An Implementation and Experimental Study of the eXplicit Control Protocol (XCP) by Yongguang Zhang & Tom Henderson

Presented by

William Scott

October 06, 2009

Note: all material taken from **An Implementation and Experimental Study of the eXplicit Control Protocol (XCP) by** Yongguang Zhang & Tom Henderson

Table of Contents

What is This Study?

- **History:**
- Proposed by Katabi, Handley & Rohrs in SIGCOMM ’02
- Fleshed out in Katabi’s Thesis at MIT ‘03
- Analysis continues..

- **What is the study about?**
  - An experimental study on XCP and its deployment.
  - The authors have implemented XCP in Linux and conducted a comprehensive experimental evaluation.

- **What are the study findings:**
  - The experiments initial validation results match the previously reported simulation results.
  - Implementation challenge that arose from the lack of double-long or floating point arithmetic in Linux kernel.
  - How the choice of data type affects accuracy and the performance of XCP.
  - XCP's performance can be adversely affected by environment factors including:
    - *TCP/IP configuration*
    - *Link sharing*
    - *Non-congestion loss*
    - *The presence of non-XCP queues.*

  - Show several such possibilities for XCP to lose its ability to control congestions and achieve fairness

What is This System?

- **Possible Motivations:**
  - Simulation results have shown that XCP controllers are stable and robust to estimation errors, and require only a few per-packet arithmetic operations, because it relies on floating-point operations, it has not been clear how the simulation models might translate to implementation code.
  - It is not well understood how XCP might perform in a partially deployed environment.
  - Very few XCP implementations.
  - The FreeBSD-based kernel does not have the same limitations on double long division that they encountered in the Linux kernel, and the initial results presented do not consider operation in mixed deployment scenarios.
  - XCP is one of many proposals in the area of .high-speed TCP extensions proposed (Scalable TCP, HighSpeed TCP, FAST TCP, and others).
  - The main difference between XCP and the other high-speed TCPs is that XCP relies on explicit router feedback, while high speed TCPs are end-to-end and can only indirectly infer the congestion state of routers along the path.

Implementation Details

- **Architecture :**
  - Implemented XCP as a TCP option. Other possibilities: XCP as a separate transport protocol, XCP as a layer 3.5. header between the IP and transport headers, or XCP as an IP header option.
  - For talking with XCP, TCP is the underlying transport mechanism (connects and delivers flow data).
  - The effect of the XCP option is to modify TCP sender's cwnd (congestion window) value according to the XCP protocol.
  - Implementing XCP as a TCP option allowed them to leverage the existing protocol and software structure, resulting in a much faster development cycle, and backward compatibility to legacy TCP stacks.

  - If you implement XCP as a TCP option, are you really testing the XCP Protocol? Hypothetical, If you put a Ferrari dashboard in a Ford are you testing a Ferrari automobile?

Implementation Details

- **Architecture :**
  - **This architecture allows for:**
    - Whenever TCP sends a data segment, the XCP congestion header is encoded in a header option and attached to the outgoing TCP header.
    - The feedback field of the XCP option can be modified by routers along the path from sender to receiver, and the receiver returns the feedback in TCP ACK packets, also in the form of an XCP option in the TCP header.
    - Upon receiving an XCP option from an incoming ACK packet, the TCP sender updates its cwnd accordingly.

  - **The 2 part software architecture for the XCP includes:**
    1. A modification to the Linux TCP stack to support XCP end-point functions.
    2. A kernel module to support XCP congestion control in Linux based routers.
        1. A simple API is used so that any application can use XCP by opening an TCP connection and setting a special socket option.

Implementation Details

- XCP Option Format :
  - 2 formats:
    1. One on the forward direction that is part of the data segments from sender to receiver,
    2. On the return direction that is part of the ACK segments from receiver to sender.
  - Since routers should only process the forward-direction XCP option, having two option formats makes it easy for the routers to know which direction the XCP option is traveling.
  - It paves the way to support two-way data transfer in a TCP connection, which will have forward direction XCP options traveling in both directions (from both end-points).

Implementation Details

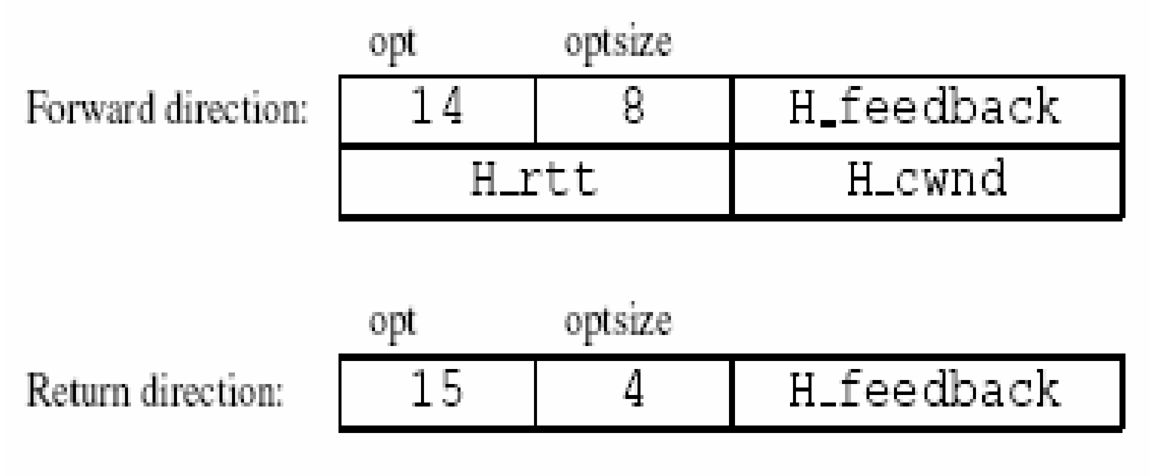


Fig. 1. XCP option formats in both directions

- Fig. 1 illustrates the 2 formats. They picked 2 unused values (14 or 15) for TCP option without making any attempt to go through IETF standardization. Other than opt and optsize, the remaining 3 are XCP congestion header fields which are each 16 bits long.

They are basically encoding XCP header as TCP header option & attaching it to outgoing TCP header.

Implementation Details

- H_cwnd stores the sender's current cwnd value, measured in packets (segments).
- H_feedback is measured in packets because it is the unit of change in TCP's cwnd in Linux kernel.
- Don't use a short integer type because that would limit the value to plus or minus 32,000 packets, which may not be sufficient for extremely large delay-bandwidth product.
- XCP feedback value cannot be rounded to an integer, a split mantissa-exponent representation was used that interprets the 16-bit H_feedback as the following:
  - The most significant 13 bits is the signed mantissa (m)
  - The remaining 3 bits is the unsigned exponent (e)
  - The value stored in H_feedback is m $*16^{(e-3)}$
- This format can represent a cwnd value from $\pm 2^{-12}$ (0.000244) to $\pm 2^{28}$ (268,435,456) and 0.
- H_rtt is measured in milliseconds. Given that it is 16-bit, the maximum round trip time supported by this XCP implementation is around 65 seconds, which should be suitable for most cases.

Implementation Details

- XCP End-point Functions
  - XCP Control Block and cwnd updates
    - XCP has a control block for each XCP connection endpoint to store XCP state information.
    - XCP is used as a TCP option, so the XCP control block is part of the TCP control block (struct tcp_opt). Some major variables:

```
int xcp;                    /* whether to use XCP option */
struct xcp_block {
  __u16 rtt;                /* RTT estimate for xcp purpose */
  __s16 feedback_req; /* cwnd feedback request */
  __s16 feedback_rcv; /* received from XCP pkt */
  __s16 feedback_rtn; /* returned by receiver */
  __s16 cwnd_frac;     /* cwnd fractions */
  __u32 force_cwnd;  /* restore cwnd after CC */
} xcp_block;
```

Implementation Details

3 feedback variables match the H_feedback field:

1. **feedback_req** is the requested amount that XCP sender will put in outgoing XCP packets
2. **feedback_rcv** is the feedback amount that XCP receiver receives before passing back to the sender.
3. **feedback_rtn** is the amount that XCP sender finally receives.

- To support delayed ACK, feedback from many packets can be stored at both feedback_rcv and feedback_rtn. The same mantissa exponent data type is used in these 3 variables.

- Steps added to TCP's usual ACK processing to handle XCP options:
  - Upon receiving a TCP packet with XCP option, it updates one of the above variables.
  - At sender, the *feedback_rtn* amount is added to TCP's cwnd (*snd_cwnd*).
  - Since *cwnd* grows or shrinks in integer units, the leftover fractional part is stored in *cwnd_frac* and will be added back to *feedback_rtn* next time.

Implementation Details

- Integrating with TCP congestion control
  - Chose to preserve the TCP congestion control code, but remove its effect on cwnd change, except in RTO case where cwnd is reset to 1.
  - How?
    - By using the *force_cwnd* variable in XCP control block.
    - After the sender updates its cwnd from XCP feedback, it stores the new cwnd value in force_cwnd.
    - After subsequent TCP ACK processing and before any retransmission event, the sender restores the cwnd value from *force_cwnd*.
  - This in effect allows fast retransmission but disallows slow start and congestion avoidance (linear increase and multiplicative decrease).
- Is preserving the TCP congestion control code a problem? Are they testing XCP still?

Implementation Details

- Increasing the advertised window
  - In TCP, the sender's cwnd is bounded by the receiver's advertised window size. Therefore, XCP may not be able to realize all of its cwnd value if the receiver's advertised window is not sufficiently large (so the receiver cannot get as much data).
  - Using the Linux implementation, the receiver grows the advertised window linearly by 2 packets per ACK.
  - This is OK for normal TCP, but will be insufficient for XCP as XCP feedback may open the sender's cwnd much faster than that.
  - So they modified the TCP receiver so the advertised window grows by the integer value of feedback_rcv.

This modification can have an adverse effect if XCP cannot utilize all of its cwnd window.

Implementation Details

- Reducing the maximum number of SACK blocks
  - This XCP implementation supports the selective acknowledgement (SACK) mechanism
  - Why?
    - Because it has been proven effective in dealing with losses in TCP, and many TCP implementation already include SACK option
    - One of there conclusions: it is important for XCP to have the SACK mechanism to deal with noncongestion loss.
  - In using XCP as a TCP option this was an easy task. They only reduced the maximum number of SACK blocks allowed in each TCP ACK packet.
  - Given that the maximum size of a TCP header (including options) is 60 bytes, with timestamp option and XCP option, they now have at most 2 SACK blocks, compared to maximum 3 SACK blocks before they added XCP.
  - They later show that, even with a reduced number of SACK blocks, it still provides significant performance improvement in wireless networks.

