dependent protocols grows proportionally faster in shorter *RTT* networks where CUBIC will grow *RTT* independently. In short *RTT* networks, Cubic's window growth is slower than standard TCP. After loss event Cubic reduces its window by factor of β, the TCP-fair additive increment would be $3\frac{1-\beta}{1+\beta}$ per RTT. The average sending rate of an Additive Increase Multiplicative Decrease protocol (AIMD) is given with

$$\frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{1+\beta}{1-\beta}\frac{1}{p}}$$

where *p* represents the loss rate. For standard TCP values of $\alpha=1$, $\beta=1/2$ the average sending rate receives the following form

$$\frac{1}{RTT}\sqrt{\frac{3}{2}\frac{1}{p}}$$

Protocol achieves same average sending rate as TCP, if α equals $3\left(\frac{1-\beta}{1+\beta}\right)$. This growth rate per *RTT* assures emulated TCP window size at time *t* after last congestion epoch to be given with

$$W_{tcp} = W_{\max}\beta + 3\frac{1-\beta}{1+\beta}\frac{t}{RTT}$$

If *Wtcp* is larger then *Wcubic*, the window size is set to *Wtcp* otherwise *Wcubic* represents the current congestion window size. Detail parameter evaluation is presented in [7]. At Figure 6 is given graphical presentation of Cubic *cwnd* change as a function of *t*.

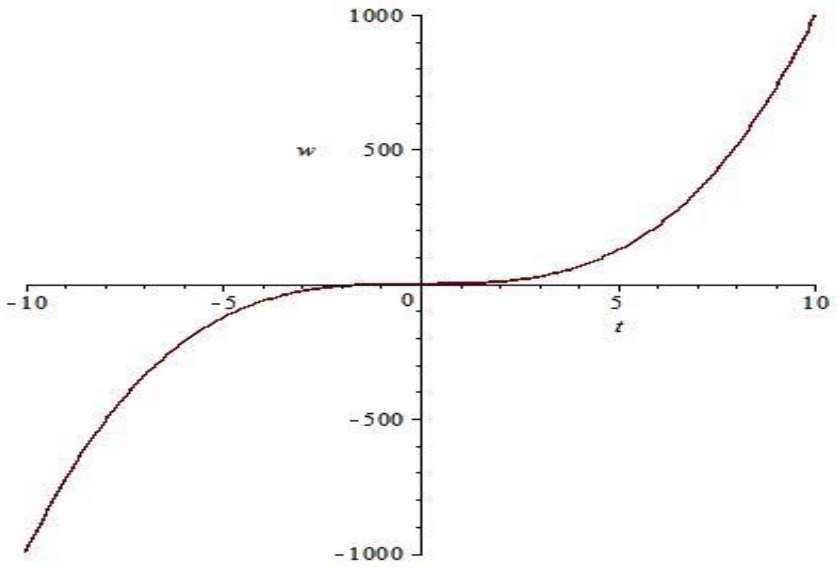


Figure 6. Graphical Presentation of wcubic

2.6. TCP Libra

Libra represents another high speed protocol version that uses following increase and decrease parameters.

$$\hat{a}_r \leftarrow \frac{\alpha_r \tilde{T}_r^2}{T_0 + \tilde{T}_r} a_r$$

$$\hat{b}_r \leftarrow \frac{T_1}{T_0 + \tilde{T}_r} b_r$$

Tr represents average round trip time, *ar* and *br* represent standard TCP increase and decrease parameters. Authors ensure that coefficients choice improves several protocol characteristics. Stabilized rate is achieved, round trip time independence is assured ($\hat{a}_r \infty \tilde{T}_r^2$); reduced round trip time sensitivity of the system matrix is obtained by having $T_0 >> \tilde{T}_r$. System convergence speed to stable point value is not penalized when coefficient is multiplied by α_r. Protocol scalability is achieved and throughput variance is reduced. Flow variance can be controlled by adjusting *T0* and *T1* parameters. By setting *T0≥1* and *T1≤1* the variance can be diminished by factor of $\frac{1}{\tilde{T}_r + 1}$, flow variance scales with $\tilde{x}_r^2$. New coefficient *αr* was introduced; it was designed to increase protocol convergence speed and to achieve algorithm scalability, main coefficient task is to keep algorithm behavior stable. This is the main reason why it was designed to be a product of scalability and penalty/dumping factor given as $\alpha = SP$

Scalability factor receives the following form *S=k1Cr*. Where *k1* is a constant and *Cr* is the bottleneck link capacity seen by the *r -th* source. Penalty factor is designed to make algorithm increase rate adaptive to the network congestion. Queuing time was adopted as a network congestion measure defined with $T_r(t) - T_r^{\min}$. Penalty factor is designed to penalize window increase when congestion occurs and to maintain window value close to

the maximum one as long as possible. *Tr(t)* represents instantaneous round trip time, *Tmaxr* represents the maximum round trip time and *Tminr* is minimum round trip time experienced by *r*-th connection. Several functions can be used to penalize window growth when instant round trip time gets close to maximum round trip time experienced during congestion.

Libra algorithm can be written in following form:

$$w \leftarrow w + \frac{1}{w} \frac{\alpha_n T_n^2}{T_n + T_0} \quad for \ received \quad ack$$

$$w \leftarrow w - \frac{T_1 w}{2\left(T_n + T_0\right)} \quad drop$$

Number of penalty functions can be used; several of them are given below:

$$k_2\left(\frac{T_r^{\max} - T_r(t)}{T_r^{\max} - T_r^{\min}}\right); \quad k_2\left(\left(\frac{T_r^{\max} - T_r(t)}{T_r^{\max} - T_r^{\min}}\right)\left(\frac{2T_r^{\max} - T_r^{\min} - T_r(t)}{T_r^{\max} - T_r^{\min}}\right)\right); \quad e^{-k_2 \frac{T_r(t) - T_r^{\min}}{T_r^{\max} - T_r^{\min}}}$$

Numbers of conditions are originating from protocol stability analysis; following expression is one of them:

$$k_2 > -\frac{T_{\max} - T_{\min}}{\tilde{T} - T_{\min}} \log\left(\frac{\tilde{T} + T_0}{k_1 C \tilde{T}^2}\right)$$

Detailed protocol analysis can be found in [8]

2.7. Hybla

Hybla tries to solve *RTT* disparity issue by modifying standard rule of congestion window increase.

$$w_{i+1} \leftarrow w_i + 1, \quad SS$$

$$w_{i+1} \leftarrow w_i + \frac{1}{w} \quad CA$$

TCP segment transmission is bounded with min of (*cwnd*, *awnd*) where advertised window corresponds to receiver's buffer size. In order to investigate *RTT* impact on congestion control, ideal channel environment is considered and continuous model should be used to describe the window evolution. *w(t)* denote *cwnd* segment value. t_γ denotes the time when *γ, ssthresh* value is reached. *w(t)* can be expressed as:

$$w(t) = \begin{cases} 2^{\frac{t}{RTT}} & for \ \ 0 \le t < t_\gamma, \ SS \\ \frac{t - t_\gamma}{RTT} + \gamma & for \ \ t \ge t_\gamma, \ CA \end{cases}$$

where $t_\gamma = RTT \log_2 \gamma$. It is obvious that for lower *RTT* values window increase will be higher. Segment transmission rate is defined with $B(t) = \frac{w(t)}{RTT}$. $T_d(t)$ Denotes transmitted data amount by standard TCP source from transmission beginning

$$T_d(t) = \int_0^t B(\tau) d\tau = \begin{cases} \frac{2^{\frac{t}{RTT}} - 1}{\ln 2} & for \quad 0 \le t < t_\gamma, \ SS \\ \frac{\gamma - 1}{\ln 2} + \frac{(t - t_\gamma)^2}{2 RTT^2} + \gamma \frac{(t - t_\gamma)}{RTT} & for \quad t \ge t_\gamma, \ CA \end{cases}$$

Hybla tries to make *B(t)* to be *RTT* invariant. It makes *W(t)* to be *RTT* independent by scale modification, compensating the effect of *RTT* division. It introduces normalized round trip time ρ.

$\rho = \frac{RTT}{RTT_0}$ where *RTT0* represents reference connection round trip time that should be optimally utilized. *W(t)* is multiplied with ρ making it *RTT* independent and receives the following form:

$$w^H(t) = \begin{cases} \rho 2^{\frac{\rho t}{RTT}} & for \quad 0 \le t < t_{\gamma,0}, \ SS \\ \rho \left\lceil \rho \frac{t - t_{\gamma,0}}{RTT} + \gamma \right\rceil & for \quad t \ge t_{\gamma,0}, \ CA \end{cases}$$

$t_{\gamma,0}$ Define the time when the congestion window reaches $\rho\gamma$ value which remains same for every *RTT*, $t_{\gamma,0} = RTT_0 \log_2 \gamma$.

Segment transmission rate is defined with $B^H(t) = \frac{w^H(t)}{RTT}$ or with

$$B^H(t) = \begin{cases} \frac{2^{\frac{t}{RTT_0}}}{RTT_0} & for \quad 0 \le t < t_{\gamma,0}, \ SS \\ \frac{1}{RTT_0} \left\lceil \frac{t - t_{\gamma,0}}{RTT_0} + \gamma \right\rceil & for \quad t \ge t_{\gamma,0}, \ CA \end{cases}$$

from where we can notice that it is *RTT* independent. *TdH* receives the following form

$$T_d^H(t) = \int_0^t B^H(\tau) d\tau = \begin{cases} \frac{2^{\frac{t}{RTT_0}} - 1}{\ln 2} & for \quad 0 \le t < t_{\gamma,0}, \ SS \\ \frac{\gamma - 1}{\ln 2} + \frac{(t - t_{\gamma,0})^2}{2 RTT_0^2} + \gamma \frac{(t - t_{\gamma,0})}{RTT_0} & for \quad t \ge t_{\gamma,0}, \ CA \end{cases}$$

It is obvious that the window change does not depend anymore on the actual *RTT* value. Dynamic window can be defined as:

$$w_{i+1}^{H} = \begin{cases} w_i^{H} + 2^{\rho} - 1 & SS\ \ phase \\ w_i^{H} + \dfrac{\rho^2}{w_i^{H}} & CA\ \ phase \end{cases}$$

|We can notice that CA update rule is similar with the constant rate algorithm. Minimum ρ value is 1, in this case Hybla behaves as standard TCP for fast connections ($RTT \le RTT_0$). The initial $cwnd$ and the original $ssthresh$ values must be multiplied with ρ as well as the transmission buffer size. w_{i+1}^{H} and B^{H} equations hold under assumption that the sender side is limited by congestion window and not by the advertised window size. If this is not fulfilled, receiver imposed limitation can be removed by increasing the advertised window by ρ, this ensures same transmission rate ceiling for all connections. Receiver side modification is not mandatory. SACK option is included and recommended to be used in TCP Hybla proposal [9]. *ssthresh* receives standard value.

