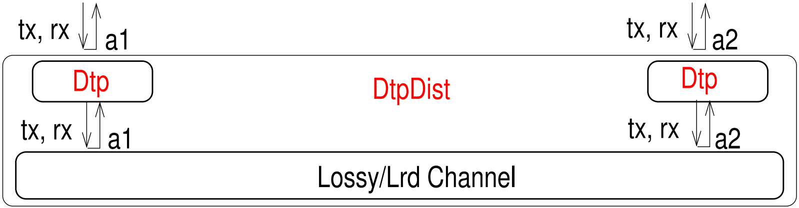
Implementing Fifo Channel using lossy|LRD channel

Shankar

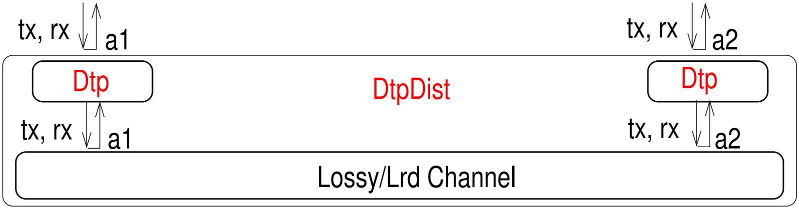
March 27, 2018

Data transfer protocol: DtpDist



- Implements FifoChannel(a1,a2) using lossy/lrd channel

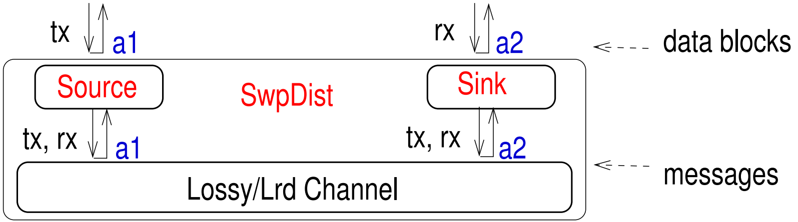
Data transfer protocol: DtpDist



- Implements FifoChannel(a1,a2) using lossy/lrd channel
- Sliding window protocol (Swp)
  - a1 $\rightarrow$ a2 fifo transfer
  - define programs at "algorithm-level": atomic rules
  - prove correctness with modulo-N sequence numbers
  - correctness-preserving refinement to await program
- Obtain DtpDist via correctness-preserving merge of two SwpDist

Outline

Sliding window protocol: SwpDist



- SwpDist(a1, a2, ...)

```
  [x_a1, x_a2] ← startSystem(Lossy/LrdChannel(a1, a2));
   y_a1 ← startSystem(Source(a1, a2, x_a1, ...));
   y_a2 ← startSystem(Sink(a2, a1, x_a2, ...));
   return [y_a1, y_a2]
```

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user
- Source
  - tx $[d_k, k]$ repeatedly until acked // k: seq #

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user
- Source
  - tx $[d_k, k]$ repeatedly until acked // k: seq #
- Sink
  - respond with awaited seq # // cumulative ack
    - has `0`...`4` $\sqcup$ `6`,...`8`: rx 7 / tx 5; rx 5 / tx 9

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user
- Source
  - tx $[d_k, k]$ repeatedly until acked // k: seq #
- Sink
  - respond with awaited seq # // cumulative ack
    - has `0 ... 4` ⊔ `6, ... 8`: rx 7 / tx 5; rx 5 / tx 9
  - passes data blocks to its user in order

Solution using sequence numbers

- $d_0,\ d_1,\ \cdots$: data blocks sent into source by its user
- Source
  - tx $[d_k, k]$ repeatedly until acked // k: seq #
- Sink
  - respond with awaited seq # // cumulative ack
    - has `0 . . . 4` $\sqcup$ `6, . . . 8`: rx 7 / tx 5; rx 5 / tx 9
  - passes data blocks to its user in order
- For good throughput
  - > 1 outstanding at source, buffer out-of-sequence at sink

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user
- Source
  - tx $[d_k, k]$ repeatedly until acked // k: seq #
- Sink
  - respond with awaited seq # // cumulative ack
    - has $0 \ldots 4 \sqcup 6, \ldots 8$: rx 7 / tx 5; rx 5 / tx 9
  - passes data blocks to its user in order
- For good throughput
  - > 1 outstanding at source, buffer out-of-sequence at sink
- Above requires unbounded seq #s (usn)
  - not good for hw implementation
- Instead use cyclic seq #s (csn)

Using modulo-N cyclic sequence numbers

- Use mod-N csn instead of usn
- Send csn $\mathrm{mod}(k, N)$ instead of usn k
- Receiver of *csn* has to infer the corresponding *usn*

Using modulo-N cyclic sequence numbers

- Use mod-N csn instead of usn
- Send csn $\mathrm{mod}(k, N)$ instead of usn k
- Receiver of *csn* has to infer the corresponding *usn*
  - maintains window of "possible" usn
    - say $L \cdots U$
  - maps rcvd *csn* to *usn* with same cyclic value
    - $usn \leftarrow L + \mathrm{mod}(csn - L, N)$;
      if $usn > U$ ignore rcvd *csn*

Using modulo-N cyclic sequence numbers

- Use mod-N csn instead of usn
- Send csn $\text{mod}(k, N)$ instead of usn k
- Receiver of *csn* has to infer the corresponding *usn*
  - maintains window of "possible" usn
    - say $L \cdots U$
  - maps rcvd *csn* to *usn* with same cyclic value
    - $usn \leftarrow L + \text{mod}(csn - L,\ N)$;
      if $usn > U$ ignore rcvd *csn*
- Seq #s in transit must remain close to window