Implementation Details

- *XCP Router Module*
  - What the router does:
    - The router has to parse the congestion header in every XCP packet
    - Compute individual feedback
    - Update the congestion header if necessary.
  - XCP router function is divided into 2 parts:
    1. the XCP congestion control engine that acts on the information encoded in each congestion header.
    2. The kernel interface that retrieves such information from the passing XCP packets.

Implementation Details

- Kernel Support and Interface
- 2 Linux kernel mechanisms that allow insertion & interface of the XCP engine:
  - Loadable module
    - The entire XCP router function is implemented as a kernel loadable module with no change to the kernel source.
  - Device-independent queueing discipline layer.
    - The Linux network device layer includes a generic packet queueing and scheduling mechanism called Qdisc
    - Its basic functions are to enqueue an outgoing packet whenever the network layer passes one down to the device layer, and to dequeue whenever the device is ready to transmit next packet.

- The XCP router module has 2 functions Qdisc calls to invoke the XCP engine (see Fig. 2):
  - xcp_do_arrival() is called when an XCP packet is enqueued.
  - xcp_do_departure() is called when the XCP packet is ready to be transmitted in hardware.
  - A new built-in Qdisc called .xcp. is included to operate XCP with a standard drop-tail queue (fifo).

Implementation Details

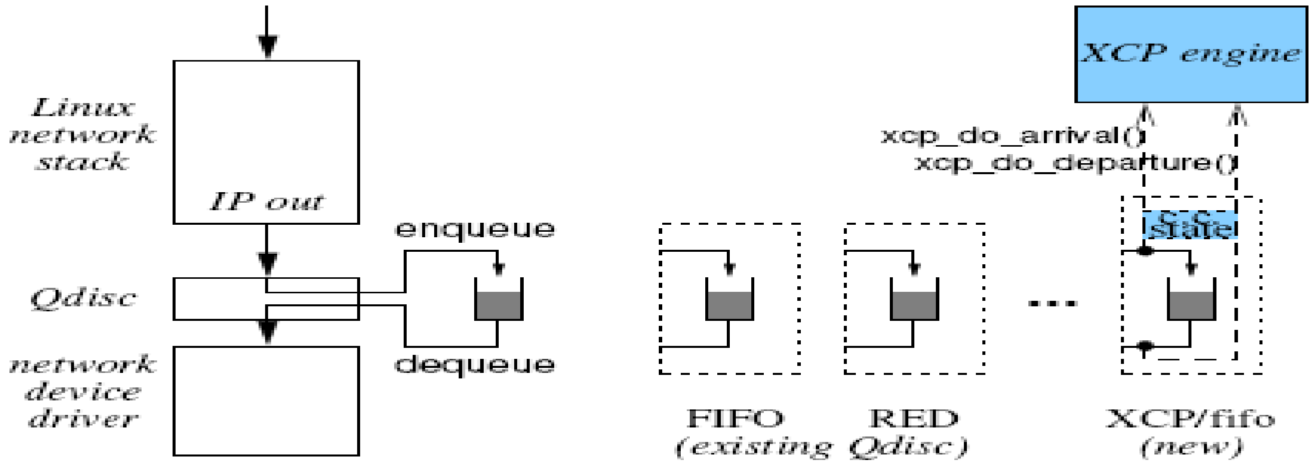


Fig. 2. Structure of XCP router module w.r.t. networking stack in Linux kernel

- Managing the XCP Qdisc,:
  - Used a loadable module for the Linux traffic control command tc.
  - This command makes using XCP Qdisc easy (network device or queueing system).
  - For example, the following command enables XCP for a 10Mbps link:
    - tc qdisc add dev eth2 root xcp capacity 10Mbit

- The configuration parameter capacity indicates the hardware-specified raw bandwidth. Adjustments were made to count for the framing overhead when XCP router calculates the true output link capacity.

Implementation Details

- Per-link State Information
  - XCP routers never maintain any per-flow state, and the XCP engine is completely stateless.
- Instead, the per-link congestion control state information is stored at each XCP-enabled Qdisc (.c.c. state. in Fig. 2).
- When the Qdisc invokes the XCP engine, it passes the packet (in skbuff structure) along with the c.c. state. structure.

Implementation Details

- Per-Packet and Per-Interval Computation
  - After implementing the Katabi et al algorithms, the XCP router uses 2 per-packet processing functions:
    - xcp_do_arrival() updates the running traffic statistics at packet arrival
    - xcp_do_departure() calculates the individual feedback when the packet departs the queue.
    - Also, the XCP router must do per-interval processing to calculate the aggregate feedback, feedback distribution factors, and the length of next interval. This is done at the end of a control interval but it requires a variable interval kernel timer.

  - Adopted a delayed computation approach to avoid managing a high resolution timer in Linux kernel.
    - The end-of-interval computations take place when either of the 2 per-packet processing functions is called next.
    - Advantage: don't maintain kernel timers & XCP engine is packet-driven.

Implementation Details

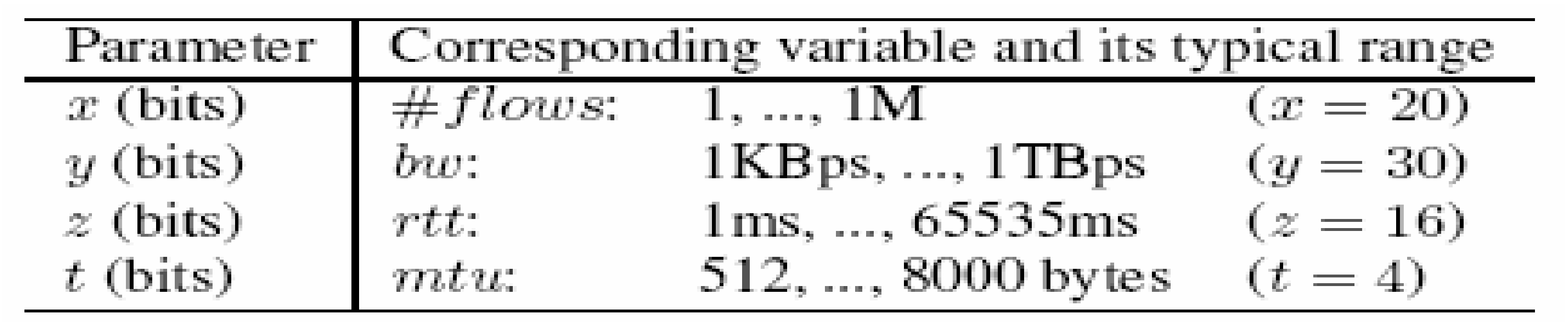
- Integer Arithmetic and Scalability Issue
  - The complex computation in XCP engine involves several real number multiplications and divisions, as well as summations over many packets.
  - However, many kernels support limited integer arithmetic only. For example, 32-bit Linux has no double long arithmetic or floating point operations in kernel mode.
  - They must therefore carefully design the data types to fit XCP feedback calculations in 32-bit integer space.
  - Unfortunately this may limit the scalability, precision, and granularity in XCP control.

  - You would have to rewrite the kernal to accommodate this,
  - You might as well rewrite the operating system
  -

Implementation Details

Scaling Analysis

- To do the calculations; first determine the unit, or scale, for each variable in the XCP computation.
  - Scale is the mean of the range of values that a variable can represent as a function of the size of the variable (in terms of bits). That is, if a variable is sizes, the range of values that it can represent after scaling is [c1 + c2, (2^s - 1) * c1 + c2], where c1 and c2 are the scaling factors . c1 for granularity and c2 for offset.
  - For example, to represent a range from 1ms to 65sec round trip delay with 1ms granularity, you need a 16-bit variable and here c1 is 1ms and c2 is zero.
- Start inputting the scales of XCP operation environment parameters. the number of flows (#flow), bandwidth (bw), round trip time (rtt), MTU (mtu).
- The following table lists these parameters and gives each a reasonable range that they think an XCP implementation should support.

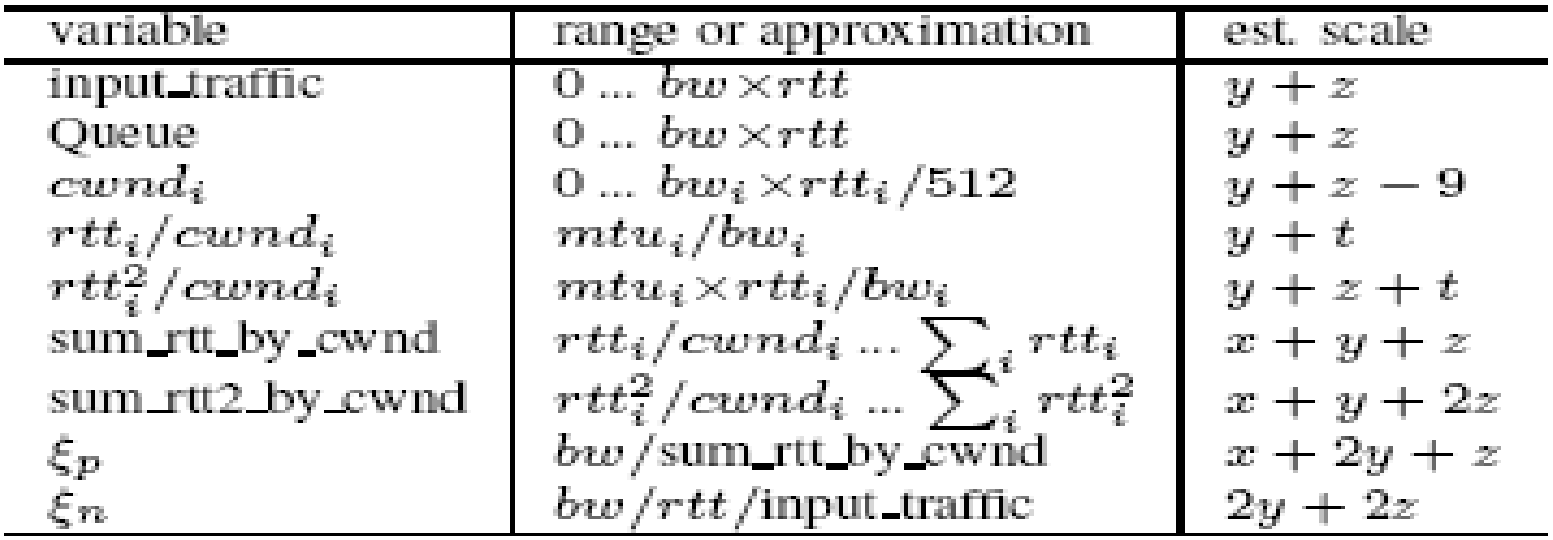


Parameter	Corresponding variable and its typical range		
x (bits)	$\#flows$:	1, ..., 1M	$(x = 20)$
y (bits)	bw:	1KBps, ..., 1TBps	$(y = 30)$
z (bits)	rtt:	1ms, ..., 65535ms	$(z = 16)$
t (bits)	mtu:	512, ..., 8000 bytes	$(t = 4)$