3. Dealy Based High Speed TCP Proposals

3.1. YeAH-TCP: Yet another High Speed TCP

YeAH TCP developers considered that newly designed high speed TCP proposals are not correctly evaluated. It is considered that the main protocol target is not achieved. Primary algorithm goal should be capacity to avoid network congestion instead capability to achieve full link utilization. YeAH TCP is claimed to be another congestion control protocol able to fully exploit high BDP links without loosing its congestion control capabilities. YeAH tries to achieve following performances: it tries efficiently to exploit network capacity by congestion window update rule modification; induced network stress should be less or equal to the one induced by TCP Reno algorithm. The protocol should achieve TCP friendliness with Reno traffic; Algorithm should be internally and *RTT* fair. Performances should not be substantially impaired by non congestion related packet loss events; Small link buffers should not prevent higher performance. In order to address all mentioned issues YeAH TCP is created to operate in two different modes, fast and slow. During fast mode operation YeAH increments the congestion window in accordance with aggressive STCP rules while in slow mode it operates as Reno. Protocol state operation depends on the estimated number of packets in the bottleneck queue. RTT_{base} is the minimum RTT measured by the sender and $RTT_{\min}$ is the minimum RTT estimated in the current data window (estimated packet *RTT* in the current $cwnd$). Estimated queuing delay is given with $RTT_{queue} = RTT_{\min} - RTT_{base}$. From RTT_{queue} we can infer the number of queued packets.

$$Q = RTT_{queue}\ G = RTT_{queue}\ \frac{w}{RTT_{\min}}$$

G represents the goodput. The ratio between the queuing RTT and the propagation delay can be evaluated, this parameter is noted as L. $L = \dfrac{RTT_{queue}}{RTT_{base}}$ is parameter that indicates network congestion level. $RTT_{\min}$ is updated once per data window. If

$Q < Q_{\max}$ and $L < \frac{1}{\varphi}$ the algorithm is in fast mode operation otherwise it is in slow mode. $Q_{\max}$ and φ are tunable parameters. The first one represents the maximum number of packets in single flow allowed to be kept in the buffers; $\frac{1}{\varphi}$ parameter represents maximum level of buffer congestion with respect of BDP.

Precautionary decongestion algorithm is implemented during slow mode operation. Whenever $Q > Q_{\max}$, congestion window is diminished by Q and *ssthresh* set to *cwnd/2*. $RTT_{\min}$ is computed once per RTT, the decongestion granularity is one RTT. Q represents the excess amount of packets estimate with respect to the minimum required *cwnd* used to exploit available bandwidth. Amount of packets can be removed from actual congestion window without goodput degradation to be caused. When the number of competing flows increases, every flow attempts to fill the buffer with the same packet quantity independently of the perceived RTT, this behavior allows internal RTT fairness to be achieved. Decongestion is optimal when flows that use it do not compete with greedy one. YeAH-TCP implements mechanism to detect when the flow is competing with "greedy" sources.

In case of competition with Reno flows, the protocol does not implement queue decongestion; when $Q > Q_{\max}$ YeAH-TCP attempts to remove packets from the queue, queuing delay increases because Reno flows are filling up the buffer. In this case, YeAH-TCP will hardly ever stay in fast mode operation and frequently will be in slow mode. On the contrary, when competing with non greedy flows, YeAH algorithm will trigger state change from fast to slow whenever buffer content builds up above $Q_{\max}$ and back as soon as the precautionary decongestion becomes effective. This protocol behavior makes it possible to distinguish between two different competition circumstances with help of *RTT* number counting when the algorithm is in the defined states. Two counting variables are defined: $count_{reno}$ and $count_{fast}$. $count_{fast}$ represent the number of $RTT's$ during fast mode protocol operation. $count_{reno}$ is estimate of congestion window value of competing Reno flows. Decongestion takes place only during slow mode operation and if $cwnd > count_{reno}$ in order to avoid congestion window decrease below the estimated *cwnd* Reno flow values. At start-up $count_{reno}$ is initialized at *cwnd/2*, this value is incremented by one during every slow mode RTT, when a packet loss is detected, $count_{reno}$ is halved. The variable is reset to current *cwnd/2* whenever $count_{fast}$ is greater than a threshold, indicating that the current flow is competing with other non-greedy flows and at the same time $count_{fast}$ is reset at 0.

When loss is detected by three duplicate ACK, current estimate of the bottleneck queue Q, can be exploited to find the number of packets that should be removed from the congestion window in order to empty the bottleneck buffer and yet to leave the pipe full. This rule is similar with TCP Westwood principle. After loss the rule permits full link utilization for every bottleneck buffer size when the loss is not caused by network congestion. In case of three duplicate ACK's, when YeAH TCP does not compete with Reno flows, *cwnd* is decreased for $\min\{\max\{cwnd/8, Q\}, cwnd/2\}$ segments. If YeAH-TCP competes with Reno flows, congestion window is halved.

3.2. TCP Africa

TCP Africa is new delay sensitive two mode congestion avoidance protocol. We know that most straightforward approach is to modify TCP increase and decrease parameters and to adjust the response function in order to provide more desirable protocol performances. Africa poses similar priorities like previously described high speed protocols. It tries to improve network utilization capabilities, it should provide inter and intra fairness and should not introduce additional network stress. High speed protocols nature is to be more aggressive than the standard protocols. Analysis of the congestion event frequency is conducted, how often congestion event occur when high speed protocols are used. $w_{\max}$ denote maximum flow congestion window per loss free path, it can be approximated as

$$w_{\max} = C \frac{RTT}{L} + B$$

C represents link capacity in bits per seconds, *RTT* is flow round trip time, *L* is packet size given in bits and *B* is maximum queue size length of the bottleneck link. Buffers usually are chosen to be proportional with link delay bandwidth product; $w_{\max}$ is expected to scale linearly with *RTT* and capacity. It is necessary to consider that after reaching some $w_{\max}$ self induced packet loss will occur. Induced congestion event period for a general AIMD protocol is defined with

$$T = \frac{(1-\beta)w_{\max}}{\alpha}$$

α and β of TCP Reno receive common values of 1 and 0.5 respectively so the congestion event period obtains the following form

$$T = \frac{w_{\max}}{2}$$

Multiplicative Increase Multiplicative Decrease protocols that use multiplicative increase parameter α and multiplicative decrease parameter β have a constant steady state induced congestion period that satisfies the given equation:

$$(\beta w_{\max})(1+\alpha)^{T} = W_{\max}$$

T can be expressed as

$$T = \frac{-\log(\beta)}{\log(1+\alpha)}.$$

AIMD equation of a given congestion event frequency allows max window to be expressed. TCP Reno max window is written below

$$w_{\max} = \frac{T\alpha}{1-\beta}$$

Design phase observation of the existing protocols is completed. HSTCP and STCP increase the rate as the window grows. These protocols are the most aggressive as they

reach the maximum capacity, we have to notice that it is reasonably the protocol to be least aggressive around the maximum capacity value. Delay based protocols are least aggressive when sending around link capacity. It is shown that worst performances of the delay based protocols are provided when competing with loss based algorithms. (ex. Reno).

Africa is a hybrid protocol that uses delay metric to determine whether the bottleneck link is congested or not. In case of uncontested link the protocol operates in fast mode and it use aggressive scalable congestion rule, otherwise it is in slow mode of operation. We can conclude that in case of congested network the protocol uses standard Reno algorithm. Africa creator has decided to use HSTCP algorithm in fast mode mainly because it wanted to achieve compatibility with Reno at low congestion window size. When packet delay starts to increase the protocol senses that the amount of available bandwidth is becoming smaller and it switches to linear increase mode which is quite conservative for every additional received packet per round trip time. If we have this in consideration it can be concluded that TCP Africa is quick to utilize available bandwidth but slow when it comes to inducing next congestion event.

aRTT is exponential smoothed high accuracy round trip time estimate that is kept by the flow. Minimum path delay, denoted as $\min\ RTT$ is used to estimate link queuing delay. Delay metric is used to switch protocol phases. Africa uses TCP Vegas delay metric. Congestion is detected with help of the following equation:

$$\frac{w\left(aRTT\ -\min\ RTT\right)}{aRTT}\underset{<}{\overset{\geq}{}}\alpha$$

Quantity $aRTT\ -\min\ RTT$ gives network queuing delay estimate. Overall round trip time is $\min\ RTT+\left(aRTT\ -\min\ RTT\right)$. Quantity $\frac{aRTT\ -\min\ RTT}{aRTT}$ represents proportion of round trip time due to queuing delay rather then propagation delay. TCP maintains average sending rate of $\frac{w}{aRTT}$ packets per seconds so $\frac{w\left(aRTT\ -\min\ RTT\right)}{aRTT}$ represents queue packet number estimate. α usually is a real number constant set at value grater than one. α determines protocol delay sensitivity. The congestion avoidance phase is entered if the following is fulfilled:

$$\begin{aligned}
&if\ \left(w\left(aRTT\ -\min\ RTT\right)<\alpha\ *\ aRTT\right)\{\\
&\quad w=w+\frac{fast_increase\ \left(w\right)}{w}\\
&\}\ else\ \{\\
&\quad w=w+\frac{1}{w}\\
&\}
\end{aligned}$$

f*ast increase(W)* function is specified by set of custom designed TCP rules. Africa uses HSTCP increase rule. Another constraint can be used as phase switching mechanism, it receives the following form

$$\left(aRTT\ -\min\ RTT\right)>or\ <\alpha\ \frac{aRTT}{w}$$

This metric represents total queue packet number estimate. It suggests that other congestion metrics might use other techniques such as packet pair probing to estimate maximum link capacity and then to estimate the total number of packets in the queue. Bandwidth estimation techniques could be used in parallel with TCP data flow in order to detect when is most appropriate aggressively to increase the congestion window size. In this case the delay metric is chosen because it provides switch over point always at the same rate for a given amount of queuing delay independently of the round trip flow time. Most queuing delay sensitive flows are those that have smallest $\frac{aRTT}{w}$ value in the given condition $(aRTT - \min RTT) > or < \alpha \frac{aRTT}{w}$, from where becomes clear that first they switch to slow mode of operation. α parameter impacts the protocol performances in a positive way. Advanced auto tuning α technique should be assured in future.