Sliding window protocol with mod-N csn

- Source
  - ng: # blks from user
  - ns: # blks sent at least once
  - na: # blks acked
  - map sbuff: for blks na ··· ng−1
  - send window: na ··· ns+SW−1 // SW < N
  - outstanding window: na ··· ns // ok if low end is na+1

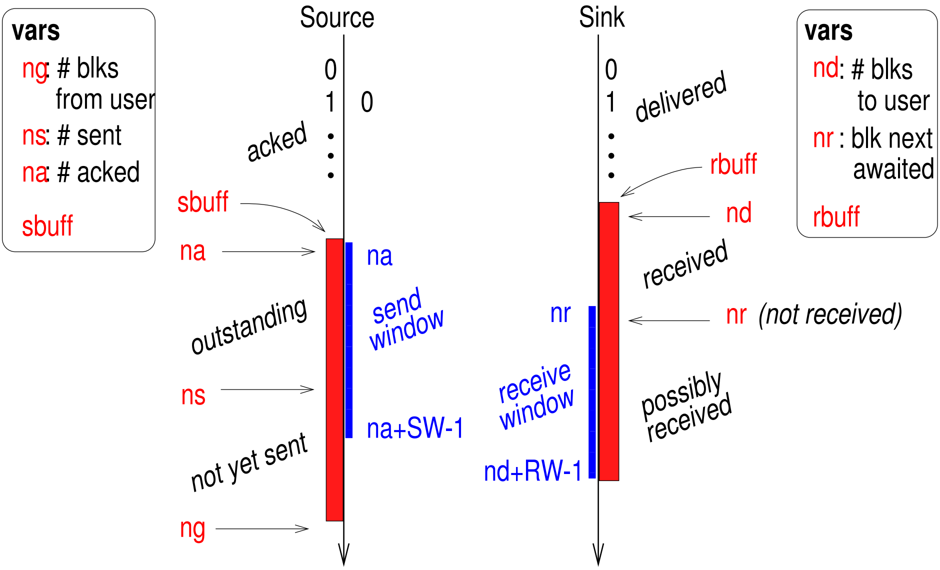
Sliding window protocol with mod-N csn

- Source
  - ng: # blks from user
  - ns: # blks sent at least once
  - na: # blks acked
  - map sbuff: for blks na $\cdots$ ng$-$1
  - send window: na $\cdots$ ns+SW$-$1 // SW $<$ N
  - outstanding window: na $\cdots$ ns // ok if low end is na+1
- Sink
  - nd: # blks to user
  - nr: # contiguous blks rcvd
  - map rbuff: for blks nd $\cdots$ nd + RW$-$1 // RW $<$ N
  - receive window: nr $\cdots$ nd + RW$-$1

Sliding window protocol with mod-N csn

- Source
  - ng: # blks from user
  - ns: # blks sent at least once
  - na: # blks acked
  - map sbuff: for blks na ⋯ ng−1
  - send window: na ⋯ ns+SW−1 // SW < N
  - outstanding window: na ⋯ ns // ok if low end is na+1
- Sink
  - nd: # blks to user
  - nr: # contiguous blks rcvd
  - map rbuff: for blks nd ⋯ nd+RW−1 // RW < N
  - receive window: nr ⋯ nd+RW−1
- Over time, windows slide to increasing seq #s

Sliding windows



Swp mod-N: algorithm-level

Source

```
na,ns,ng ← 0; sbuff ← []
```

Swp mod-N: algorithm-level

Source

na,ns,ng ← 0; sbuff ← []

- db from user:

  sbuff_{ng} ← db; ng++

Swp mod-N: algorithm-level

Source

$\texttt{na,ns,ng} \leftarrow \texttt{0}$; $\texttt{sbuff} \leftarrow \texttt{[]}$

- db from user:

  $\texttt{sbuff}_{\texttt{ng}} \leftarrow$ db; ng++

- ns < min (na+SW, ng):

  send $[\texttt{sbuff}_{\texttt{ns}}, \overline{\texttt{ns}}]$; ns++

Swp mod-N: algorithm-level

Source

$\mathtt{na,ns,ng} \leftarrow \mathtt{0}$; $\mathtt{sbuff} \leftarrow \mathtt{[]}$

- db from user:
  $\mathtt{sbuff}_{\mathtt{ng}} \leftarrow$ db; $\mathtt{ng}$++
- $\mathtt{ns} < \mathtt{min}\,(\mathtt{na+SW}, \mathtt{ng})$:
  send $[\mathtt{sbuff}_{\mathtt{ns}}, \overline{\mathtt{ns}}]$; $\mathtt{ns}$++
- $\mathtt{k}$ in $\mathtt{na..ns}-1$:
  send $[\mathtt{sbuff}_{\mathtt{k}}, \overline{\mathtt{k}}]$

Swp mod-N: algorithm-level

Source

$\texttt{na,ns,ng} \leftarrow \texttt{0}$; $\texttt{sbuff} \leftarrow \texttt{[]}$

- db from user:
  $\texttt{sbuff}_{\texttt{ng}} \leftarrow$ db; ng++
- ns < min (na+SW, ng):
  send $[\texttt{sbuff}_{\texttt{ns}}, \overline{\texttt{ns}}]$; ns++
- k in na..ns−1:
  send $[\texttt{sbuff}_{\texttt{k}}, \overline{\texttt{k}}]$
- rcv [cn]:
  $\texttt{j} \leftarrow \texttt{na} + \overline{\texttt{cn}-\texttt{na}}$
  if (na < j $\leq$ ns)
    sbuff.remove(na..j−1)
    na $\leftarrow$ j