Implementation Details

Scaling Analysis

- They then estimate the scales for all other variables used in XCP calculations relative to these environmental parameters.
- They take into account how they are calculated, their ranges, and granularities.
- For example, the size for cwnd_i (cwnd in flow i) should be y + z - 9, because its value converges to bw_i x $\text{rtt}_i/\text{mtu}_i$,which can range from 0 to 1TBpsx65s/512 with the granularity of 1 (packet).
- The following table lists such estimation for other XCP variables.



variable	range or approximation	est. scale
input_traffic	0 ... $bw \times rtt$	$y + z$
Queue	0 ... $bw \times rtt$	$y + z$
$cwnd_i$	0 ... $bw_i \times rtt_i / 512$	$y + z - 9$
$rtt_i / cwnd_i$	mtu_i / bw_i	$y + t$
$rtt_i^2 / cwnd_i$	$mtu_i \times rtt_i / bw_i$	$y + z + t$
sum_rtt_by_cwnd	$rtt_i / cwnd_i$ ... $\sum_i rtt_i$	$x + y + z$
sum_rtt2_by_cwnd	$rtt_i^2 / cwnd_i$ ... $\sum_i rtt_i^2$	$x + y + 2z$
ξ_p	$bw /$ sum_rtt_by_cwnd	$x + 2y + z$
ξ_n	$bw / rtt /$ input_traffic	$2y + 2z$

Implementation Details

Scaling Analysis

- Based on this analysis, they are able to choose the scale and hence the data type for each variable based on the range of networking environment hoped to support.
- For example, if you are to support the range given before the previous paragraph, you will need triple-long (96-bit) integer arithmetic. Of course, this assumes the extreme scenario of 1 million flows passing through the same router, some having 1ms RTT and 1TBps bandwidth while some others having 64s RTT and only 1KBps bandwidth.
- If you give up range or the precision (granularity), you can use a shorter integer (64-bit).
- Since the 32-bit Linux kernel doesn’t have native support for double-long operations, you have to use a special data type called shiftint to accommodate the wide range of scales.
- It consists of a 32-bit integer to store the most significant bits, a short integer to store the scaling factor, and another short integer to store the shifting factor.
- Integers of any scale can be shifted and stored in a variable of this data type.
- Much of the XCP algorithm is implemented as operations on this data type.
- The tradeoff is that you will lose precision in some calculations.
- A simulation study that compared the shiftint results with floating point operations puts the precision at 0:001%.
- The advantage is that you don't need to worry about manually scaling each variable as the scaling factor component automatically keeps track of the change.

Implementation Details

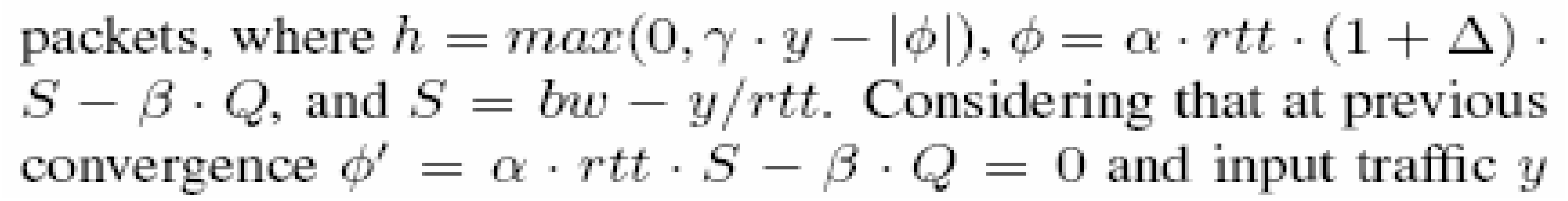
Feedback Fractions

- In Linux and other TCP stacks, the unit of change in cwnd is a whole packet.
- So, you would have easily used an integer type for H_feedback, but the following analysis contradicts this intuition and shows that if H_feedback is measured in packets you must keep the fractional part and avoid rounding.
- Consider a single flow with round trip delay rtt and bandwidth bw.
- Under XCP, its cwnd would converge at cwnd =rtt x bw/mtu packets i.e., during one XCP control interval, the router will encounter cwnd packets from this flow.
- Now, let's assume that the available bandwidth for this flow has changed from bw to (1 +Δ) bw.
- All things being equal & unchanged, the individual feedback according to the XCP algorithm is

$$H\_feedback = \frac{h + max(\phi, 0)}{rtt \cdot \sum \frac{rtt \cdot s}{cwnd}} \cdot \frac{rtt^2}{cwnd} - \frac{h + max(-\phi, 0)}{rtt \cdot y} \cdot rtt$$

Implementation Details

Feedback Fractions continued



packets, where $h = max(0, \gamma \cdot y - |\phi|)$, $\phi = \alpha \cdot rtt \cdot (1 + \Delta) \cdot S - \beta \cdot Q$, and $S = bw - y/rtt$. Considering that at previous convergence $\phi' = \alpha \cdot rtt \cdot S - \beta \cdot Q = 0$ and input traffic y

should equal cwnd *mtu, you can deduce that H_feedback is ά * ∆ (packets).

- If you look at this intuitively, when the bandwidth changes by a factor of ∆, the sender should match with a cwnd change by a factor of ά * ∆ (where ά = 0:4 is the stability constant).
- This amount will be divided into small individual feedback shares among all packets during a control interval (also a round trip time).
- So if you only use integers to represent H_feedback and if the individual feedback share is small (| ά * ∆ | < 0:5), you will lose all the individual feedbacks to rounding.
- And you will also lose the accumulative feedback since the sender can only accumulate cwnd changes by adding individual feedbacks together.

Implementation Details

Feedback Fractions continued

- The result (Fig. 3) shows that XCP sustains higher performance than with the alternative implementation.

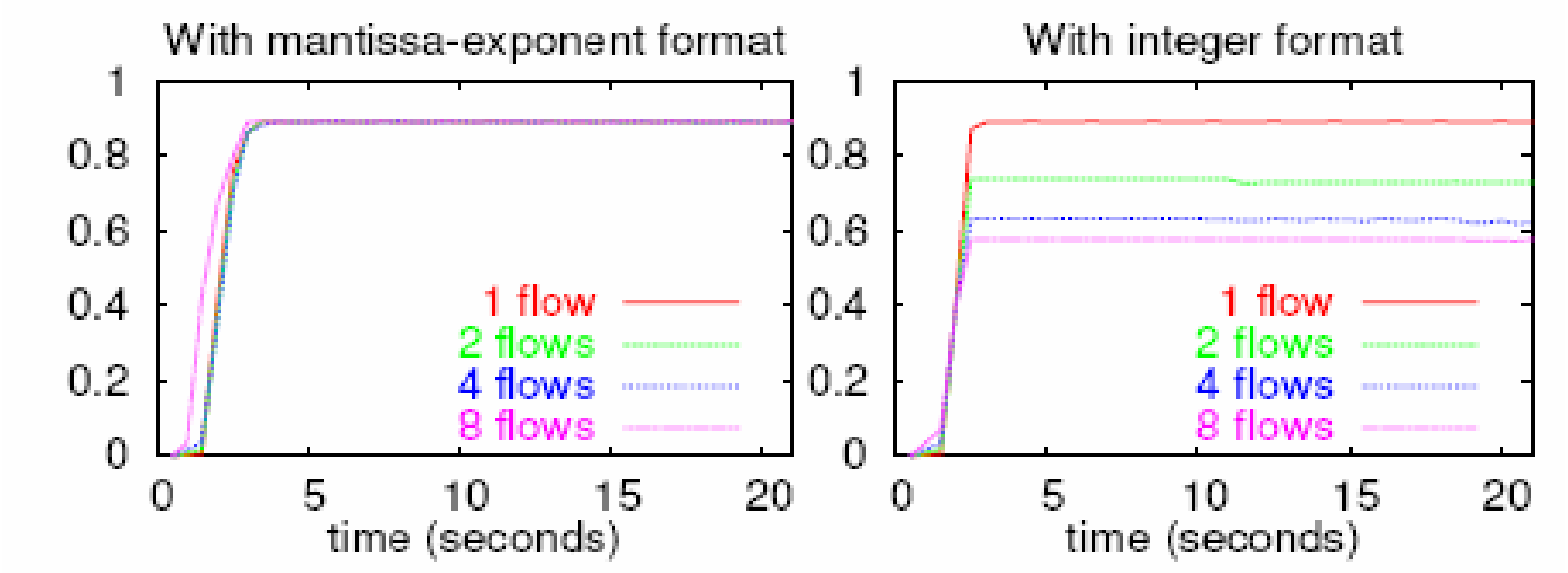


Fig. 3. Bandwidth utilization (total goodput of all flows as a ratio of raw bandwidth) comparison when 1, 2, 4, or 8 flows sharing the same bottleneck.

Experimental Evaluation

- **Experimental Setup**
- The experimental study has been conducted in a real test network illustrated in Fig. 4. XCP end-system code runs at end hosts S1...S4 and D. XCP router code runs at router R.
- The end-to-end latency is emulated by placing a dummynet host between R & D.
- The bottleneck link is between R and D, whereas the bandwidth between sources S and R is higher, and the transmission queue will build up at R's network interface to D. XCP router code will operate on that interface to regulate flow throughput.