3.3. Compound TCP

Compound TCP uses both delay and loss based approaches to control the sending rate. The main idea of CTCP is to add scalable delay based component into standard TCP. Compound TCP is designed to be TCP friendly and efficient at the same time. Compound behaves similar like the previous proposals, if it senses underutilized link it tries quickly to increase the sending rate. It uses combination of loss and delay based approach of high speed congestion control. Like we stated before this protocol is designed to provide efficient use of network resources and to achieve high link utilization, it achieves good TCP fairness with help of the delay based component. It has improved *RTT* fairness than the common TCP. New state variable is introduced, delayed window (*dwnd*) which controls the CTCP delay based component. Congestion window *cwnd* controls the loss based algorithm component. CTCP sending window is controlled by both *cwnd* and *dwnd*. TCP sending window is calculated as follows:

$$win = \min(cwnd + dwnd, awnd)$$

where *awnd* is the advertised window of the receiver. *cwnd* update is done like in TCP congestion avoidance phase, it is increased by one *MSS* every *RTT* and halved upon a packet loss event.

$w \leftarrow w + \frac{1}{win}$ where $w \equiv cwnd$ and *win* is defined with previous equation. When the connection is initialized, standard slow start mechanism is used because exponential window increase provides quick bandwidth utilization in high speed networks. *dwnd* receives null value when connection is in slow start, this component is effective only when the algorithm is in congestion avoidance phase.

CTCP should be aggressive at the beginning when the network is underutilized and it should reduce its sending rate accordingly when fully utilized network is sensed. It should react appropriately in case of packet loss because it could indicate congestion. Algorithm delay based component is derived from TCP Vegas. $baseRTT$ is new state variable maintained as packet transmission delay estimation over the network path. At connection start $baseRTT$ is updated by the minimal *RTT* that has been observed. $sRTT$ is exponentially smoothed value of the current *RTT*. The number of connection backlogged packets is estimated as:

$$Expected = \frac{win}{baseRTT}$$

$$Axtual = \frac{win}{RTT}$$

$$Diff = (Expected - Actual)baseRTT$$

Expected gives throughput information obtained when the network path is not overrun. *Actual* provides information for really obtained throughput. *Diff* gives information of the difference between expected and actual throughput. Multiplying with $baseRTT$ is done because of the injected but not delivered network data into the last round, this corresponds with the backlogged bottleneck queue data. Threshold parameter γ is used to detect congestion. If Diff<γ the network path is underutilized otherwise it can denote congestion and delay based algorithm component should reduce the window size.

CTCP uses following algorithm

$$win \leftarrow win + \alpha win^{k} \quad for\ each\ ack$$

$$win \leftarrow win(1 - \beta) \quad in\ case\ of\ loss$$

α, β and k are tunable parameters that give the desirable scalability, smoothness and responsiveness. Delay loss component is designed as follows:

$$dwnd = \begin{cases} dwnd + (\alpha win^{k} - 1)^{+}, & if\ diff < \gamma \\ (dwnd - \xi diff)^{+}, & if\ diff \geq \gamma \\ \left(win(1 - \beta) - \frac{cwnd}{2}\right)^{+}, & if\ loss\ is\ \text{det}\ ected \end{cases}$$

where $(\ )^{+}$ is defined as $\max(\ ,0)$

$dwnt$ needs to increase for $(\alpha win(t)^{k} - 1)^{+}$ packets mainly because of the loss based component and *cwnd* will be increased by one packet. In case of loss, *dwnd* is set to the difference between the desired reduced window size and *cwnd*. Second rule line is important it shows that *dwnd* does not decrease when the queue is built. ζ defines how rapidly the delay based component should reduce this window when early congestion is detected. *dwnd* receives only positive values. CTCP window is low bounded by its loss based component. During this elaboration it is considered that loss is detected by three duplicate ACKs. In case of retransmission *dwnd* is reset to zero and the delay based component is disabled. After time out the sender is in slow start mode, when it exits this mode the delay based component may be enabled. When the window is small CTCP behaves as standard TCP. Delay based component is used only when *win* is larger than *wlow*. Detail parameter setting is given in [12-13].

3.4. TCP Illinois

Illinois use packet loss information as a decision measure when to increase or decrease the congestion window size. AIMD based algorithms increase the window size by $\frac{\alpha}{w}$ packets for each ACK and thus the window is increased by constant $\frac{\alpha}{b}$ per round trip time (*RTT*) if all packets are acknowledged within an *RTT*. *b* represents the number of acknowledged packets by each ACK (it has value of 1 for original TCP and 2 for delayed ACK). Congestion window is decreased by fixed proportion βw (β has standard value of ½) when packet loss is detected. Conservative nature of AIMD protocol is the main reason for poor protocol performances in high bandwidth delay networks; it takes long time to recover after backoff which highly impacts bandwidth utilization. TCP average window is related with event loss probability as follows

$$\overline{w} \approx \sqrt{\frac{3}{2bp}} \quad \text{or} \quad p \approx \sqrt{\frac{3}{2bw^2}}$$

TCP is designed to interpret all packet losses as congestion signals that upper bounds $\overline{w}$ with $\sqrt{\frac{3}{2bp_t}}$ where p_t represents the transmission error rate with value of 10-7 in optical fiber networks and higher for wireless loss networks.

Illinois algorithm is designed to use packet loss information as primary congestion signal in order to determine window size direction change. Queuing delay is used as secondary congestion signal to adjust window size pace change. It is clamed that following requirements are satisfied by newly designed protocol:

Intra protocol requirements like: Efficiency, Intra Protocol Fairness, Responsiveness and Heavy Congestion Avoidance.

Choice of larger α and smaller β value represent simplest way to achieve large window size. Rapid increase and small window size decrease can cause large packet drops during congestion. This event is called heavy congestion and can lead to undesirable consequences. Often timeouts caused by heavy congestion make TCP to enter slow start phase more often causing underutilization;

Router Independence is achieved when protocol is designed to work well regardless router performances like buffer size and queue management algorithm;

Protocol should be robust against noise during congestion signal measurements, especially when it uses queuing delay as congestion measure because queuing delay is highly affected by noise.

Inter protocol requirements represent compatibility in a way that in low speed networks it should achieve similar rate as standard TCP and in high speed networks standard TCP should not suffer significant throughput loss when coexists with new designed protocol;

Incentive to switch means that when switching to new protocol usage higher average user throughput should be achieved in a network that accommodates both standard TCP and newly designed high speed TCP protocol.

Illinois tries to achieve concave window curve which is the main reason why it tries to set α large when the network congestion is far and small when it is close to congestion. β should be set small when the network is far from congestion and large when it is close to congestion. This should be satisfied if we want to achieve better throughput utilization in case when packet losses are not caused by congestion. One of the main TCP protocol

goals is network congestion level measuring and forecasting it before it really happens. Queuing delay measure is used by Illinois. When average queuing delay d_a is small the sender assumes that connection is not imminent so it sets large α and small β, when d_a is large the sender assumes that the connection is imminent and sets small α and large β. This results with following $\alpha = f_1(d_a)$ and $\beta = f_2(d_a)$ where $f_1(d_a)$ and $f_2(d_a)$ are decreasing and increasing functions which result with concave window curve.

These kinds of algorithms are named Concave-AIMD algorithms or C-AIMD they use loss to determine direction and use delay to adjust the pace of window size change. Loss is primary and delay is secondary congestion signal. Problem occurs when using queuing delay to control congestion because delay cannot be measured accurately mainly because *RTT* measurements are noisy. Noisy *RTT* measurements could significantly degrade protocol performances. There are algorithms that use delay component as secondary signal and are much more noise robust during *RTT* measurement. $f_1(d_a)$ and $f_2(d_a)$ can be chosen at numerous ways. TCP Illinois is special case of C-AIMD algorithm that uses the following functions:

$$\alpha = f_1(d_a) = \begin{cases} \alpha_{\max} & if\ d_a \le d_1 \\ \dfrac{k_1}{k_2 + d_a} & otherwise \end{cases}$$

$$\beta = f_2(d_a) = \begin{cases} \beta_{\min} & if\ d_a \le d_2 \\ k_3 + k_4 d_a & if\ d_2 < d_a < d_3 \\ \beta_{\max} & otherwise \end{cases}$$

$f_1(d_a)$ and $f_2(d_a)$ are let to be continuous, represented with $\alpha_{\max} = \dfrac{k_1}{k_2 + d_1}$, $\beta_{\max} = k_3 + k_4 d_3$ and $\beta_{\min} = k_3 + k_4 d_2$. Suppose that *dm* is the maximum average queuing delay and let $\alpha_{\min} = f_1(d_m)$, written with exact equation it receives the following form $\alpha_{\min} = \dfrac{k_1}{k_2 + d_m}$. Above given conditions aloud us to obtain the following:

$$k_1 = \frac{(d_m - d_1)\alpha_{\min}\alpha_{\max}}{\alpha_{\max} - \alpha_{\min}} \quad \text{and} \quad k_2 = \frac{(d_m - d_1)\alpha_{\min}}{\alpha_{\max} - \alpha_{\min}} - d_1$$

$$k_3 = \frac{\beta_{\min} d_3 - \beta_{\max} d_2}{d_3 - d_2} \quad \text{and} \quad k_4 = \frac{\beta_{\max} - \beta_{\min}}{d_3 - d_2}$$