$\overline{\texttt{k}}$: mod(k,N)

Swp mod-N: algorithm-level

Source

```
na,ns,ng ← 0; sbuff ← []
```

- db from user:
  $sbuff_{ng} \leftarrow db$; ng++
- ns < min (na+SW, ng):
  send [$sbuff_{ns}$, $\overline{ns}$]; ns++
- k in na..ns−1:
  send [$sbuff_k$, $\overline{k}$]
- rcv [cn]:
  $j \leftarrow na + \overline{cn - na}$
  if $(na < j \leq ns)$
    sbuff.remove(na..j−1)
    $na \leftarrow j$

Sink

```
nd,nr ← 0;  rbuff ← []
```

- nd < nr
  $rbuff_{nd}$ to user; nd++
- rcv [db,cn]
  $j \leftarrow nr + \overline{cn - nr}$
  if $(nr \leq j < nd + RW)$
    $rbuff_j \leftarrow db$
    while (nr in rbuff.keys)
      $nr \leftarrow nr+1$
  send [$\overline{nr}$]

$\overline{k}$: mod(k,N)

Outline

Desired properties of `SwpDist`

- dbh: auxiliary var indicating seq of blks sent by user

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user
- Desired properties

  X_1 : Inv (nd < nr $\Rightarrow$ $\texttt{rbuff}_{\texttt{nd}} = \texttt{dbh}_{\texttt{nd}}$)

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user

- Desired properties

  X_1 : Inv (nd < nr $\Rightarrow$ $\text{rbuff}_{\text{nd}} = \text{dbh}_{\text{nd}}$)

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send sbuff_{ns}, resend sbuff_{na},
  deliver rbuff_{nd} to user,
  source/sink rcv msg,
  channel progress

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user

- Desired properties

  X_1 : Inv (nd < nr $\Rightarrow$ $\text{rbuff}_{\text{nd}} = \text{dbh}_{\text{nd}}$)

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send sbuff_{ns}, resend sbuff_{na},
  deliver rbuff_{nd} to user,
  source/sink rcv msg,
  channel progress

- Intermediate desired Inv A_1–A_3

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user

- Desired properties

  X_1 : *Inv* (nd < nr ⇒ $\text{rbuff}_{\text{nd}} = \text{dbh}_{\text{nd}}$)

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send sbuff_{ns}, resend sbuff_{na},
  deliver rbuff_{nd} to user,
  source/sink rcv msg,
  channel progress

- Intermediate desired *Inv* A_1–A_3

  A_0 : nd ··· nr−1 in rbuff.keys

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user

- Desired properties

  X_1 : *Inv* (nd < nr $\Rightarrow$ $\text{rbuff}_{\text{nd}} = \text{dbh}_{\text{nd}}$)

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send sbuff_{ns}, resend sbuff_{na},
  deliver rbuff_{nd} to user,
  source/sink rcv msg,
  channel progress

- Intermediate desired *Inv* A_1–A_3

  A_0 : nd $\cdots$ nr$-$1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $\text{rbuff}_{\text{k}} = \text{dbh}_{\text{k}}$ // $A_{0,1} \Rightarrow X_1$.pred

Desired properties of SwpDist

- dbh: auxiliary var indicating seq of blks sent by user

- Desired properties

  X_1 : Inv (nd < nr $\Rightarrow$ $rbuff_{nd} = dbh_{nd}$)

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send $sbuff_{ns}$, resend $sbuff_{na}$,
  deliver $rbuff_{nd}$ to user,
  source/sink rcv msg,
  channel progress

- Intermediate desired Inv A_1–A_3

  A_0 : nd $\cdots$ nr−1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $rbuff_k = dbh_k$ // $A_{0,1} \Rightarrow X_1$.pred

  A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW and ns $\leq$ ng

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_{\mathtt{j}}$, $\overline{\mathtt{j}}$, $\mathtt{j}$]
  - sink sends [$\overline{\mathtt{nr}}$, $\mathtt{nr}$]

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_{\mathtt{j}}$, $\overline{\mathtt{j}}$, j]
  - sink sends [$\overline{\mathtt{nr}}$, nr]
- Sink maps rcvd [$\overline{\mathtt{j}}$, j] wrt rcv window nr $\cdots$ nr+RW$-$1

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{\text{nr}}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW$-$1
  - j in window: mapped correctly

Correct interpretation conditions

- Add *auxiliary usn* field to msgs // not read
  - source sends [$\texttt{sbuff}_j$, $\overline{j}$, j]
  - sink sends [$\overline{nr}$, nr]

- Sink maps rcvd [$\overline{j}$, j] wrt rcv window nr ⋯ nr+RW−1
  - j in window: mapped correctly
  - j < window:

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{\text{nr}}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓,

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]

- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr ⋯ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓,

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [$\texttt{sbuff}_{\texttt{j}}$, $\bar{\texttt{j}}$, j]
  - sink sends [$\overline{\texttt{nr}}$, nr]
- Sink maps rcvd [$\bar{\texttt{j}}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, $\cdots$ ✓,

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\overline{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\overline{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, $\cdots$ ✓, nr−N+RW−1 ✗

Correct interpretation conditions

- Add *auxiliary usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_j$, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, $\cdots$ ✓, nr−N+RW−1 ✗
  - j > window:

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_j$, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ⋯ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓,

Correct interpretation conditions

- Add *auxiliary usn* field to msgs // not read
  - source sends [sbuff_{j}, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ⋯ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓,

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_{\mathtt{j}}$, $\bar{\mathtt{j}}$, j]
  - sink sends [$\overline{\mathtt{nr}}$, nr]