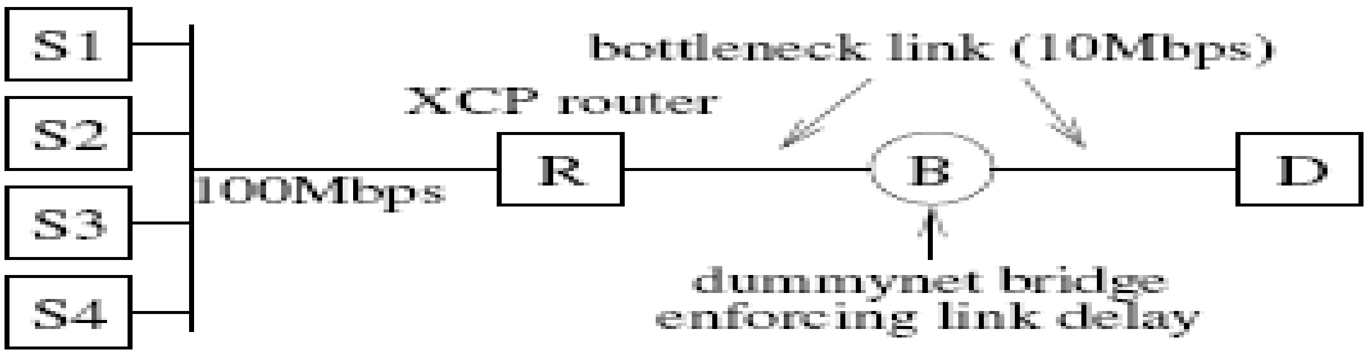


Fig. 4. Experimental network configuration

Experimental Evaluation

- **Experimental Setup**
- The round trip delay is set at 500ms and the bottleneck link is 10Mbps full-duplex, unless specified otherwise in some particular experiments.
- Flows start at sources S1...S4 and end at D and they follow the TCP performance tuning guides to set optimal values for TCP parameters.
- Large buffer sizes are used and set the TCP window scaling option so that the connection can sustain the throughput in this large bandwidth-delay-product network.
- The maximal router queue size is set to twice that product, as is a common assumption for Internet networking, and In each experiment the flow performance is measured and compared.
- The following performance data was gathered from the traces collected in the experiments:
  - *Bandwidth utilization* - the ratio (between 0 and 1) of total flow goodput to the raw bottleneck bandwidth, measured every RTT as time progresses.
  - *Per-flow utilization* - the ratio of each flow's goodput to the raw bottleneck bandwidth.
  - *cwnd value* - the progress of XCP sender's cwnd size (in packets), sampled by recording XCP packet's H_cwnd field.
  - *Router queue length* - the standing queue size at bottleneck link, sampled whenever a packet enters the queue at router R.
  - *Packet drops* -the number of queue drops at bottleneck link, recorded every RTT at router R.

Experimental Evaluation

- **Validation Experiment**
- The purpose of this set of experiments is to validate the XCP implementation and to validate the previously published simulation results on the XCP protocol.
- The Katabi XCP paper shows extensive packet-level simulations and how XCP achieves fair bandwidth allocation, high utilization, small standing queue size, and near-zero packet drops.
- The authors' reach a similar conclusion in real network controlled experiments.
- The first experiment compares XCP performance with TCP performance under FIFO and RED queuing disciplines.
- They start four flows at the same time, one from each source S, to D.
- For the XCP test case, the flows are XCP flows (TCP with XCP options); otherwise they are normal TCP flows.
- For RED test case, router R is configured with RED queue management with the drop probability 0.1 and min & max thresholds set to 1/3 and 2/3 the buffer size.
- Fig. 5 plots the performance measurement results.
- XCP has the best and most stable performance, in terms of better bandwidth utilization, smaller queue buildups, and zero packet drops.

Experimental Evaluation

- **Validation Experiment**

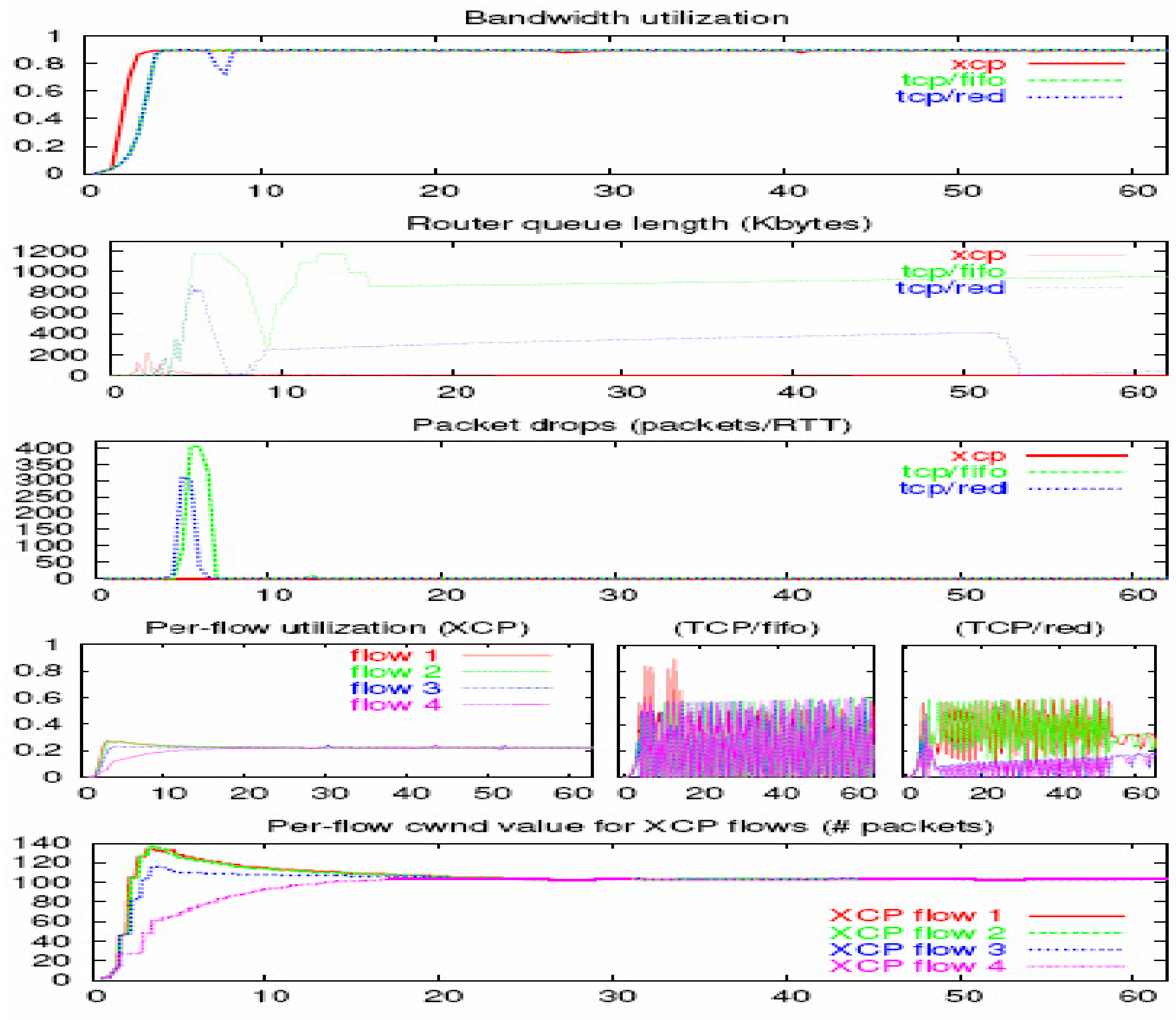


Fig. 5. Performance comparison between XCP and TCP.

Experimental Evaluation

- **Validation Experiment**
- The per-flow utilization charts further illustrate that XCP flows share the bandwidth equally and stably, but both TCP cases experience significant fluctuations among flows.
- The per-flow cwnd chart also reveals that XCP flows can quickly converge to an optimal cwnd value.
- They further study XCP fairness toward flows that do not start at the same time or have different RTTs but that share the bottleneck link . a well-known limitation of TCP congestion avoidance.
- In the next experiment, the setting is same except that the first flow starts at time 0 with each additional flow starting 30 seconds later and each flow lasts approximately 130 seconds.
- In the third experiment, They modify the dummynet setting so that the delay is 50ms between S1 and D, 100ms between S2 and D, 250ms between S3 and D, and 500ms between S4 and D.
- That is, They change the four flow's RTTs to 100ms, 200ms, 500ms, and 1000ms, respectively.
- Fig. 6 shows that XCP exhibit fairness in both experiments.

- Summary: under controlled settings, they demonstrate very good performance with XCP.