Illinois keeps all NewReno features except AIMD algorithm. In CA phase, the sender measures each acknowledgement *RTT* and averages *RTT* measurements over the last w acknowledgements (one *RTT* interval) to derive the average *RTT*. The sender records maximum and minimum ever seen *RTT* average as *Tmax* and *Tmin*, respectively computes the average maximum queuing delay *dm=Tmax-Tmin* and the current average queuing delay *da=Ta-Tmin*. The sender chooses following parameters $0 < \alpha_{\min} \le 1 \le \alpha_{\max}$ and $0 < \beta_{\min} \le \beta_{\max} \le \frac{1}{2}$, $w_{thresh} > 0$, $0 \le \eta_1 < 1$, $0 \le \eta_2 \le \eta_3 \le 1$. The sender sets $d_i = \eta_i d_m$ for $i = 1,2,3$ and computes k_i for $i = 1,2,3,4$ after obtaining *k* value it computes the values of α and β according the given equations. $\alpha \leftarrow 1$ and $\beta \leftarrow \frac{1}{2}$ if $w < w_{thresh}$. k_i ($i = 1,2,3,4$) values are updated if *Tmax* or *Tmin* are updated. α and β are updated once per *RTT*.

$$w \leftarrow w + \frac{\alpha}{w} \quad for \quad each \quad ack$$

$$w \leftarrow w - \beta w \quad loss \quad (3 \; duplicate \quad ack) \; in \; last \quad RTT$$

In case of timeout the sender sets the slow start threshold to be $\frac{w}{2}$ and enters slow start phase, it resets $\alpha = 1$ and $\beta = \frac{1}{2}$. α and β values are unchanged one *RTT* after slow start phase ends. Fast recovery and fast retransmission characteristics of NewReno are retained by Illinois. Illinois uses SACK features. It increases robustness against sudden fluctuations during delay measurement.

α adaptation is presented as key Illinois feature, β adaptation as a function of average queuing delay is relevant in networks where non congestion losses are present (wireless, extremely high speed networks). Losses in wireless networks are mainly caused by channel fluctuation characteristics and it is desired sharp window size reductions to be avoided. Illinois has advantage because it reduces the window size but the reduction is small because of the small queuing delay.

4. Loss Base with Bandwidth Estimation High Speed TCP Proposals

4.1. Fast TCP

Fast TCP aims to transform high speed long latency networks in steady efficient and fair. It exploits delay as a congestion measure because of several advantages. Queuing delay is more accurate estimated than loss probability mainly because packet losses in large bandwidth delay product networks are rare events and loss samples provides coarser information than queuing delay samples.

Delay measurements are nosy like those of loss probability. Each measurement of packet loss provides one bit information of noise filtering and each measurement of

queuing delay provides multi bit information. This makes equation-based implementation to transform the network in steady state with target fairness and high utilization. Queuing delay dynamics seems to have right scaling with respect of network capacity which helps to maintain stability as network capacity scales up. As is known congestion control algorithm can be described in two levels. Flow level design aims to achieve fairness, stability, high utilization, low queuing delay and losses. Packet level design implements the flow level goals constrained by end to end control characteristics. Like we stated before the AIMD algorithm can be written in the following form:

$$w \leftarrow w + \frac{1}{w} \quad CA \quad phase \quad for \quad received \quad ack$$

$$w \leftarrow w - \frac{1}{2} w \ in \ case \ of \ loss \ / drop$$

This equation describes packet level model and some flow level properties like throughput fairness and stability. Flow level model of AIMD algorithm can be described as follows:

Let $w_i(t)$ denote the window of source i, which increases by 1 (it should be equal to $1-p$ packets but p is small so roughly it equals to 1) packet per *RTT* and decreases by $x_i(t)p_i(t)\frac{1}{2}\frac{4}{3}w_i \quad packets$ per unit time where $x_i(t) = \frac{w_i(t)}{T_i(t)}$, *Ti* represents the round trip time, *pi(t)* is the end to end loss probability in period *t*. $\frac{4}{3}w_i(t)$ is peak window size which gives the average window of $w_i(t)$. Flow AIMD model is given with

$$\dot{w}_i(t) = \frac{1}{T_i(t)} - \frac{2}{3} x_i(t) p_i(t) w_i(t)$$

When $\dot{w}_i(t) = 0$ it results with the known formula $\frac{1}{\sqrt{p}}$ which relates the loss probability with the window size at following manner $\hat{p}_i = \frac{3}{2\hat{w}_i^2}$ .These equations describe the flow level and equilibrium point respectively when Reno TCP is observed.

For sure it can be stated that all TCP versions have same flow level dynamic structure. Let us define

$$k_i(w_i, T_i) = \frac{1}{T_i} \quad u_i(w_i, T_i) = \frac{1.5}{w_i^2}$$ and

If we have in mind that $w_i = x_i T_i$ then $\dot{w}_i$ can be expressed with $\dot{w}_i(t) = k(t)\left(1 - \frac{p_i(t)}{u_i(t)}\right)$

This equation represents general form in which different functions can be substituted and it will describe all known TCP versions. This is true because TCP versions mainly differ in the choice of *ki*, marginal utility functions *ui* and whether the congestion measure *pi* is loss probability or queuing delay.

$k_i(w_i, T_i)$ denotes gain function choice that determines dynamic protocol properties like stability, responsiveness and does not affect the equilibrium properties.

$u_i(w_i, T_i)$ represents marginal utility function choice that determines equilibrium properties like equilibrium rate allocation and its fairness.

p_i or congestion measure choice is limited with loss probability or queuing delay.

Flow level has goal to equalize marginal utility function $u_i(t)$ with end to end congestion measure $p_i(t)$. Direction and window size adjustments are based on the difference between the ratio $\frac{p_i(t)}{u_i(t)}$ and the target of 1. This approach eliminates packet level oscillations due to congestion signal binary nature, $p_i(t)$ precise estimation is mandatory. Without explicit feedback $p_i(t)$ can be loss probability. Queuing delay can be more precise estimated than loss probability mainly because the small loss probability value and the fact that packet loss in high speed networks is rear event. Packet loss measurement provide one bit information, queuing delay measurement provide multi bit information that allows equation based implementation to stabilize the network in steady state with target fairness and high utilization. At flow level feedback system dynamics is important to be stable in presence of delay as the network capacity increases. Queuing delay dynamic seems to have right scaling with respect to the network capacity that highly maintains (network) stability as network capacity grows.

New algorithm should be designed to have small window adjustment when current state is close to equilibrium and large otherwise. This behavior should be independent of equilibrium point value. Most of the protocols (Reno, HSTCP, STCP, etc where the window adjustment depend only of the current window size and does not take care for the current state) does not fulfill the given condition. This approach avoids the problem of slow increase and drastic decrease as the network scales up. Multi bit congestion measure approach eliminates the packet level oscillations due to binary feedback. Using the queuing delay as a congestion measure $p_i(t)$ allows network stabilization in the region below overflowing point when the buffer size is sufficiency large. This is important because stabilization at this operation point eliminates large queuing delay and unnecessary packet loss.

The algorithm must adopt α_i in order to maintain small but sufficient queuing delay. Window control algorithm must be stable. Queuing delay usage naturally scales with capacity. FAST TCP creators have designed two level congestion control algorithm.

At flow level the main goal is to design a class of function pairs $u_i(w_i, T_i)$ and $k_i(w_i, T_i)$ in order to describe the system with $\dot{w}_i(t) = k(t)\left(1 - \frac{p_i(t)}{u_i(t)}\right)$ and $p_i(t)$. This is the case when the equilibrium is fair, efficient and stable in presence of feedback delay. At packet level the algorithm should be designed in order to deal with issues ignored by flow level model. It should correct the anomalies in order to achieve flow level goals (burstiness control, loss recovery and parameter estimation). Protocol implementation

takes in consideration following conditions: various system components should be determined; the flow level algorithm should be translated into packet level algorithm; packet level algorithms should be implemented in a specific operating system.

Point of view is to separate the congestion control mechanism in several components: data control, window control, burstiness control and estimation. Each of these components is functionally independent and can be designed and upgraded independently. Data control determines which packet to be transmitted; window control determines how many packets to transmit at RTT timescale, burstiness control determine when to transmit these packets. Component decision is done on estimation component base.

Estimations are calculated based on various input parameters. In case when positive acknowledgement is received the protocol calculates RTT for corresponding data packet, it updates the average queuing delay and the minimum RTT . In case of negative acknowledgement, loss indication is generated for this data packet to the rest of the components. Estimation component generates a multi-bit queuing delay sample and a one-bit loss-or-no-loss sample for each data packet. The queuing delay is smoothed by the weight $\eta(t) = \min\ (3/w_i(t), 1/4)$ which depends of the window $w_i(t)$ at time t , as follows:

$k-th$ RTT sample $T_i(k)$ updates the average RTT denoted as $\overline{T_i}(k)$ according the equation $\overline{T_i}(k+1) = (1-\eta(t_k))\overline{T_i}(k) + \eta(t_k)T_i(k)$

t_k is the time at which the $k-th$ RTT sample is received. $d_i(k)$ is the minimum RTT observed so far, the average queuing delay is estimated as: $\hat{q}_i(k) = \overline{T_i}(k) - d_i(k)$. The weight $\eta(t)$ is usually smaller than the weight (1/8) used in TCP Reno. Average RTT , $\overline{T_i}(k)$ attempts to track the average over congestion window. During each *RTT*, entire window worth of RTT samples are received if every packet is acknowledged. If delayed ACK is used, the number of queuing delay samples is reduced so $\eta(t)$ should be adjusted accordingly.

Window control determinates the window size based on queuing delay and packet loss provided by the estimation component. This protocol design uses the same algorithm for congestion window computation independent of the sender state.