- Sink maps rcvd [$\bar{\mathtt{j}}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ⋯ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, ⋯ ✓,

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, $\cdots$ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, $\cdots$ ✓, nr+N ✗

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{\text{nr}}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr ⋯ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ⋯ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, ⋯ ✓, nr+N ✗
  - desired $Inv\ A_3$

A_3 : data j rcvable ⇒ j in nr−N+RW ⋯ nr+N−1

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr ··· nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ··· ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, ··· ✓, nr+N ✗
  - desired $Inv\ A_3$

A_3 : data j rcvable ⇒ j in nr−N+RW ··· nr+N−1

- Source maps rcvd [$\bar{j}$, j] wrt outstanding window na ··· ns

Correct interpretation conditions

- Add *auxiliary* *usn* field to msgs // not read
  - source sends [sbuff_j, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr ⋯ nr+RW−1
  - j in window: mapped correctly
  - j < window: nr−1 ✓, nr−2 ✓, ⋯ ✓, nr−N+RW−1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, ⋯ ✓, nr+N ✗
  - desired $Inv\ A_3$

    A_3 : data j rcvable ⇒ j in nr−N+RW ⋯ nr+N−1
- Source maps rcvd [$\bar{j}$, j] wrt outstanding window na ⋯ ns
  - desired $Inv\ A_3$

    A_4 : ack j rcvable ⇒ j in ns−N+1 ⋯ na+N

Recap

- Goal

  A_0 : nd ··· nr−1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $\text{rbuff}_k = \text{dbh}_k$

  A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

- Correct interpretation

  A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW ··· nr+N−1

  A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 ··· na+N

Recap

- Goal

  A_0 : nd $\cdots$ nr$-$1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $\text{rbuff}_k = \text{dbh}_k$

  A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

- Correct interpretation

  A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1

  A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- For every step e: $\{A_0 - A_4\}\ e\ \{A_0, A_1, A_2\}$ holds

Recap

- Goal
  A_0 : nd $\cdots$ nr$-$1 in rbuff.keys
  A_1 : (k in rbuff.keys) $\Rightarrow$ rbuff$_k$ $=$ dbh$_k$
  A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
- Correct interpretation
  A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1
  A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N
- For every step e: $\{A_0 - A_4\}$ e $\{A_0, A_1, A_2\}$ holds
- Suffices to establish
  - $\{A_{2,3,4}\}$ e $\{A_3\}$ for e: send data; rcv data affecting nr

- Goal

  A_0 : nd $\cdots$ nr$-$1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $\text{rbuff}_k = \text{dbh}_k$

  A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

- Correct interpretation

  A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1

  A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- For every step e: $\{A_0 - A_4\}\ e\ \{A_0, A_1, A_2\}$ holds

- Suffices to establish
  - $\{A_{2,3,4}\}\ e\ \{A_3\}$ for e: send data; rcv data affecting nr
  - $\{A_{2,3,4}\}\ e\ \{A_4\}$ for e: send ack; rcv ack affecting ns

Outline

Analysis for Lossy Channel

- For `SwpDist` with lossy channel
  - now show that $\mathit{Inv}\ A_3$–A_4 holds if **N ≥ SW + RW**

Preserving A_3 wrt send data

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1

Preserving A_3 wrt send data

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- Send data k preserves A_3.rhs lower bound if N $\geq$ SW + RW
  - k $\geq$ na // guards, A_2
    $\geq$ ns − SW // A_2
    $\geq$ nr − SW // A_2
    $\geq$ nr − N + RW // N $\geq$ SW + RW

Preserving A_3 wrt send data

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- Send data k preserves A_3.rhs lower bound if $N \geq SW + RW$
  - $k \geq na$ // guards, A_2
    $\geq ns - SW$ // A_2
    $\geq nr - SW$ // A_2
    $\geq nr - N + RW$ // $N \geq SW + RW$
- Send data k preserves A_3.rhs upper bound
  - $k \leq na + SW - 1$ // guards, A_2
    $\leq nr + SW - 1$ // A_2
    $\leq nr + N - 1$ // $N \geq SW$

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW ⋆⋆ // A_2

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW ⋆⋆ // A_2
  - let j be rcvable after t_0, so sent at some t_2 after t_1

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW ⋆⋆ // A_2
  - let j be rcvable after t_0, so sent at some t_2 after t_1
    - j $\geq$ na(t_2) $\geq$ na(t_1) // na non-decreasing

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW ⋆⋆ // A_2
  - let j be rcvable after t_0, so sent at some t_2 after t_1
    - j $\geq$ na(t_2) $\geq$ na(t_1) // na non-decreasing
      $\geq$ k+1−SW // ⋆⋆

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW ⋆⋆ // A_2
  - let j be rcvable after t_0, so sent at some t_2 after t_1
    - j $\geq$ na(t_2) $\geq$ na(t_1) // na non-decreasing
      $\geq$ k+1−SW // ⋆⋆
      $\geq$ k+1−N+RW // N $\geq$ SW + RW

Preserving A_3 wrt *nr* increase

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- nr increase preserves A_3 if N $\geq$ SW + RW
  - let nr become k at time t_0.
  - so k−1 rcvd at t_0 or prior, so sent at some t_1 // $t_1 < t_0$
    - ns(t_1) $\geq$ k+1 // guards
    - na(t_1) $\geq$ k+1−SW $\star\star$ // A_2
  - let j be rcvable after t_0, so sent at some t_2 after t_1
    - j $\geq$ na(t_2) $\geq$ na(t_1) // na non-decreasing
      $\geq$ k+1−SW // $\star\star$
      $\geq$ k+1−N+RW // N $\geq$ SW + RW
- Above is an operational proof; see text for an assertional proof