Experimental Evaluation

- **Validation Experiment**

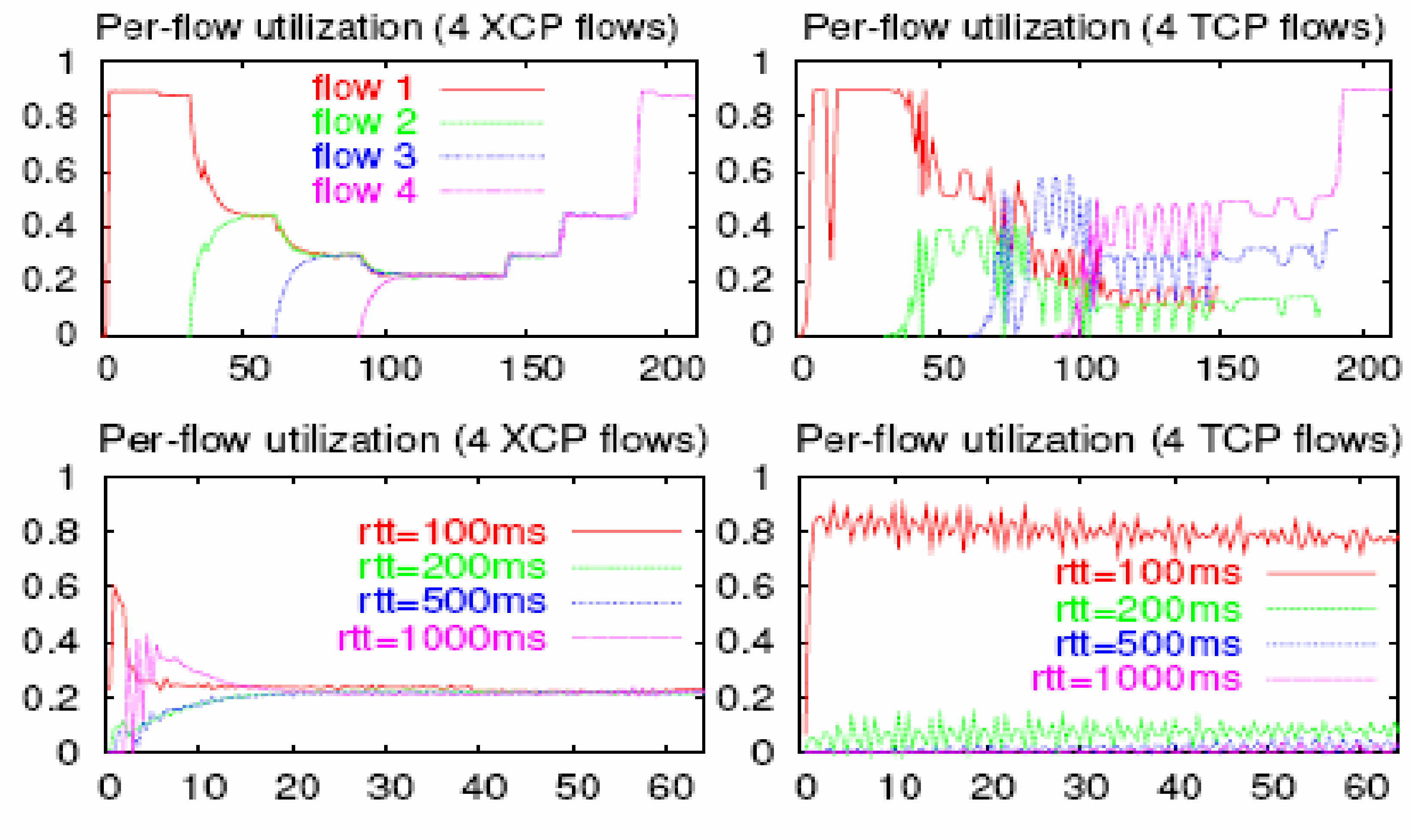


Fig. 6. Fair shares among 4 XCP flows with different start time or different RTT.

Experimental Evaluation

- **Fine-tuning the Link Capacity**
- To accurately calculate feedbacks, XCP router must know the precise link capacity in advance. It must factor in proper framing and MAC overhead in the raw bandwidth given in tc command.
- This is important because if you overestimate the overhead, you will under-utilize the link and lower the performance.
- Likewise, underestimating it, will over-promise capacity, inflate feedbacks, and drive up the queue.
- Unfortunately, this overhead cannot be easily predicted because it varies by datalink format and sizes of actual data packets.
- An empirical approach is used to estimate this overhead.
- In the same validation experiment setup, the number of bytes is varied to add as per-packet overhead estimate and the packet sizes (through MTU setting).
- They then compare the link utilizations and queue lengths.
- The result (Fig. 7) shows that 90-bytes is a good estimate to balance both performance metrics optimal bandwidth utilization and minimal queue buildup.

Experimental Evaluation

- **Fine-tuning the Link Capacity**.

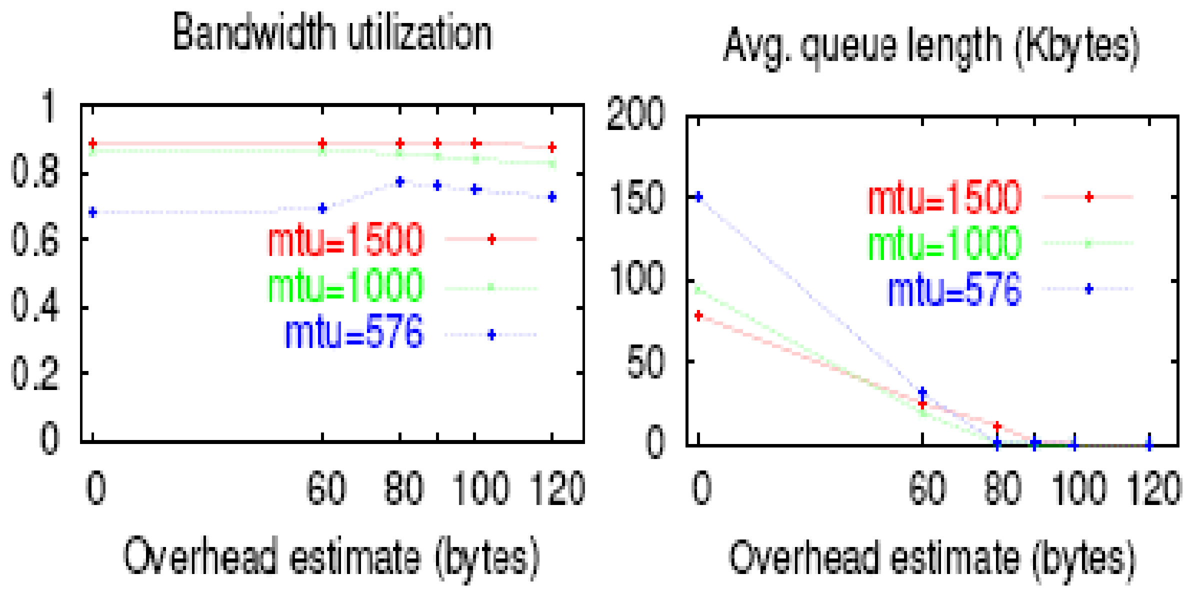


Fig. 7. Per-packet overhead estimates

- This estimation in XCP router's link capacity was included in the calculation.
- Note that this value is from experiments using 10Mbps full-duplex Ethernet.
- The number may be different for other types of links but the same approach can be followed to arrive at the best estimation.

Sensitivity Study

- In this section, we look in more detail at other protocol sensitivity issues under the following four operational conditions:
  - *TCP/IP parameter configuration-* Linux and many other operating systems provide mechanisms for user to tune TCP/IP stack parameters, such as kernel buffer size, receiver's kernel buffer size, sender's socket buffer size, and receiver's socket buffer size. These memory allocations can affect XCP's performance because the throughput is limited not only by cwnd value but also by the buffer space available at both sender and receiver.
  - *Link sharing* - Not all links in the Internet are point-to-point or have deterministic amounts of bandwidth. There are many contention-based multi-access links such as Ethernet and IEEE 802.11, and the true link capacity may be difficult or impossible to obtain. This may affect XCP feedback calculations.
  - *Wireless networks*- Wireless networks almost always have the same link-sharing problem because the media is shared multi-access and interference-prone. In addition, wireless networks often have frequent non-congestion losses due to imperfect channel condition, interference, mobility, terrain effects,etc. Wireless networks can also have temporary blockage or blackout period when the link experiences 100% loss for a short period of time.
  - *Hybrid networks-* Where not all queues are XCP-capable. XCP is designed assuming an all-XCP network, but it will have to co-exist with non-XCP queues if it is to be incrementally deployed in the Internet. Further, many link-layer devices (such as switches) have embedded queues for better traffic control and XCP will need to handle this type of hybrid networks as well.

Sensitivity Study

- **Sensitivity to TCP/IP Parameter Configuration**
- By XCP's control law, the XCP sender informs XCP routers of its current cwnd value and gets feedback for the new value to use during the next interval.
- They call the value that XCP sender puts in the H_cwnd field the *advertised* cwnd value.
- For the control law to work properly, XCP routers must expect the sender to send the advertised amount during a RTT.
- However, if other factors limit XCP's sending rate, such as memory buffer shortage at the sender or receiver, the control law can be broken and XCP may not converge.
- To prove this point, they repeat the above validation experiment with a single XCP flow.
- They first use the system default buffer size (64K) at the receiver and then repeat with a larger value matching the bandwidth-delay-product (640K). Fig. 8 compares the bandwidth utilization of these 2 flows.
- Due to buffer limitation, the flow with small buffer size cannot fully utilize its cwnd to fully utilize the bandwidth.
- If the sender fails to send as much as advertised, XCP routers will see the difference as spare bandwidth.
- Since XCP does not keep per-flow state to monitor the sending rate, when a router sees spare bandwidth, it will use positive cwnd feedback to increase a sender's allocation.
- This goes into a loop such that the XCP sender can keep increasing its cwnd value monotonically, well beyond the convergence point.

Sensitivity Study

- **Sensitivity to TCP/IP Parameter Configuration**

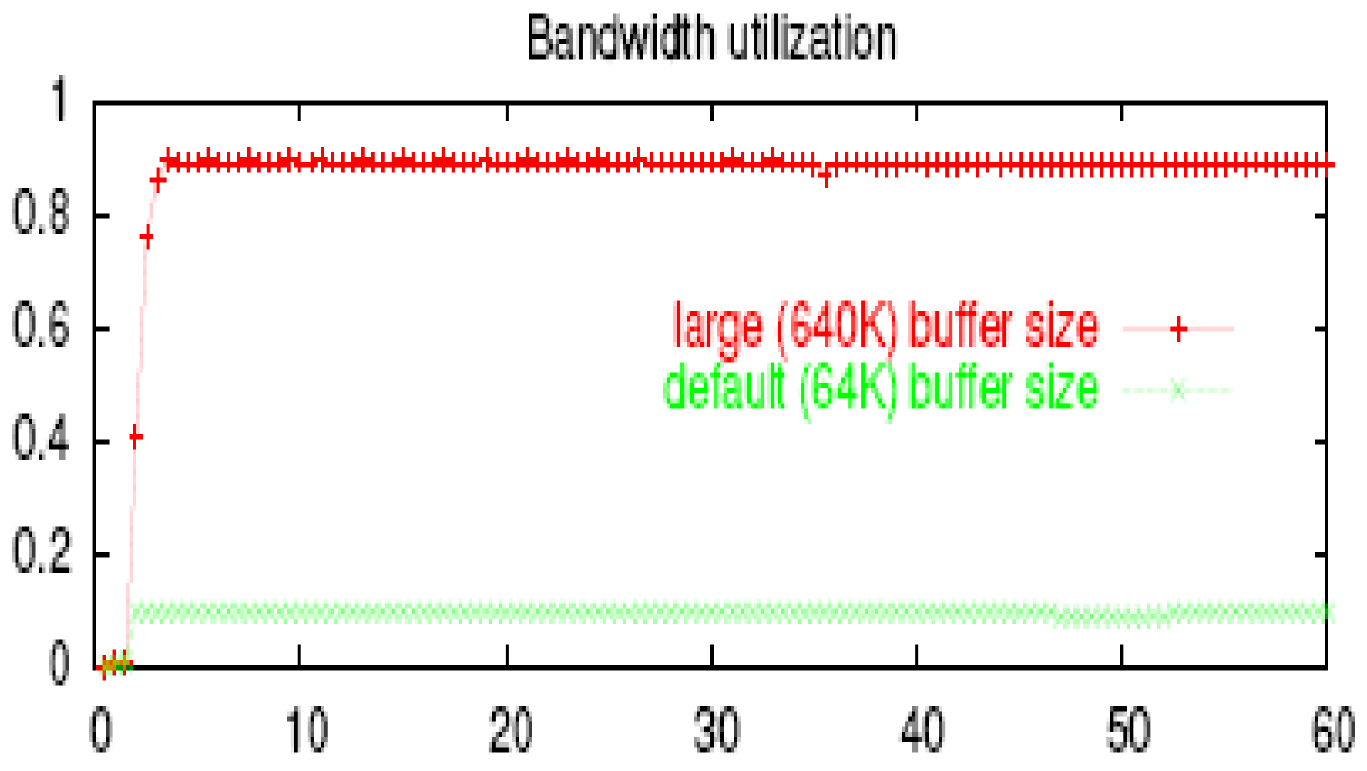


Fig. 8. XCP bandwidth utilization under different parameter tuning.