FAST TCP uses queuing delay and packet loss during congestion avoidance phase in accordance with:

$$w \leftarrow \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{RTT_{base}}{RTT}w + \alpha(w, qdelay)\right)\right\}$$

γ receives values in the range of 0 and 1, RTT_{base} is the minimum *RTT* observed, *qdelay* is the average end to end queuing delay.

In this algorithm congestion window changes its value over two *RTTs*, it is updated in the first one and it keeps the same value in the second. In this case $\alpha(w, qdelay)$ is chosen to be constant all the time, assuring linear convergence when *qdelay* is null. Other mechanism can be used, $\alpha(w, qdelay)$ can be constant when *qdelay* receives nonzero value and value proportional to the window when *qdelay* is zero, $\alpha(w, qdelay) = aw$.

FAST performs multiplicative increase and grows exponentially at neighborhood rate when *qdelay > 0*. α(*w, qdelay*) switches to a constant α, as is described in Theorem 2, window converges exponentially to equilibrium at a different rate that depends on *qdelay*. α represents number of packets that each flow attempts to maintain in the network buffer(s) at equilibrium (similar as TCP Vegas). When a packet loss is detected, FAST halves its window and enters loss recovery. The goal is to back off packet transmission quickly when severe congestion occurs, in order to bring the system back to a regime where reliable *RTT* measurements are available for window adjustment

$$w \leftarrow \min\left\{2w, (1-\gamma)w + \gamma\left(\frac{RTT_{base}}{RTT}w + \alpha(w, qdelay)\right)\right\}$$

to work effectively. A source does not react to delay until it exits loss recovery.

TCP is an event based so the main actions are triggered by events. Flow level algorithms have to be translated in event based packet level.

FAST TCP recognizes four events: ACK reception – the estimation component computes the average queuing delay, the burstiness control component determines whether packets can be injected into the network; after packet transmission – the estimation component records a time stamp and the burstiness control updates the corresponding data structures for keeping records, at a constant time interval this is checked on the arrival of each acknowledgement, window control calculates new window size; at the end of the *RTT* – burstiness reduction calculates the target throughput using window and *RTT* measurements in the last RTT; on each packet loss.

The algorithm is designed and analyzed at flow level in order to be sure that it meets requested fairness and stability. The analysis gives freedom to choose necessary congestion control components. Flow level algorithm is translated in a packet level consisted of event based tasks.

4.1.1. Equilibrium and Stability of Window Control Algorithms

Network can be represented by set of links with finite capacities c_l. The network is shared by set of unicast flows that are identified by sources. Round trip propagation delay of source i is denoted by d_i. R denotes a routing matrix where $R_{li} = 1$ if source i uses link l and receives 0 value otherwise. $p_l(t)$ denotes link l queuing delay at time t. $q_i(t) = \sum_l R_{li} p_l(t)$ denotes round trip that can be expressed in vector form as $q_i(t) = R^T p(t)$. Round trip time of source *i* is given with $T_i(t) = d_i + q_i(t)$. Each source *i* periodically adapts $w_i(t)$ according:

$$w_i(t+1) = \gamma\left(\frac{d_i w_i(t)}{d_i + q_i(t)} + \alpha_i(w_i(t), q_i(t))\right) + (1-\gamma)w_i(t)$$

when α_i is a constant the equation can be written as:

$$w_i(t+1) = w_i(t) + \gamma_i(\alpha_i - x_i(t)q_i(t))$$

we know that Vegas updates the window according:

$$w_i(t+1) = w_i(t) + \frac{1}{T_i(t)} \text{sgn}\ (\alpha_i - x_i(t) q_i(t))$$

Having in mind above equations we can conclude that FAST TCP is enhanced version of Veges. If we observe $w_i(t+1) = \gamma\left(\frac{d_i w_i(t)}{d_i + q_i(t)} + \alpha_i(w_i(t), q_i(t))\right) + (1-\gamma) w_i(t)$ where γ receives values in the range of 0 and 1 at time *t* and α_i is defined as

$$\alpha_i(w_i, q_i) = \begin{cases} a_i w_i & \text{if } q_i = 0 \\ \alpha_i & \text{otherwise} \end{cases}$$

we can conclude that FAST TCP defers from the rest of the protocols by assumption that the source send rate defined as $x_i(t) = \frac{w_i(t)}{T_i(t)}$ can not exceed the throughput it receives. As a consequence of this assumption we obtain that the link queuing delay vector c, $p(t)$ is determined implicitly by the instantaneous window size in a static manner.

$w_i(t) = w_i$ for all *i* , link queuing delays $p_l(t) = p_l \geq 0$ for all *l* are given by

$$\sum_i R_{li} \frac{w_i}{d_i + q_i} \begin{cases} = c_l & \text{if } p_l > 0 \\ \leq c_l & \text{if } p_l = 0 \end{cases} \quad \text{where} \quad q_i(t) = \sum_l R_{li} p_l(t)$$

The equilibrium values of windows and delays $\hat{w}$ and $\hat{p}$ can be characterized as follows:

The utility maximization problem $\max_{x \geq 0} \sum_i \alpha_i \log x_i \qquad Rx \leq c$ and the dual problem

$$\min_{p \geq 0} \sum_l c_l p_l - \sum_i \alpha_i \log \sum_l R_{li} p_l$$

Theorem 1: Suppose *R* has full row rank. The unique equilibrium point of the network ($\hat{w}$, $\hat{p}$) defined by the above equations exists and is such that $\hat{x} = \frac{w_i}{d_i + \hat{q}_i}; \forall i$ is the unique maximizer of $\max_{x \geq 0} \sum_i \alpha_i \log x_i; \quad Rx \leq c$ and $\hat{p}$ is the unique minimizer of $\min_{p \geq 0} \sum_l c_l p_l - \sum_i \alpha_i \log \sum_l R_{li} p_l$.

This implies that equilibrium rate $\hat{x}$ is α_i -weighted proportionally fair.

We can conclude that FAST TCP has the same equilibrium properties as TCP Vegas. Its throughput is given with $x_i = \frac{\alpha_i}{q_i}$ and it does not penalize sources with large propagation delay d_i. This relation implies that, in equilibrium, source *i* maintains αi packets in the buffers along its path. The total amount of network buffering must be at least $\sum_i \alpha_i$ in order to reach the equilibrium.

We now turn to the algorithm stability analysis. General network global stability in presence of feedback delay is open problem. State-of-the-art results either prove global stability while ignoring feedback delay, or local stability in the presence of feedback delay. FAST TCP stability result is restricted to a single delay free link.

Theorem 2: Suppose there is a single link with capacity *c*. The network defined with upper equations is globally stable, and converges geometrically to unique equilibrium ($\hat{w}$, $\hat{p}$).

The basic proof idea is to show that the iteration from *w(t)* to *w(t+1)* defined with

$$w_i(t+1) = \gamma\left(\frac{d_i w_i(t)}{d_i + q_i(t)} + \alpha_i(w_i(t), q_i(t))\right) + (1-\gamma)w_i(t)$$

;

$\alpha_i(w_i, q_i) = \begin{cases} a_i w_i & if\ \ q_i = 0 \\ \alpha_i & otherwise \end{cases}$; $\sum_i R_{li} \frac{w_i}{d_i + q_i} \begin{cases} = c_l & if\ \ p_l > 0 \\ \le c_l & if\ \ p_l = 0 \end{cases}$ is a contraction mapping. *w(t)* converges geometrically to the unique equilibrium.

Corollary 3: 1) Starting from any initial point ($\hat{w}$, $\hat{p}$), the link is fully utilized, i.e., equality holds in $\sum_i R_{li} \frac{w_i}{d_i + q_i} \begin{cases} = c_l & if\ \ p_l > 0 \\ \le c_l & if\ \ p_l = 0 \end{cases}$, after a finite time.

2) The queue length is lower and upper bounded after a finite amount of time.

This analysis is defined as Global stability analysis that implements the flow model with help of usage of dual and primal algorithms that provide solution of the defined issue.

The analysis stands for most of the TCP algorithms mainly because they differ in the utility function choice, the choice of the constants and the stable point value.

4.2. New Vegas

New Vegas uses round trip time estimation to sense network congestion based on Vegas algorithm. As we know Vegas uses the difference between expected and actual source rate.

$$Expected = \frac{w}{RTT_{base}}$$

$$Actual = \frac{FlightSize}{RTT}$$

$$Diff = (Expected - Actual)RTT_{base}$$

w represents the congestion window value, RTT_{base} is the minimum round trip time expected by connection. *Flight Size* is the number of not acknowledged packets and RTT is the round trip time expected by target packet. Vegas calculate normalized difference for each incoming ACK. When the algorithm operates in slow start, threshold parameter γ is maintained, it usually has value of 1. When *Diff* is less than γ, congestion window is increased by one packet every round trip time. During slow start, Vegas grows its window exponentially until it reaches slow start threshold w_{th} or *Diff* become larger than γ when it enters congestion avoidance phase. Two threshold parameters α and β are maintained. If *Diff* is less than α, the congestion window grows linearly in the next round trip time. If *Diff* is larger than β, Vegas decrease its window linearly in the next round trip time; if *Diff* is between α and β the window is not changed. In case of packet loss detection via timeout expiration the slow start threshold is set at half of the current congestion window value and the congestion window is set at 1, the algorithm enters slow start phase. In case of packet loss detection via three duplicated *ACK* the algorithm performs fast retransmit and fast recovery like Reno does. After fast retransmit Vegas set its congestion window at ¾ of the current one and enters congestion avoidance phase. New Vegas set threshold parameters α, β at default values $\alpha 0$, $\beta 0$. When *ACK* arrives at the source, New Vegas updates the threshold according:

$$\alpha = \alpha + 1; \quad when \quad RTT > RTT_{old} \quad and \quad W \leq W_{old}$$

$$\beta = \beta + 1; \quad when \quad RTT > RTT_{old} \quad and \quad W \leq W_{old}$$

$$\alpha = \alpha - 1; \quad RTT \leq RTT_{old} \quad and \quad \alpha > \alpha_0$$

$$\beta = \beta - 1; \quad RTT \leq RTT_{old} \quad and \quad \alpha > \alpha_0$$

RTT is the round trip time of the target packet, RTT_{old} is the previous one, w is the current congestion window size and w_{old} is the previous one. Threshold updates are done once per round tip time. In case of packet loss caused by timer expiration or arrival of three duplicated *ACK*, threshold parameters are set at initial values $\alpha 0, \beta 0$. First algorithm condition indicates that congestion is happening, new RTT value is larger than the previous one and New Vegas does not increase its congestion window because it does not consider that it is congestion responsible. It is allowed the number of packets that can be put in the network buffers to be increased by the source. If RTT start to decrease TCP NewVegas reduces its threshold. New Vegas request only sender side modification, it uses existing thresholds modification but it does not differ drastically in performances than Vegas.