Preserving A_4

A_2 :	na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 :	ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

Preserving A_4

A_2 : na ≤ nr ≤ ns ≤ na+SW

A_4 : ack j rcvable ⇒ j in ns−N+1 ⋯ na+N

- Proof of Inv A_4

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- Proof of Inv A_4
  1. acks sent have non-decreasing usn // nr non-decreasing

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- Proof of Inv A_4
  1. acks sent have non-decreasing usn // nr non-decreasing
  2. acks in transit have non-decreasing usn // no reodering

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- Proof of $Inv\ A_4$
  1. acks sent have non-decreasing usn // nr non-decreasing
  2. acks in transit have non-decreasing usn // no reodering
  3. ack usn lower bounded by na // na is an ack usn

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- Proof of $Inv\ A_4$
  1. acks sent have non-decreasing usn // nr non-decreasing
  2. acks in transit have non-decreasing usn // no reodering
  3. ack usn lower bounded by na // na is an ack usn
  4. Inv (ack j rcvable $\Rightarrow$ na $\leq$ j $\leq$ nr) // 1, 2, 3

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- Proof of $\mathit{Inv}\ A_4$
  1. acks sent have non-decreasing usn // nr non-decreasing
  2. acks in transit have non-decreasing usn // no reodering
  3. ack usn lower bounded by na // na is an ack usn
  4. Inv (ack j rcvable $\Rightarrow$ na $\leq$ j $\leq$ nr) // 1, 2, 3
  5. $\mathit{Inv}\ A_4$ // 4, A_2

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- Proof of Inv A_4
  1. acks sent have non-decreasing usn // nr non-decreasing
  2. acks in transit have non-decreasing usn // no reodering
  3. ack usn lower bounded by na // na is an ack usn
  4. Inv (ack j rcvable $\Rightarrow$ na $\leq$ j $\leq$ nr) // 1, 2, 3
  5. Inv A_4 // 4, A_2
- Above is an operational proof; see text for an assertional proof

Outline

Analysis for LRD Channel

- For SwpDist with LRD channel
  - obviously $\mathit{Inv}\ A_3$–A_4 does not hold for any N
- But $\mathit{Inv}\ A_3$–A_4 holds if
  - LRD channel has max msg lifetime L
  - min time δ between ns increments
  - $\mathtt{N} \geq \mathtt{SW} + \mathtt{RW} + \frac{L}{\delta}$

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_3 : data j rcvable $\Rightarrow$ j in nr$-$N+RW $\cdots$ nr+N$-$1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows
  1 let data j be rcvable at t_0

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

  1 let data j be rcvable at t_0

  2 data j was sent at some $t_1 > t_0 - L$ // 1

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows
  1 let data j be rcvable at t_0
  2 data j was sent at some $t_1 > t_0 - L$ // 1
  3 j $\geq$ na(t_1) $\geq$ ns(t_1) − SW // guards, A_2

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

  1 let data j be rcvable at t_0

  2 data j was sent at some $t_1 > t_0 - L$ // 1

  3 j $\geq$ na(t_1) $\geq$ ns(t_1) − SW // guards, A_2

  4 during $[t_1, t_0]$: ns increases by at most L/δ // guards

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

1 let data j be rcvable at t_0
2 data j was sent at some $t_1 > t_0 - L$ // 1
3 j $\geq$ na(t_1) $\geq$ ns(t_1) − SW // guards, A_2
4 during $[t_1, t_0]$: ns increases by at most L/δ // guards
5 j $\geq$ ns(t_0) − L/δ − SW // 3, 4

Preserving A_3

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

1 let data j be rcvable at t_0
2 data j was sent at some $t_1 > t_0 - L$ // 1
3 j $\geq$ na(t_1) $\geq$ ns(t_1) − SW // guards, A_2
4 during $[t_1, t_0]$: ns increases by at most L/δ // guards
5 j $\geq$ ns(t_0) − L/δ − SW // 3, 4
6 j $\geq$ ns(t_0) − N + RW // 5, N $\geq$ SW + RW + L/δ

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_3 : data j rcvable $\Rightarrow$ j in nr−N+RW $\cdots$ nr+N−1

- A_3.rhs upper bound holds exactly as in lossy channel case
- A_3.rhs lower bound holds as follows

1 let data j be rcvable at t_0

2 data j was sent at some $t_1 > t_0 - L$ // 1

3 j $\geq$ na(t_1) $\geq$ ns(t_1) − SW // guards, A_2

4 during $[t_1, t_0]$: ns increases by at most L/δ // guards

5 j $\geq$ ns(t_0) − L/δ − SW // 3, 4

6 j $\geq$ ns(t_0) − N + RW // 5, N $\geq$ SW + RW + L/δ

7 j $\geq$ nr(t_0) − N + RW // 6, A_2

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows

  1 let ack j be rcvable at t_0

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows
  1 let ack j be rcvable at t_0
  2 ack j was sent at some $t_1 > t_0 - L$ // 1

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows
  1 let ack j be rcvable at t_0
  2 ack j was sent at some $t_1 > t_0 - L$ // 1
  3 j $=$ nr(t_1) $\geq$ na(t_1) $\geq$ ns(t_1) $-$ SW // 2, guard, A_2

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns$-$N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows
  1. let ack j be rcvable at t_0
  2. ack j was sent at some $t_1 > t_0 - L$ // 1
  3. j $=$ nr$(t_1) \geq$ na$(t_1) \geq$ ns(t_1) $-$ SW // 2, guard, A_2
  4. during $[t_1, t_0]$: ns increases by at most L/δ