Sensitivity Study

- **Sensitivity to TCP/IP Parameter Configuration**
- Fig. 9 shows such a negative effect. When large buffer sizes are used, the cwnd value has converged quickly to the optimal value around 400 packets.
- But when the default buffer size is used the cwnd value increases linearly above 10,000 packets.
- Although the actual sending rate is still small due to buffer limitation, this limit may be removed when a transient memory buffer shortage is eased later.
- Now the congestion may become severe because the sender can suddenly send more than the network can handle under the inflated cwnd value.

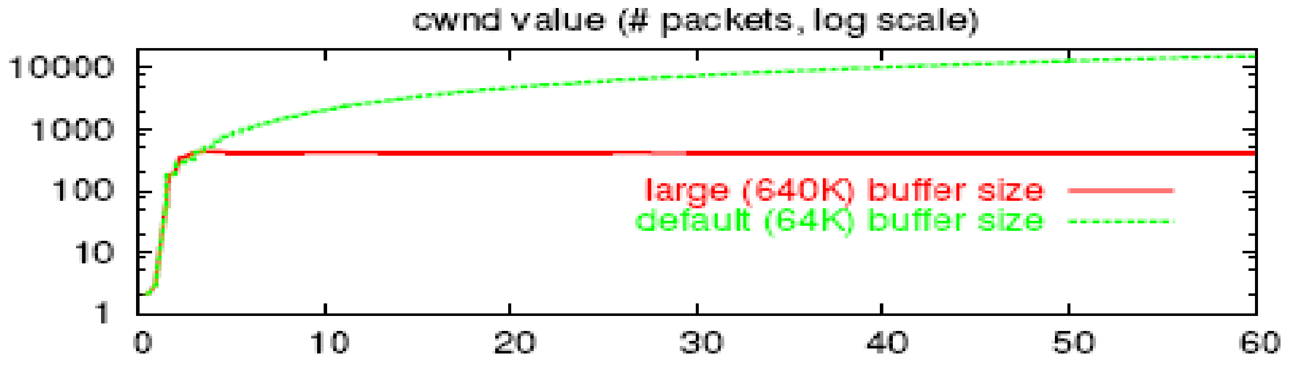


Fig. 9. XCP flow cwnd convergence under different parameter tuning.

- To avoid this problem, the XCP sender advertises the true cwnd limit instead of the cwnd face value e.g., if the additional limiting factor is the receiver's advertised window size, an XCP sender can put the lowest of either these limits or the current cwnd value as the advertised cwnd in H_cwnd field.
- The sender can put the lowest limit in H_feedback field to set an upper bound on the positive feedback.

Sensitivity Study

- **Sensitivity to Link Contention**
- The previous results are based on a point-to-point full-duplex Ethernet link (cross-over cable) at the bottleneck.
- However, if XCP operates in a multiple-access shared network, there will be cross traffic and media-access contention.
- But, there is no easy way to predict and plan for such contention in calculating the true output capacity.
- XCP can only take link capacity at its face value.
- The damage will be self inflicted: XCP will generate inflated feedbacks, the senders will send more than the link can transfer, and queue will build up.
- To show this, the previous validation experiment is repeated with a 10Mbps Ethernet hub for the bottleneck link.
- First, set the hub to be half-duplex, in which case the XCP data packets will have to compete with the ACKs for the link access.
- Then, add another host to the same hub and dump an additional 4Mbps UDP traffic onto the link, creating media-access contention.

Sensitivity Study

- **Sensitivity to Link Contention**
- Results (Figure 10) clearly fault the XCP algorithm when the link is half-duplex or is not contention-free.
- Understandably the utilization is reduced because the link capacity is reduced, but since XCP does not know this it over-estimates the spare bandwidth and inflate the feedbacks.
- That is why the results show inflated cwnd values and significant queue buildups (compared to nearly zero queue length).
- Looking at per-flow utilization, you can see that individual XCP flows all have trouble converging (compared with Fig 5).
- But, there nothing to do at the network layer to remedy this deficiency, unless you add MAC delays or conservative assumptions into the XCP control law.

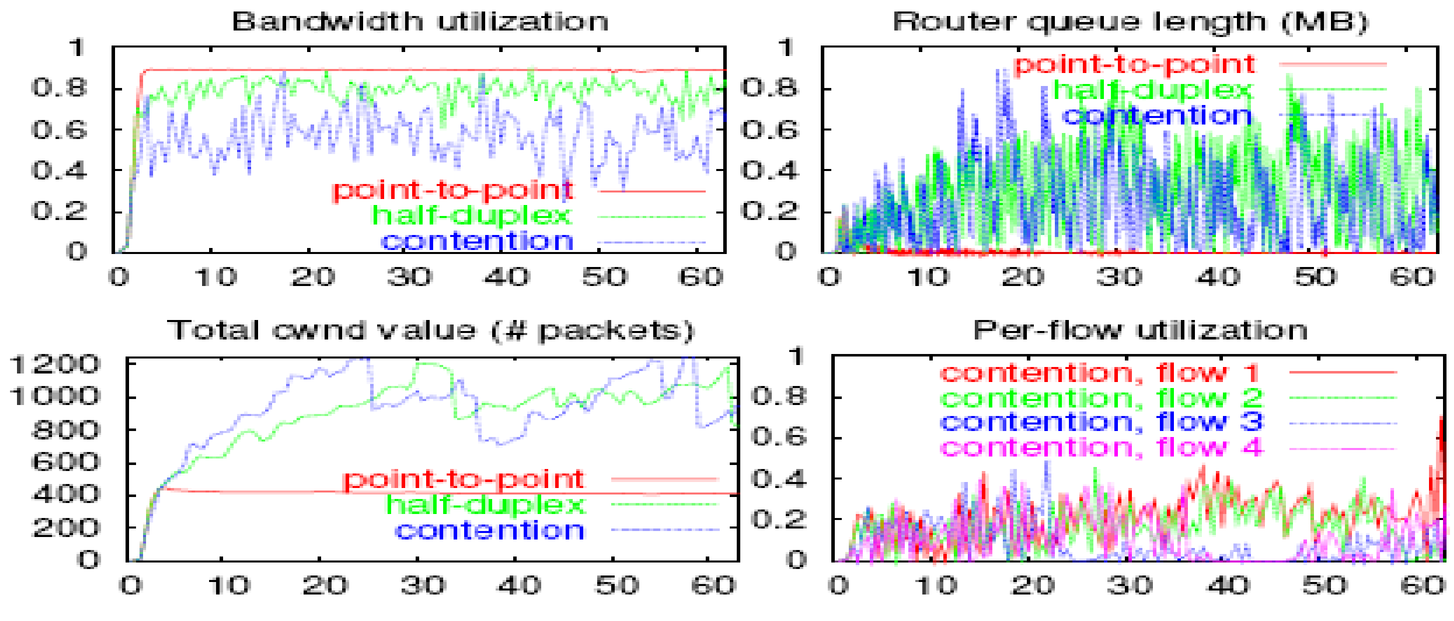


Fig. 10. XCP performances with different link environment.

Sensitivity Study

- **Sensitivity to Non-congestion Loss**
- To study the effect of non-congestion losses on XCP, experiments were done to inject losses artificially & randomly at a fixed probability, to emulate a lossy wireless channel.
- Packet losses were injected at 1 of the 3 locations: in the forward direction between the XCP sender and the XCP router, in the forward direction between the XCP router and the XCP receiver, and on the return direction.
- Since the XCP router does not process return-direction XCP options, it is unnecessary to make a distinction as to where to drop the return-direction XCP packets.
- They label these 3 different loss sources as .pre., .post., and .ack., and they varied them in different experiments to understand how to cope with different loss sources.
- They also varied the loss probability (packet loss ratio), in different experiments.
- They chose 10 different levels of loss ratios: 0.0001, 0.00025, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.025,0.05, and 0.1.
- They cover a wide range of loss ratios typically seen in a wireless network.
- In the experiments they let XCP converge first, and then started the loss period after 15 seconds.
- The loss period was 60 seconds, and the bandwidth utilization was measured at this time.
- In addition to varying the loss source at .pre., .post., and .ack., all the experiments were repeated with SACK option turned off and included a TCP case (with SACK) as a comparison.