4.3. TCPW-A

TCP Westwood with agile probing is algorithm that employs two mechanisms. In case of initial connection phase or after timeout restart, TCPW-A uses Agile probing mechanism and repeatedly resets *ssthresh* based on Eligible Rate Estimation (*ERE*), forcing exponential *cwnd* climb which results with fast convergence to more appropriate *ssthresh* value. During congestion avoidance phase the protocol invokes Agile probing upon received indication for unused extra bandwidth with help of Load Gauge (LG) scheme. When bandwidth delay product increases one of the issues that occur is setting *ssthresh* value. Initial *ssthresh* is set at predefined value that ranges from 4k bytes up to maximum predefined possible value. Setting *ssthresh* to arbitrary value can cause TCP performance to suffer from following problems: if *ssthresh* is set at relative high network Bandwidth Delay Product value, exponential window increase generates packets too fast and causes multiple losses at connection throughput; If *ssthresh* is set at low value the connection will exit slow start phase and will switch to linear increase that will result with poor connection utilization.

In TCP Westwood, the sender continuously monitors receivers *ACKs*, computes its current Eligible Rate Estimate and adaptively computes T_k, an interval over which ERE sample is calculated. A ERE sample is computed by the amount of data bytes successfully delivered in T_k. T_k depends on the congestion level defined as a difference between expected and achieved rate presented with following equation:

$$T_k = RTT \frac{\frac{cwin}{RTT_{\min}} - RE}{\frac{cwin}{RTT_{\min}}}$$

$RTT_{\min}$ represents the minimum RTT value of all acknowledged packets during communication and RTT presents smoothed RTT measurement value. Expected connection rate is given with $\frac{cwnd}{RTT_{\min}}$. RE denotes the achieved rate computed on base of acknowledged data amount during last RTT, it is exponentially averaged over time using a low pass filter. When there is no queuing time which indicates that there is no congestion the expected and the achieved rates have almost same value that impacts T_k value which is proportional with the difference of the mentioned variables. The result is small T_k value and ERE become close to a packet pair measurement. In case of congestion RE is much smaller than $\frac{cwnd}{RTT_{\min}}$ because of the larger queue delay and it results with increased T_k. After ERE samples are computed a discrete version of continuous first order low pass filter using the Tustin approximation is used to obtain smoothed ERE. In current TCPW implementation, upon packet loss (indicated by 3 duplicated *ACK* or a timeout) the sender sets *cwnd* and *ssthresh* based on its current ERE. TCPW sets its *ssthresh* to *ERE*RTTmin*, and *cwnd* to *ssthresh* (*cwnd* is set at 1 in case of timeout).

Agile probing used in this protocol tries to improve TCP performances during start up behavior over high speed long distance network with help of Load Gauge. It uses ERE to adaptively and repeatedly reset *ssthresh*. When Agile Probing is used and the current *ssthresh* is lower than ERE, the sender resets *ssthresh* accordingly and exponentially

increases *cwnd* otherwise, *cwnd* increases linearly to avoid overflow. This is the way how Agile Probing probes the available network connection bandwidth and allows the protocol to exit Slow-start phase close to an ideal window that corresponds to its path bandwidth share. With help of repeating cycles of linear and exponential increase, *cwnd* adaptively converges in timely manner to the desired window value, enhancing Slow Start link utilization.

Load Gauge (LG) mechanism is used to monitor the available bandwidth and to invoke Agile Probing accordingly. During congestion avoidance phase, connection constantly monitors the congestion level. If a TCP sender detects light load conditions, indicating that connection may be eligible for more bandwidth, the connection invokes Agile Probing to capture such bandwidth providing improved utilization. If the network is not congested and extra bandwidth is available, RE will increase as *cwnd* increases. If the network is congested, RE flattens despite the *cwnd* increase. RE represents the Achieved Rate corresponding to the Expected Rate 1.5 times *RTT* earlier thus we must use it in a comparison, the corresponding Expected Rate is *(cwnd - 1.5)/RTTmin.* RE tracks the Expected Rate in non-congestion conditions, but flattens, remaining close to the initial Expected Rate (*ssthresh / RTTmin*) under congestion. Congestion Boundary is defined with

$$\text{Congestion Boundary} = \beta \text{ ExpectedRate} + (1 - \beta) \text{ InitialExpectedRate} \quad \text{for } 0 < \beta < 1$$

RE may fluctuates crossing above and below the Congestion Boundary. To declare light load condition a (non-congestion) counter is used, which increases by one every time when RE is above the Congestion Boundary and decreases by one if RE is below the Congestion Boundary. If β is greater than 0.5, the Congestion Boundary line gets closer to the Expected Rate. The algorithm can be more conservative by setting $\beta > 0.5$. Even if the LG algorithm can accurately indicate light load condition, there is always possibility that the network becomes congested immediately after it switches to Agile Probing phase. One such scenario is after a buffer overflow at the bottleneck router. Many of the TCP connections may decrease their *cwnd* after a buffer overflow, and congestion is relieved in a short time period. The LG mechanism in some connection may observe light load and invoke Agile Probing. We have to mention that the erroneous indication is not a serious problem. Unlike New Reno Slow Start exponential *cwnd* increase, the TCP connection adaptively seeks fair share estimate in Agile Probing mode. If the network has already been congested when a new Agile Probing begins, the "Agile Probing" connection will not increase much *cwnd*, and will go back to linear probing soon enough.

4.4. Westwood+

Westwood+ algorithm employs logarithmic window increase function in loss free environment. It uses linear additive window increase function. In case of packet loss the window update follows the following equation:

$$w \leftarrow \max\left(2, \frac{BWE * RTT_{\min}}{seg\_size}\right)$$

BWE represents estimated end to end bandwidth and *RTTmin* is the minimum *RTT* trip time measured over connection duration, *seg_size* denotes TCP segment size in bits.

Logarithmic function is proposed to replace the linear congestion window evolution and it operates during congestion avoidance phase, it is important to note that slow start phase remains unchanged.

$$w \leftarrow w + \frac{w_{\max} - w}{\alpha w} \quad for\ each\ \ ack$$

Wmax defines congestion window value at which the last packet loss event was detected. *w* chooses value between current *w* and *wmax* scaled by *α*. *α* dynamically controls increase algorithm aggressiveness level. It is increased by factor of two in case of detected packet drop before *w* reaches *wmax* value and is decreased by the same factor for every errorless increase on the same interval. α lower bound is two because values lower than two make the increase very aggressive.

If we take in consideration previous window growth equation that is more aggressive for smaller *w* values and is less aggressive as *w* approaches *wmax* value we can conclude that the protocol requests the algorithm to be at last aggressive as the standard TCP, this is achieved with bound of the minimum window increase to one packet per window, given with:

$$w \leftarrow w + \max\left(\frac{w_{\max} - w}{\alpha w}, \frac{1}{w}\right)$$

This proposal has similar concept as BIC TCP, the novelty allows window increase always to follow a logarithmic behavior when *w*< *wmax*, aggressiveness level is controlled by α. This modification ensures algorithm to behave well for small and large window values.

Decrease Westwood+ congestion control mechanism is untouched; these increase functions should be active only when *w* is in range of actually provided BDP by end-to-end link and the total available BDP. Aggressiveness level of the discussed strategies corresponds with the window increase speed. TCP flow performance is proportional with the window size in every moment of time, i.e., larger amount of time spent at higher window value results with increased throughput performances. Different congestion window strategies can be compared by integral squares of increase functions.

4.5. ARENO

AReno dynamically adjusts TCP response function based on congestion estimation with help of RTT measurement. It uses faster congestion window increase in case of underutilized network resources and uses Reno when congestion occurs. AReno is TCP Westwood BBR- (Buffer and Bandwidth Estimation) based protocol which enhances TCP Westwood friendliness in networks with varying buffer capacities, flows with varying RTT 's and with/without used AQMs. It estimates network congestion level with help of RTT measurements which indicate if the packet loss is caused by congestion or not. In case of unrecognized congestion loss the congestion window is reduced in accordance with TCPW bandwidth estimation. When congestion happens the protocol uses TCP Reno window reduction mechanism. AReno adjusts its response function based on congestion measurements introduced by TCPW-BBE. In case of competing TCP-AReno and TCP-Reno flows, the protocol detects increased RTT values and thus increased congestion level. In this case, it recognizes packet losses that are likely to happen and adopts TCP-Reno response function (i.e. increasing the congestion window by 1MSS/RTT). AReno determines RTT range between $RTT_{\min}$ and RTT_{cong} value and

is expected *RTT* to be in it when congestion packet loss occurs. RTT_{cong} is estimated using RTT .

$$RTT_{cong}^{j} = (1 - a)RTT_{cong}^{j-1} + aRTT^{j}$$

Used values are measured right before the packet loss. RTT_{cong} is updated upon each packet loss event where the index *j* identifies the *j-th* packet loss event, *a* represents exponential smoothing factor. When the actual RTT is close to $RTT_{\min}$, TCP AReno recognizes underutilized network; when RTT is close to RTT_{cong} it recognize congested network. The congestion level *c* is defined as:

$$c = \min\left(\frac{RTT - RTT_{\min}}{RTT_{cong} - RTT_{\min}}, 1\right)$$

Its value is in range of $0 \le c \le 1$

AReno manages its window with help of w_{base} that represents the base part and w_{prob} defined as probing *cwnd* part. The first one maintains Reno congestion window size, the second is used quickly to fill the bottleneck link. Increase window functions are given below:

$$w_{base} = w_{base} + \frac{1MSS}{w}$$

$$w_{probe} = \max\left(w_{probe} + \frac{w_{inc}}{w}, 0\right)$$

When the network is underutilized AReno sets w_{inc} close to the maximum value $w_{inc}^{\max}$ that should be adequately sized according the bottleneck link capacity, pipe size and congestion window size. Several ways are proposed. During the initial version the protocol creator has found that it is most common to provide linear scaling of $w_{inc}^{\max}$ according the estimated bottleneck link capacity *B*.