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows

1 let ack j be rcvable at t_0

2 ack j was sent at some $t_1 > t_0 - L$ // 1

3 j $=$ nr(t_1) $\geq$ na(t_1) $\geq$ ns(t_1) − SW // 2, guard, A_2

4 during $[t_1, t_0]$: ns increases by at most L/δ

5 j $\geq$ ns(t_0) − L/δ − SW // 3, 4

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW

A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ na+N

- A_4.rhs upper bound holds exactly as in lossy channel case
- A_4.rhs lower bound holds as follows

1 let ack j be rcvable at t_0

2 ack j was sent at some $t_1 > t_0 - L$ // 1

3 j $=$ nr(t_1) $\geq$ na(t_1) $\geq$ ns(t_1) − SW // 2, guard, A_2

4 during $[t_1, t_0]$: ns increases by at most L/δ

5 j $\geq$ ns(t_0) − L/δ − SW // 3, 4

6 j $\geq$ ns(t_0) − N + 1 // 5, N $\geq$ SW + RW + L/δ, RW > 0

Outline

Desired progress of Swp

X_2 : (nd $=$ k $<$ ng *leads-to* nd $>$ k)

assuming

- wfair send (of $\mathtt{sbuff}_{ns}$)
- wfair resend (of $\mathtt{sbuff}_{na}$)
- wfair sink user rx (of $\mathtt{rbuff}_{nd}$)
- wfair source rx (of msg from lossy/lrd channel)
- wfair sink rx (of msg from lossy/lrd channel)
- lossy/lrd channel progress

Progress of sliding windows

Progress of sliding windows

P_1 : na $=$ k $<$ nr $\leq$ ns *leads-to* na $>$ k

// src resend, snk rx, src rx, channel progress

Progress of sliding windows

P_1 : na $=$ k < nr $\leq$ ns *leads-to* na > k

// src resend, snk rx, src rx, channel progress

P_2 : (na $=$ k $=$ nr < ns and nr < nd+RW) *leads-to* nr > k

// src resend, snk rx, channel progress

Progress of sliding windows

P_1 : na = k < nr $\leq$ ns *leads-to* na > k

// src resend, snk rx, src rx, channel progress

P_2 : (na = k = nr < ns and nr < nd+RW) *leads-to* nr > k

// src resend, snk rx, channel progress

P_3 : (na = k = nr < ns and nr = nd+RW)

leads-to nr > k // sink user rx, P_2

Progress of sliding windows

P_1 : na = k < nr ≤ ns *leads-to* na > k
// src resend, snk rx, src rx, channel progress

P_2 : (na = k = nr < ns and nr < nd+RW) *leads-to* nr > k
// src resend, snk rx, channel progress

P_3 : (na = k = nr < ns and nr = nd+RW)
leads-to nr > k // sink user rx, P_2

P_4 : na = k = nr < ns *leads-to* nr > k // P_2, P_3

Progress of sliding windows

P_1 : na $=$ k $<$ nr $\leq$ ns *leads-to* na $>$ k
// src resend, snk rx, src rx, channel progress

P_2 : (na $=$ k $=$ nr $<$ ns and nr $<$ nd+RW) *leads-to* nr $>$ k
// src resend, snk rx, channel progress

P_3 : (na $=$ k $=$ nr $<$ ns and nr $=$ nd+RW)
leads-to nr $>$ k // sink user rx, P_2

P_4 : na $=$ k $=$ nr $<$ ns *leads-to* nr $>$ k // P_2, P_3

P_5 : na $=$ k $<$ ns *leads-to* na $>$ k // P_4, P_1

Progress of sliding windows

P_1 : na = k < nr ≤ ns *leads-to* na > k

// src resend, snk rx, src rx, channel progress

P_2 : (na = k = nr < ns and nr < nd+RW) *leads-to* nr > k

// src resend, snk rx, channel progress

P_3 : (na = k = nr < ns and nr = nd+RW)

leads-to nr > k // sink user rx, P_2

P_4 : na = k = nr < ns *leads-to* nr > k // P_2, P_3

P_5 : na = k < ns *leads-to* na > k // P_4, P_1

P_6 : nr = k < ns *leads-to* nr > k // P_5, *Inv* nr ≥ na

P_7 : (ns = k < ng and ns < na+SW) *leads-to* ns > k
// src send

P_8 : (ns = k < ng and ns = na+SW) *leads-to*
(ns = k < ng and ns < na+SW) // P_5

P_9 : ns = k < ng *leads-to* ns > k // P_7, P_8

P_{10} : nr = k < ng *leads-to* nr > k // P_9, P_6

P_{11} : nd = k < ng *leads-to* nd > k // P_{10}, sink user rx

Outline

Correctness-preserving transformation

- Given: algorithm-level program A
  - init, vars, atomic rules, fairness for rules
- Goal: program B that implements A
  - init, vars, threads, await statements, fairness for threads

Correctness-preserving transformation

- Given: algorithm-level program A
  - init, vars, atomic rules, fairness for rules
- Goal: program B that implements A
  - init, vars, threads, await statements, fairness for threads
- Construct B as follows
  - include A-vars
  - each A-rule $\rightarrow$ await statement // preserves
  - additional B-steps do not affect A-vars // safety

Correctness-preserving transformation

- Given: algorithm-level program A
  - init, vars, atomic rules, fairness for rules
- Goal: program B that implements A
  - init, vars, threads, await statements, fairness for threads
- Construct B as follows
  - include A-vars
  - each A-rule $\rightarrow$ await statement // preserves
  - additional B-steps do not affect A-vars // safety