Sensitivity Study

- **Sensitivity to Non-congestion Loss**
- First note: XCP handles noncongestion losses better than TCP in all cases, if XCP makes the assumption that observed losses are not due to congestion.
- Prior studies show packet losses on a high bandwidth-delay product path can substantially degrade TCP throughput because TCP cannot distinguish it from congestion loss.
- XCP: assumes all losses as non-congestion losses because it handles congestion separately through feedback.
- The next observation is that the loss of return-direction XCP packets (ACKs) does not have a noticeable impact in the whole range of loss rates, either with or without the SACK option.
- This is because ACKs are accumulative so a loss of an ACK can be recovered by the subsequent ACK and when XCP converges, the feedback carried in each ACK and the total feedback in a RTT is diminishing, losing a small number of feedback packets (say, 10%) will not affect the throughput by much.
- But, if forward-direction XCP packets are frequently lost, the impact on XCP performance can be significant because the lost segments must be retransmitted.
- However, it makes a big difference whether the SACK option is used or not.
- Without SACK, XCP's performance will suffer even with infrequent losses at a ratio as small as 0.001.
- With SACK, XCP will not have noticeable degradation until the loss ratio is more than 25 times higher, at more than 0.025.
- Even at that high loss ratio, the degradation is much smaller with SACK than without SACK.
- This result validates the assertion earlier that it is very important for XCP to include SACK to deal with non-congestion loss.

Sensitivity Study

- **Sensitivity to Blockage**
- Wireless networks can often have a temporary blockage or blackout period when the link experiences 100% packet loss for a short period of time.
- This can be caused by mobility when 2 nodes move out of range from each other, or by environmental effects like terrain or moving objects.
- It is obvious that no transport mechanism can communicate during the blockage period, but it is important to see how it can recover quickly to the previous state when the blockage ends.
- In this experiment, they emulated blockage by setting firewall rules at the router to drop all packets during the blackout period.
- The blackout period started at 15 seconds after the flows start so as to let XCP first converge.
- They measured the bandwidth utilization and observed its behavior before, during, and after the blackout period repeated the measurement using 1, 2 or 4 XCP flows.
- The result of a 2-second blackout period is illustrated in Fig. 12. They also included 1, 2, and 4 TCP flows as a comparison.

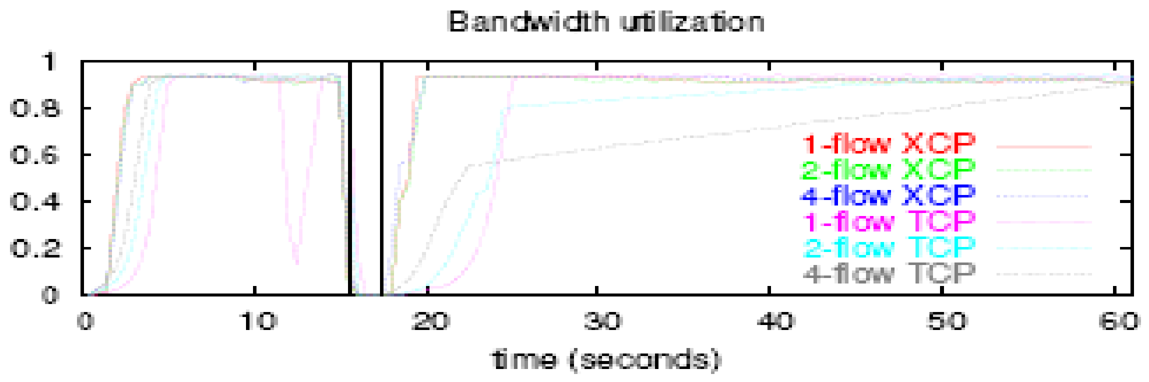


Fig. 12. XCP behavior around 2-second blockage (15th-17th second)

Sensitivity Study

- **Sensitivity to Blockage**
- Results show:
  - XCP can get out of a blockage quick, as fast as when the flow started.
  - TCP: much slower as it must go through slow-start.
  - In XCP's case, the number of flows does not make much of a difference.
- The blockage problem is further explored by varying the duration of blackout period, including 10, 20, 50, and 100 seconds.
- Fig. 13 shows a gap between the end of a blackout period and the start of XCP's climb when the blackout period is longer.
- A packet trace investigation showed this effect is caused by XCP's use on top of TCP.
- XCP has no policy dealing with blockage, it inherits TCP's exponential back-off of its retransmit timer.

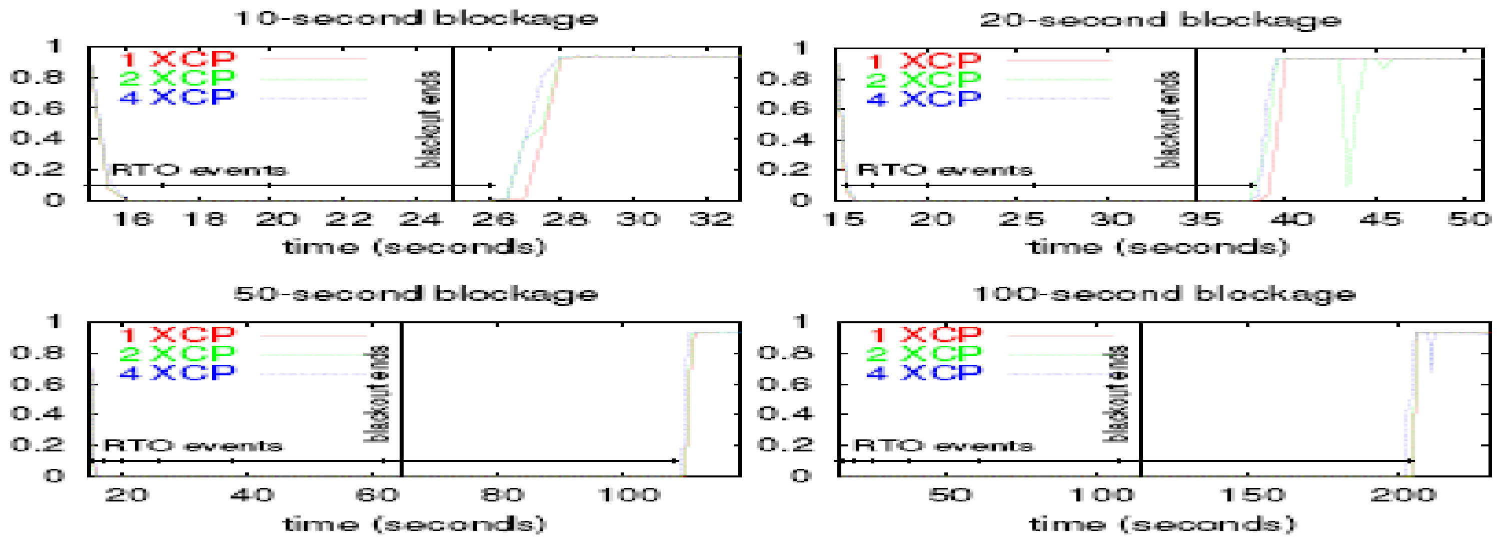


Fig. 13. XCP bandwidth utilization during and after blockage of various length

Sensitivity Study

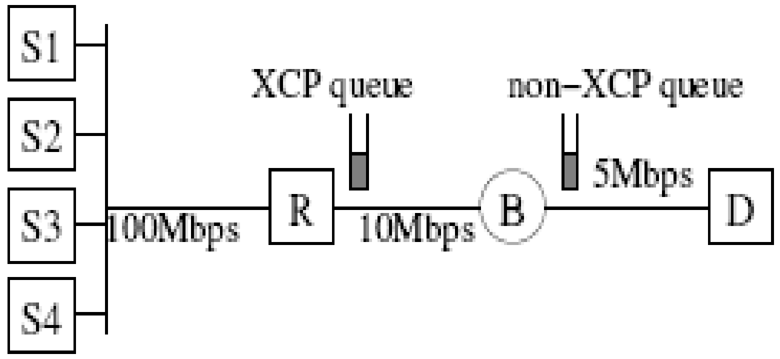
- **Sensitivity to Blockage**
- TCP: when the blockage is much longer than round-trip time, a retransmission timeout (RTO) event will happen.
- Then the first unacknowledged packet will be retransmitted and the RTO value will double.
- Fig. 13 has a plot of the RTO expiration events along the bottom of each chart.
- Notice, XCP will not recover until an RTO expires after the blackout period.
- The longer the blockage period lasts, the larger the RTO value will be, and the longer XCP will wait until it recovers.
- Unless notified by the network routers, XCP can only rely on retransmissions to learn when the blockage ends.
- A way to improve the inefficiency due to RTO growth is to replace the exponential back-off with a fixed RTO strategy.
- XCP handles congestion loss extremely well, so consecutive RTO implies route failure or blockage.
- If RTO does not double, the blockage ends, retransmission happens and XCP can recover.

Sensitivity Study

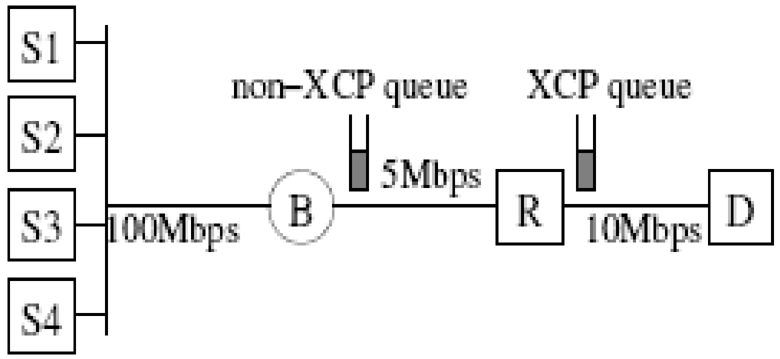
- **Sensitivity to Non-XCP Queues**
- The authors' think that XCP will perform poorly in a hybrid network, worse than TCP, if the bottleneck is a non-XCP queue.
- Since the XCP router with the lowest per-flow link capacity in the path will dictate the sender's cwnd, if this capacity is still higher than the actual bottleneck link, the cwnd may still be high enough to cause congestion and packet losses.
- XCP flows can only take commands from XCP feedbacks; it has no mechanism to react to this congestion while TCP reacts to packet losses by reducing cwnd
- To verify this belief, 2 experiments were conducted:
  - one put a tighter non-XCP queue after the XCP router
  - the other put it before (Fig. 14).
- In both cases, the non-XCP queue is a fifo with a 100-packet queue limit and its output is limited to 5Mbps, half of the XCP link capacity at router R. Rest of the setup remains the same as the previous experiments.