$$w_{inc}^{\max} = \frac{B}{M} MSS$$

M represents scaling factor; *B* represents estimated time sequence of ACK-ed sequence numbers like it is presented in TCPW. As the congestion level *c* increase AReno decreases its w_{inc} value even lower than 0 which enables congestion window of coexisting Reno flow to have time to catch up the AReno flow. When *c* approaches 1 and packet loss is obvious to happen w_{inc} is gradually increased so that it approaches null value, in this case the base window is dominant and the protocol behaves like Reno. AReno is designed to have U curve w_{inc} is calculated as a combination of exponential and linear function given with following equation presented as a function of *c*

$$w_{inc}(c) = \frac{w_{inc}^{\max}}{e^{\alpha c}} + \beta c + \gamma$$

α determines underutilized network range, it's value should be small enough in order to allow *RTT* measurement errors. Improved Reno flow friendliness is assured when w_{probe} convergence to null when a packet loss is like to happen. If we have in mind the given constraints β and γ can be obtained as follows:

$$\beta = 2 w_{inc}^{\max} \left(\frac{1}{\alpha} - \frac{\frac{1}{\alpha} + 1}{e^{\alpha}} \right)$$

$$\gamma = 1 - 2 w_{inc}^{\max} \left(\frac{1}{\alpha} - \frac{\frac{1}{\alpha} + \frac{1}{2}}{e^{\alpha}} \right)$$

AReno window reduction is based on congestion measurement, the congestion window is halved in case of congested network and packet loss is expected to happen. Reduction is mitigated in case of underutilized network and the window reduction is done according the following equation

$$w_{base} = w * w_{dec} = \frac{w}{1 + c}, \quad w_{probe} = 0$$

Exact parameter choice is presented in [20]

4.6. TCP Fusion

TCP Fusion uses Reno, Vegas and Westwood protocol properties. It is a loss based protocol that uses RTT metric category. The key concept is to use Reno metric in case of fully utilized network and to use more aggressive one otherwise. It maintains two congestion windows; the first one has Vegas and Westwood properties that provide efficiency in large leaky pipes. The other window use TCP Reno update mechanism, its value is increased by 1*MSS/RTT* and upon loss it is halved. Fusion adopts the bigger one as new congestion window size. In order to improve efficiency Fusion uses decrease parameter optimization based on TCPW-RE algorithm. After loss, the decrease parameter can be expressed as $\frac{RTT_{\min}}{RTT}$ where $RTT_{\min}$ and RTT are the minimum and RTT value just before packet loss. TCPW-RE reduces congestion window size to clear the buffer and makes the protocol Reno friendly only if the buffer capacity is equal to BDP where RTT grows up to $2 RTT_{\min}$. If the buffer capacity is larger than BDP the decrease value is less than ½ upon a congestion loss and TCPW-RE cannot obtain a bandwidth share.

$$cwnd_{new} = \max\left(\frac{RTT_{\min}}{RTT} cwnd_{last}, \frac{cwnd_{last}}{2}\right)$$

In this case the threshold is set at ½. *cwndnew* and *cwndlats* denote the congestion window sizes just before and after packet loss respectively.

TCP Fusion has three functional phases: increase, decrease and steady phase. The switch is triggered by the number of packets in the bottleneck queue denoted as *diff*. *diff* value is estimated according

$$diff = cwnd \frac{RTT - RTT_{\min}}{RTT}$$

If *diff* is less than lower bound threshold the link is underutilized and the window size is rapidly increased. If *diff* is larger than the upper bound threshold the link is determined as utilized, early congestion is detected and the congestion window size is decreased to at least the lower bound threshold value of the bottleneck queue.

$$w_{new} = \begin{cases} w_{last} + \frac{W_{inc}}{w_{last}} & if \quad diff < \alpha \\ w_{last} + \frac{(-diff + \alpha)}{w_{last}} & if \quad diff > 3\alpha \\ w_{last} & otherwise \end{cases}$$

$$w_{new} = w_{reno} \qquad if \quad w_{new} < w_{reno}$$

wnew, *wlast* and *wreno* are the congestion window sizes after update, before update and the last is Reno equivalent. α is the lower bound threshold used to switch phases and has huge protocol performance impact. When defining α value, several things should be considered. *winc* is increment parameter used rapidly to increase congestion window size. During protocol design phase friendliness with TCP Reno should be considered, α should receive small value to minimize the queuing delay that affects coexisting TCP Reno flows. When α receive very small value it becomes meaningless because of TCP timer granularity. *diff* is always larger than α except in case when *RTT=RTTmin*. In order Fusion phases adequately to be switched, α should be proportional to the link bandwidth. In case of coexisting of *N* Fusion flows each flow will try to put α packets in the bottleneck buffer, the router buffer size has to accommodate at least *N*a* packets. If the router buffer size is less than the requested value *diff* parameter will never receive up to α value. This indicates that all Fusion flows always will aggressively increase their congestion window sizes. The analysis indicates that α should be set to a small value according the number of coexisting Fusion flows. In order to satisfy this request it should be assumed that the buffers should be larger or equal at *G* packets, corresponding at queuing delay of *Dmin* in the bottleneck queue. *G* is given with:

$$G = \frac{BD_{\min}}{8 * packet\_size}$$

B represents the bottleneck bandwidth. In case when all *N* links employ Fusion algorithm since the total packets (*N*a*) are equal at *G* packets, α can be expressed as *G/N*. The number of coexisting flows is hard to be known just like the bandwidth. *B/N* can be approximated by achieved rate estimation:

$$\alpha = \frac{G}{N} = \frac{\frac{B}{N} D_{\min}}{packet\_size * 8} \approx \frac{RE * D_{\min}}{packet\_size * 8}$$

RE represents achieved rate estimation according TCPW-RE mechanism. This form of α fulfills the first requirement, α scales up the link capacity and receives small value in inverse proportion of the number of coexisting Fusion flows which was second requirement. Achieved *RE* rate is equal to *cwnd/RTT* and α can be written as

$$\alpha = \frac{RE * D_{\min}}{packet\_size * 8} = cwnd \frac{D_{\min}}{RTT}$$

This means that α accuracy is limited by TCP timer granularity *tcp_tick* which depends of the operating system. *Dmin* can be set to be equal at *tcp_tick* or higher value in order to absorb *RTT* oscillations under dynamic network conditions. *winc* is set according assumption that the routers will have buffer values higher or equal at *G* packets. *winc* is upper bounded with:

$$w_{inc} \leq G = \frac{B * D_{\min}}{8 * packet\_size}$$

Where *B* is the bandwidth estimation achieved by time sequence of ACK sequence numbers like in TCPW-BE.

5. Conclusion

TCP development was elevated at higher level mainly because of poor TCP performances in high speed networks. Congestion Avoidance phase was identified as main reason for this behavior. Numbers of TCP proposals were developed with aim to improve protocol utilization of high speed environment. Several issues should be addressed adequately during novel protocol design. Friendliness with legacy TCP, efficiency, fairness, responsiveness are only part of them. Solutions can be identified in three major groups: loss based, delay based and loss based with bandwidth estimation. Each of them is truing to find creative way to improve protocol performances. HSTCP and STCP were first developed high speed transport protocol versions. HSTCP has modified the response function at very low packet drop rates in order to achieve 10 *Gbps* speed. Its additive increase and multiplicative decrease parameters are function of current congestion window size when high congestion window value is reached. STCP is MIMD based window increase protocol. Both proposals were identified as *RTT* unfair when multiple flows with different *RTT's* are competing for the same bottleneck bandwidth. BIC TCP tries to solve *RTT* unfairness and synchronized loss problem with help of additive and binary search increase schemes. HTCP use increase parameter represented as a function of time in seconds since the last congestion event during fast mode of operation. Cubic was new creation designed to improve BIC anomalies, it simplifies BIC window control, improves its TCP friendliness and *RTT* fairness. Libra has introduced new designed increase and decrease parameters and provides improved TCP behavior.

Hybla is truing to solve *RTT* disparity problem, it tries to make segment transmission rate and *cwnd* change to be *RTT* invariant/independent. All mentioned protocols use packet loss as a congestion measure and phase switching trigger.

Delay based versions are less aggressive mainly because of the protocol nature (used congestion measure as phase switching trigger). They are last aggressive when sending around link capacity. It is discussed that worst delay based protocol performances are achieved when competing with loss based algorithms.

YeAH TCP operates in two different modes, fast and slow. In fast mode it increments the congestion window according aggressive STCP rule, while in slow mode it behaves as Reno. In which protocol state it will operate depends on the estimated number of packets in the bottleneck queue. $Q_{\max}$ and φ are tunable parameters. The first one represents maximum number of packets in single flow allowed to be kept in the buffers; $\frac{1}{\varphi}$ parameter represents maximum level of buffer congestion with respect to BDP. Africa uses Vegas delay metric to determine bottleneck link congestion. It operates in two modes, fast and slow, in fast it uses HSTCP protocol otherwise it use Reno mechanism. Compound TCP uses both delay and loss based approach to control the sending rate. The main idea of CTCP is to add scalable delay based component in standard TCP. It uses combination of loss and delay based approach of high speed congestion control. Illinois uses packet loss information to make decision when to increase or decrease the window size. It is designed to use packet loss information as primary congestion signal in order to determine window size change direction. Queuing delay is used as secondary congestion signal to adjust window size pace change. α adaptation is presented as key Illinois feature.