  - allocate local/guest threads to awaits // preserves
  - fairness of A-rules $\rightarrow$ fairness of threads // progress

Source program – 1

- program Source(aL, aR, xL, N, SW, RW)
  - parameters
    - aL: local addr, aR: remote addr, xL: channel access sid
  - constants: DAT, ACK, RTO // data msg, ack msg, timeout
  - functions
    - input mysid.tx: get db from user
    - doTxDat: send ns, resend k // calls xL.tx
    - doRxAck: rcv ack msg // calls xL.rx

Source program – 1

- program Source(aL, aR, xL, N, SW, RW)
  - parameters
    - aL: local addr, aR: remote addr, xL: channel access sid
  - constants: DAT, ACK, RTO // data msg, ack msg, timeout
  - functions
    - input mysid.tx: get db from user
    - doTxDat: send ns, resend k // calls xL.tx
    - doRxAck: rcv ack msg // calls xL.rx
  - main
    - ng, ns, na ← 0, sbuff ← []
    - timer ← OFF // OFF, 0, 1, ···
    - tSrcTx ← startThread(doTxDat())
    - tSrcRx ← startThread(doRxAck())
    - return mysid

Source program – 2

- ```
  input mysid.tx ( aR, db)
    await (true)
      sbuff_ng ← db;
      ng++;
      return;
  ```
  ```

Source program – 2

- input mysid.tx (aR, db)

```
  await (true)
    sbuff_ng ← db;
    ng++;
    return;
```

- function doTxDat()

```
   while (true)
   • await ( (timer = OFF  and  ns < min(ng, na+SW))  or
               (timer > RTO  and  na < ns) )
        ns ← min(ng, na+SW);
        for (j in na..ns−1)
           xL.tx ( aR, [DAT, sbuff_j,  mod(j,N)]);
        timer ← 0;                       //  (re)start timer
```

- function void doRxAck()

```
while (true)
  • Seq msg ← xL.rx();
    ia {msg in Tuple<ACK, 0..N−1>}
    await (true)
      int j ← na + mod(msg[1] − na, N);
      if (na < j ≤ ns)
        for (k in na..j−1) sbuff.remove(k);
        na ← j;
      if (na = ns)
        timer ← OFF;                // stop timer
```

- atomicity assumption: awaits
- progress assumption: weak fairness of threads

Source program: improving performance

- Send new data blocks asap, instead of upon timeout
- Reduce extent of blocking at awaits
  - concurrent access to sbuff
  - duplicate counters
  - ...
- Adapt RTO to measured roundtrip time // congestion control

Sink program – 1

- program Sink(aL, aR, xL, N, SW, RW)
  - parameters
    - aL: local addr, aR: remote addr, xL: channel access sid
  - constants: DAT, ACK // data msg, ack msg
  - functions
    - input mysid.rx: deliver db to user
    - doRxDatTxAck: rcv data, send ack // calls xL.rx, xL.tx

- program Sink(aL, aR, xL, N, SW, RW)
  - parameters
    - aL: local addr, aR: remote addr, xL: channel access sid
  - constants: DAT, ACK // data msg, ack msg
  - functions
    - input mysid.rx: deliver db to user
    - doRxDatTxAck: rcv data, send ack // calls xL.rx, xL.tx
  - main
    - nd, nr ← 0, rbuff ← []
    - tSnkRx ← startThread(doRxDatTxAck())
    - return mysid

Sink program – 2

- `input mysid.rx ()`
  - `await (nd < nr)`

    Seq db $\leftarrow$ $\mathtt{rbuff}_{\mathtt{nd}}$;

    ```
    rbuff.remove(nd);
    nd++;
    return db;
    ```

Sink program – 3

- function doRxDatTxAck()

```
function doRxDatTxAck()
    while (true)
      • Seq msg ← xL.rx();
        ia {msg in Tuple<DAT, 0..N−1, Seq>}
        await (true)
            int j ← nr + mod(msg[2] − nr, N);
            if ((nr ≤ j < nd+RW)
                 and (not j in rbuff.keys))
                rbuff_j ← msg[1];
                while (nr in rbuff.keys)
                    nr++;
            xL.tx(aR, [ACK, mod(nr,N)])
```

- atomicity assumption: awaits
- progress assumption: weak fairness of threads

Outline

Program Dtp overview

- `DtpDist`: merge two `SwpDist`, but one lossy/lrd channel
- `Dtp`: `Source` + `Sink`, but share channel access

Program Dtp overview

- DtpDist: merge two SwpDist, but one lossy/lrd channel
- Dtp: Source + Sink, but share channel access
  - params, ia, constants: // same as src, snk

Program Dtp overview

- DtpDist: merge two SwpDist, but one lossy/lrd channel
- Dtp: Source + Sink, but share channel access
  - params, ia, constants: // same as src, snk
  - Source
    - ng, ns, na,sbuff, timer
    - mysid.tx()
    - doTxDat(): thread tSrcTx // await{.. xL.tx ..}
    - ~~doRxAck()~~: thread ~~tSrcRx~~ // xL.rx

Program Dtp overview

- DtpDist: merge two SwpDist, but one lossy/lrd channel
- Dtp: Source + Sink, but share channel access
  - params, ia, constants: // same as src, snk
  - Source
    - ng, ns, na,sbuff, timer
    - mysid.tx()
    - doTxDat(): thread tSrcTx // await{.. xL.tx ..}
    - ~~doRxAck()~~: thread ~~tSrcRx~~ // xL.rx
  - Sink
    - nd, nr, rbuff
    - mysid.rx
    - ~~doRxDatTxAck~~: thread ~~tSnkRx~~ // xL.rx, await{.. xL.tx ..}

Program Dtp overview

- DtpDist: merge two SwpDist, but one lossy/lrd channel
- Dtp: Source + Sink, but share channel access
  - params, ia, constants: // same as src, snk
  - Source
    - ng, ns, na,sbuff, timer
    - mysid.tx()
    - doTxDat(): thread tSrcTx // await{.. xL.tx ..}
    - ~~doRxAck()~~: thread ~~tSrcRx~~ // xL.rx
  - Sink
    - nd, nr, rbuff
    - mysid.rx
    - ~~doRxDatTxAck~~: thread ~~tSnkRx~~ // xL.rx, await{.. xL.tx ..}
  - doRxDatAck(): thread tRx // xL.rx, await{.. xL.tx ..}
    - rcv data/ack msg, do doRxAck / doRxDatTxAck

Program Dtp – 1

- params: aL, aR, xL, N, SW, RW
- main
  - ng, ns, na, sbuff, timer
  - nd, nr, rbuff
  - tSrcTx ← startThread (doTxDat())
  - tSnkRx ← startThread (doRxDatAck())
- input mysid.tx (aR, db)
  <as in Source()>
- input mysid.rx ()
  <as in Sink()>
- function doTxDat()
  <as in Source()>

Program Dtp – 2

- function doRxDatAck()