Sensitivity Study

- **Sensitivity to Non-XCP Queues**



Case 1: non–XCP queue after an XCP queue



Case 2: non–XCP queue before an XCP queue

Fig. 14. Network configuration for hybrid network experiments

Sensitivity Study

- **Sensitivity to Non-XCP Queues**
- Utilization and packet drops were measured as before but on the non-XCP queue because it is now the new bottleneck.
- Results from the first experiment validate the hypothesis (see Fig. 15).
  - XCP does not reduce its sending rate upon loss detection it assumes all losses are due to link impairments
  - As a result, XCP has lower utilization and much higher packet drops than with TCP. This is severe congestion on the bottleneck link and bandwidth waste elsewhere, as XCP senders transmit nearly 50% more than they should and XCP is not able to correct that.
  - XCP flows also fail to achieve fair bandwidth sharing, shown by the per-flow utilization and cwnd charts in Fig. 16.

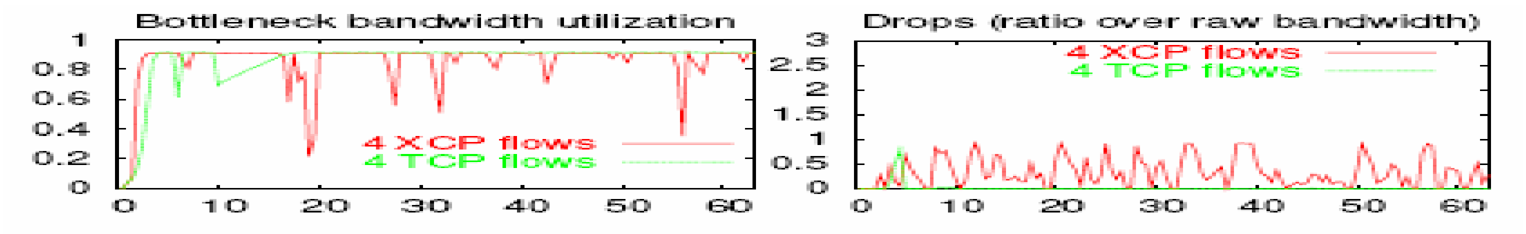


Fig. 15. XCP and TCP performance in the first experiment

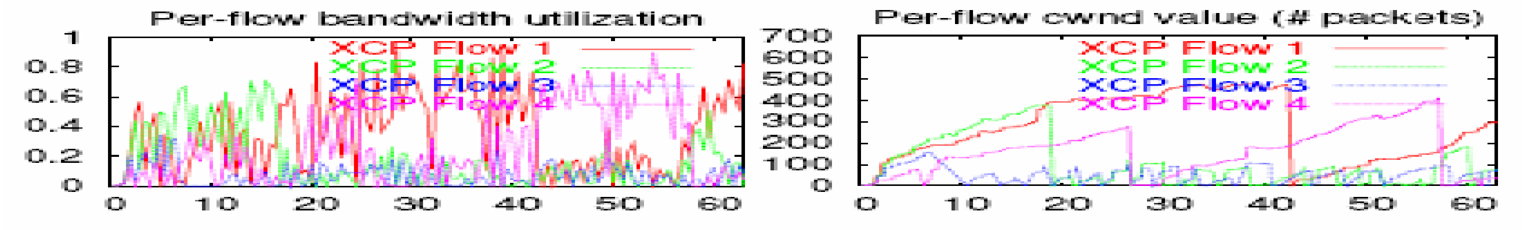


Fig. 16. XCP fails to converge (case 1)

Sensitivity Study

- **Sensitivity to Non-XCP Queues**
- Second experiment results show a similar picture (see Fig. 17).
  - The congestion is even worse this time, XCP senders over-transmit by nearly 100%, and as before, XCP fails to converge to a fair sharing state (see Fig. 18).
- This study shows that XCP is incapable of dealing with the presence of non-XCP queue if it becomes the bottleneck.

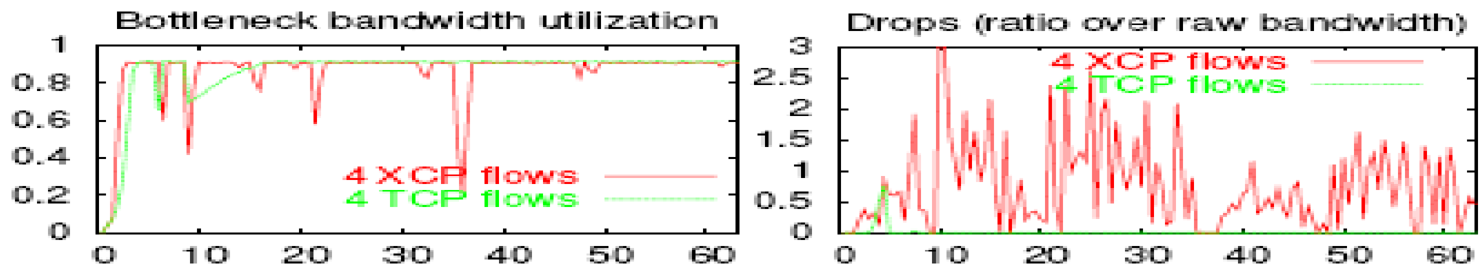


Fig. 17. XCP and TCP performance in the second experiment

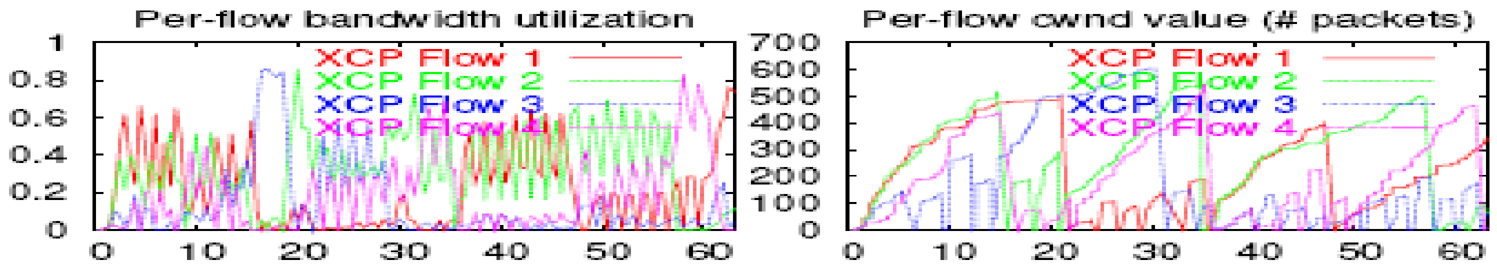


Fig. 18. XCP fails to converge (case 2)

Analysis Of XCP Control Law Applicability

- The sensitivity study suggests that XCP must have accurate input for its control law to work properly.
- External factors, including OS setting and network configuration, can have significant effects on the behavior of XCP control law.
- Analysis is done on the control law input and incorrect control loop scenarios to understand this effect.
- The XCP feedback control operates on the following 5 input values:
  - cwnd and RTT for each flow
  - link capacity
  - aggregated input traffic
  - queue length.
- The first 2 are carried in each XCP packet and indirectly imply a flow's maximum rate.
- The third, link capacity, is a run-time configuration parameter.
- The last 2 are accurately measured at the XCP router itself.

Deployment Issues

- XCP faces a number of significant deployment challenges (if wide scale use).
  - The biggest impact would be the changes to end hosts OS's & routers.
  - Middleboxes (firewalls & network address translators in the Internet), have caused difficulty in deploying even new purely end-to-end protocols.
- XCP faces this and the following additional challenges.
  - Incremental Deployment
    - XCP performance is sensitive to the absence of XCP-capable routers on the path.
    - No way for an XCP endpoint to reliably determine that it is running over XCP-capable queues across the end-to-end path.
  - XCP Responses to Lost Packets
    - XCP would perform well in a lossy environment if the loss was not caused by congestion but, if the loss was caused by an operation over a congested non-XCP queue, XCP would perform poorly.
    - This is because XCP relies on its feedback loop to control congestion and considers all loss as rare events with no significance in congestion control.
  - XCP in the Internet architecture
    - XCP if used as an interposed protocol layer, may cause deployment issues (middleboxes).
    - XCP requires an additional 64 bits per data segment for the XCP header, plus a possible extra 32 bits of XCP feedback header.
    - The extra 64 or 96 bits per packet places additional load on the network, and can effect satellite links.
    - Satellite and wireless links often use header compression, and any XCP approach should also be compatible with such compression.
    - One possible avenue to explore is whether XCP headers are required on every packet or whether only some subset of the packets could carry an (aggregated) XCP header.
    - This type of compression would likely be less robust to packet loss and may lead to a more complicated router algorithm.

Deployment Issues

- Security considerations
  - XCP opens up another vector for denial-of-service attacks, because malicious hosts can potentially disrupt the operation of XCP by promising to send at one rate but sending at another, or by disrupting the flow of XCP information.
  - When deploying XCP in public networks, it would seem that steps need to be taken to avoid malicious activity.
  - This suggests that ingress nodes in each network area probably need to police the XCP flows to some degree, on a per-flow basis.
  - Such a requirement likely undermines one of the attractive features of XCP: its avoidance of requiring per-flow state in the network.
  - In addition, XCP presently is incompatible with IPsec encryption, unless bypasses are denied to allow the XCP header to be copied over to the encrypted tunneled packet and back again to the plaintext side.

Conclusion

- **Significance:**
  - Report on a study to implement XCP in the Linux kernel and to examine its performance in real network test beds.
- **Goals:**
  - Identify issues important to XCP deployment (have found several challenges).
- **Findings:**
  - First found that the implementation was challenging because of the lack of support for precision arithmetic in the Linux kernel.
  - Found that it is important to use a floating point data type in the XCP protocol header.
  - Studied the sensitivity of XCP to accurate reporting of the sending rate, to operation over contention-based media-access protocols, to non-congestion induced losses, and to incremental deployment, they found that XCP has potentially significant performance problems unless miscongruation, estimation errors, and XCP-induced congestion can be detected and prevented.
  - First to report such deployment challenges that XCP must overcome to see widespread deployment in IP-based networks.
- These deployment challenges are significant and appear to present a formidable barrier to acceptance of XCP unless additional XCP extensions are researched and developed.

References

1. Dina Katabi, Mark Handley, and Charlie Rohrs, .Congestion control for high bandwidth-delay product networks,. in *Proceedings of the ACM Conference on Communications Architectures and Protocols (SIGCOMM)*, Aug. 2002, pp. 89.102.
2. Y. Zhang, T. Henderson, An implementation and experimental study of the explicit control protocol (XCP), in: Proceedings of the IEEE INFOCOM, March 2005, pp.1037–1048.