We can conclude that each packet loss measurement provides one bit of noise filtering information and each queuing delay measurement provides multi bit information. This assures equation-based implementation to transform the network in steady state with target fairness and high utilization. Queuing delay dynamics seems to have right scaling with respect of network capacity which helps maintaining stability as the network capacity scales up. It is known that congestion control algorithm can be designed in two levels. Flow level design aims achieving fairness, stability, high utilization, low queuing delay and losses. Packet level design implements flow level goals constrained by end to end control characteristics.

FAST TCP poses congestion control algorithm designed in two levels: main goal of flow level is to design a class of function pairs $u_i(w_i, T_i)$ and $k_i(w_i, T_i)$ in order system to be described with $\dot{w}_i(t) = k(t)\left(1 - \frac{p_i(t)}{u_i(t)}\right)$ and $p_i(t)$. This is the case when equilibrium is fair, efficient and stable in presence of feedback delay. At packet level the algorithm is designed to deal with issues ignored by flow level model. New Vegas uses round trip time estimation to sense network congestion based on Vegas algorithm. TCP Westwood with agile probing is algorithm which employs two mechanisms. During initial connection phase or restart after timeout, TCPW-A uses Agile probing mechanism and repeatedly resets *ssthresh* based on Eligible Rate Estimation (ERE). It forces *cwnd* in exponential climb which results with fast convergence to more appropriate *ssthresh* value. During congestion avoidance phase the algorithm invokes Agile probing upon indication of unused extra bandwidth with help of Load Gauge (LG) scheme. When bandwidth delay product increases one of the problems that occur is setting *ssthresh* value. Westwood+ algorithm employs logarithmic window increase function when no losses are experienced. Logarithmic function is proposed to replace linear congestion window evolution and is used during congestion avoidance phase; slow start phase remains unchanged. This

proposal has similar concept with BIC TCP, novelty allows window increase always to follow a logarithmic behavior when $w<wmax$, aggressiveness level is controlled by α. This modification ensures well protocol behave for small and large window values. AReno dynamically adjusts TCP response function based on congestion estimation with help of RTT measurement. It uses faster congestion window increase in case of underutilized network, when congestion occurs it uses Reno. AReno is based on TCP Westwood BBR (Buffer and Bandwidth Estimation) it enhance TCP Westwood friendliness in varying buffer capacities networks, flows with varying $RTT's$ and networks with/without AQMs. Protocol estimates the congestion level with help of RTT measurements that indicate the packet loss reason, was it caused by congestion or not. In case of unrecognized congestion loss, congestion window is reduced according TCPW bandwidth estimation mechanism. When congestion happens protocol uses TCP Reno window reduction function and halves congestion window value. AReno adjusts its response function based on congestion measurements introduced by TCPW-BBE. In case of flow competition between TCP-AReno and TCP-Reno flows, the mechanism detects increased RTT values and thus increased congestion level. TCP Fusion uses Reno, Vegas and Westwood protocol properties. It is a loss based protocol using RTT metric. Key concept is to use Reno metric in case of fully utilized network and more aggressive mechanism otherwise. It maintains two congestion windows; the first has Vegas and Westwood properties and provides efficiency in large leaky pipes, the other is updated according TCP Reno mechanism. Fusion adopts the larger as new congestion window size. It uses decrease parameter optimization based on TCPW-RE in order to improve protocol efficiency.

In this paper we have described the basic functional concepts of high speed TCP protocols and have detected space for additional improvements. Most of the used protocol phases switching parameters are tunable. Additional work can be done in order to modify and to improve the increase function used in congestion avoidance phase during fast mode protocol operation. Freedom is given when deciding which parameters to be used as a protocol phase switching trigger (packet loss, queuing delay, queuing delay with bandwidth estimation) and additional combinations can be offered. It is obvious that more advanced protocols can be designed using combination of flow and packet base equation models. Competitive analysis should be done in order to detect further possibilities of protocol improvements.

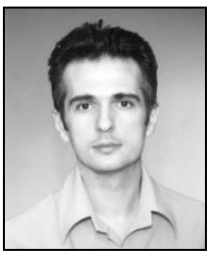
References

[1] S. Floyd, "RFC3649—HighSpeed TCP for large congestion windows," RFC, **(2003)**.
[2] S. Floyd, "High Speed TCP and Quick-Start for Fast Long-Distance High Speed TCP and Quick-Start for fast long distance networks", TSVWG, IETF, **(2003)** March.
[3] S. Floyd, Sylvia Ratnasamy and Scott Shenker, "Modifying TCP's Congestion Control for High Speeds", Very rough preliminary draft, **(2002)** May 5.
[4] T. Kelly, "Scalable TCP: improving performance in high speed wide area networks," Computer Communications Review, vol. 32, no. 2, **(2003)** April.
[5] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast, long distance networks," in Proc. IEEE INFOCOM, vol. 4, **(2004)** March, pp. 2514–2524.
[6] D. Leith, "H-TCP: TCP congestion control for high bandwidth-delay product paths," IETF Internet Draft, http://tools.ietf.org/html/draftleith-tcp-htcp-06, **(2008)**.
[7] I. Rhee and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," SIGOPS Operating Systems Review, vol. 42, no. 5, **(2008)** July, pp. 64–74.
[8] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "TCP Libra: Exploring RTT-Fairness for TCP," UCLA Computer Science Department, Tech. Rep. UCLA-CSD TR-050037, **(2005)**.
[9] C. Caini and R. Firrincieli, "TCP Hybla: a TCP enhancement for heterogeneous networks," International J. Satellite Communications and Networking, vol. 22, **(2004)**, pp. 547–566.
[10] A. Baiocchi, A. P. Castellani, and F. Vacirca, "YeAH-TCP: yet another high speed TCP," in Proc. PFLDnet, ISI, Marina Del Rey (Los Angeles), **(2007)** February, California.

[11] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: an adaptive and fair rapid increase rule for scalable TCP," in Proc. IEEE INFOCOM, vol. 3, **(2005)** March, pp. 1838–1848.
[12] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," **(2005)** July.
[13] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks."
[14] S. Liu, T. Basar, and R. Srikant, "TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks," in Proc. First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), **(2006)**.
[15] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "FAST TCP: motivation, architecture, algorithms, performance," IEEE/ACM Trans. Net., vol. 14, no. 6, **(2006)**, pp. 1246–1259.
[16] J. Sing and B. Soh, "TCP New Vegas: Improving the Performance of TCP Vegas Over High Latency Links," in Proc. 4th IEEE International Symposium on Network Computing and Applications (IEEE NCA05), **(2005)**, pp. 73–80.
[17] A. Vendictis, M. Bonacci, A. Baiocchi "TCP New Vegas: Providing Good TCP Performance in both Homogeneous and Heterogeneous Environments"
[18] K. Yamada, R. Wang, M. Y. Sanadidi, and M. Gerla, "TCP Westwood with Agile Probing, Dealing with Dynamic, Large, Leaky Pipes", IEEE Communication Society, **(2004)**.
[19] D. Kliazovich, F. Granelli, and D. Miorandi, "Logarithmic window increase for TCP Westwood+ for improvement in high speed, long distance networks," Computer Networks, vol. 52, no. 12, **(2008)** August, pp. 2395–2410.
[20] H. Shimonishi, T. Hama, and T. Murase "TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm".
[21] K. Kaneko, T. Fujikawa, Z. Su, and J. Katto, "TCP-Fusion: a hybrid congestion control algorithm for high-speed networks," in Proc. PFLDnet, ISI, **(2007)** February, Los Angeles, California.

Authors

Ivan Petrov, is with Makedonski Telekom Skopje, Macedonia – mobile and fix network telecom operator member of Deutsche Telekom Group for almost 10 years, currently he holds position of Senior Sales Project and Provisioning Manager in Business and Vip Accounts Department. He received his Dipl.Ing. and M.Sc. from the Faculty of Electrical Engineering, Ss. Cyril and Methodius University in Skopje, in 2005 and 2009, respectively. From 2009 he is working towards his Ph.D. degree under mentorship of Prof. Toni Janevski. His research interests include transport protocols and cross-layer optimization techniques in heterogeneous wireless IP networks.



Toni Janevski Ph.D., is a Full Professor at the Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University, Skopje, Macedonia. He received his Dipl. Ing., M.Sc. and Ph.D. degrees in electrical engineering all from Faculty of Electrical Engineering and Information Technologies, Ss. Cyril and Methodius University in Skopje, in 1996, 1999 and 2001, respectively. During 1996–1999 he has worked for the Macedonian mobile operator Mobimak (currently T-Mobile, Macedonia), contributing to the planning, dimensioning and implementation of the first mobile network in Macedonia. From 1999 he is with Faculty of Electrical Engineering and Information Technologies in Skopje. In 2001 he has conducted research in optical communications at IBM T. J. Watson Research Center, New York. During 2005–2008 he was an elected member of the Commission of the Agency for

Electronic Communications (AEC) of the Republic of Macedonia. During the periods 2008–2012 and 2012–2016 he is an elected member of the Senate of the Ss. Cyril and Methodius University in Skopje. In 2009 he has established Macedonian International Telecommunication Union (ITU) Centre of Excellence (CoE) as part of the Europe's CoE network, and serves as its head/coordinator since than. He is the author of the book titled ''Traffic Analysis and Design of Wireless IP Networks'', which is published in 2003 by Artech House Inc, USA. Also, he is the author of the book ''Switching and Routing'', written in Macedonian language, published in September 2011 by the Ss. Cyril and Methodius University in Skopje. In 2012 he has won ''Goce Delchev'' award, the highest award for science in the Republic of Macedonia. He also received the best scientist award for 2013 from Ss. Cyril and Methodius University in Skopje. In 2014 his book ''NGN Architectures, Protocols and Services'' is published by publisher John Wiley & Sons, UK. He has published numerous research papers and led research and applicative projects in the area of Internet technologies and mobile and wireless networks. He is a Senior Member of IEEE. His research interests include Internet Technologies, Mobile, Wireless and Multimedia Networks and Services, Traffic Engineering, Quality of Service, Design and Modeling of Telecommunication Networks, as well as Next Generation Networks.