```
while (true)
  • Seq msg ← xL.rx();
    ia {msg in union ( Tuple<ACK, 0..N−1>,
                       Tuple<DAT, 0..N−1, Seq> }
  if (msg[0] = ACK)
     <Source.doRxAck() update>
  else if (msg[0] = ACK)
     <Sink.doRxDatTxAck() update>
```

- atomicity assumption: awaits
- progress assumption: weak fairness of threads

Proving that DtpDist implements Fifo Channel

- Obtain fifo channel inverse
  - only two addresses, so no internal nondeterminism
- Define program Z of DtpDist and fifo channel inverse
- Define assertions that Z must satisfy
- Proof that Z satisfies them follows from Swp properties
  - fifo-channel inverse $\texttt{txh}_{\texttt{a1},\texttt{a2}}$ = aux var Dtp(a1).dbh
  - see text for details

Outline

Adding graceful closing to Dtp

- Allow user to close the data transfer
  - open $\rightarrow$ closing $\rightarrow$ closed
  - after call: user cannot tx, but can rx
  - returns when
    - remote user also has requested closing
    - all data in both directions have been delivered to users

Adding graceful closing to Dtp

- Allow user to close the data transfer
  - open → closing → closed
  - after call: user cannot `tx`, but can `rx`
  - returns when
    - remote user also has requested closing
    - all data in both directions have been delivered to users

- New messages
  - `[FIN]`: indicates closing
  - `[FINACK]`: ack to `[FIN]`

- New flags, all initially false:
  - `finRcvd`
  - `finAckRcvd`
  - `closed`

Graceful-closing Dtp – 1

- input `mysid.close()`
  - wait for na = ng; // all outgoing data acked
  - repeatedly send `[FIN]` until `finAckRcvd` is true;
  - wait for `finRcvd` to be true;
  - wait for nd = nr; // no more incoming data
  - set `closed` to true;
  - return;

Graceful-closing Dtp – 2

- Augment doRxDatAck() to handle FIN and FINACK

  Change part after "msg ← xL.rx()" to:

  - if (not closed)

    if msg is DAT or ACK: handle as before

    if msg is FIN: set finRcvd, send [FINACK]

    if msg is FINACK: set finAckRcvd

  - else if (closed and msg is [FIN])

    send [FINACK]

Graceful-closing Dtp – 2

- Augment `doRxDatAck()` to handle `FIN` and `FINACK`

  Change part after "`msg` ← `xL.rx()`" to:

  - if (not closed)

    if msg is `DAT` or `ACK`: handle as before

    if msg is `FIN`: set `finRcvd`, send `[FINACK]`

    if msg is `FINACK`: set `finAckRcvd`
  - else if (closed and msg is `[FIN]`)

    send `[FINACK]`

- Modify `mysid.rx`
  - return `[0,data]` if nr > nd // incoming data
  - return `[−1]` if `finRcvd` and nr = nd // no more incoming data

Graceful-closing DtpDist(j, k)

- Safety properties
  - if j.rx returns [0,data]: j.drxh $\circ$ [data] prefix-of k.dtxh
  - if j.rx returns [$-$1] or j is closed:
    j.drxh $=$ k.dtxh and k is closing or closed.
- Progress properties
  - j.tx ongoing *leads-to* j.tx returns
  - j.rx ongoing, j.drxh $\neq$ k.dtxh *leads-to* j.rx returns
  - j.rx ongoing, j.drxh $=$ k.dtxh, k is closing or closed
    leads-to j.rx returns
  - j closing, k is closing or closed
    leads-to j becomes closed or j.rx not ongoing

Outline

Abortable Dtp

- Graceful-closing terminates connection
  - but not dtp systems or lossy/lrd channel
  - closed dtp system still needs to respond to `FIN`

- To close dtp systems and channel, need to modify `Dtp`
  - abort if `FINACK` not rcvd within some K sends of `FIN`
  - can do the same for data transfer
  - abort: return guest threads, retrieve local threads, end system

- Abortable DtpDist provides a weaker service
  - data delivered is prefix of data sent // still holds
  - ~~all data sent is delivered when close returns~~
  - ~~all data sent is eventually delivered~~