# Calendaring for Wide Area Networks

September 10, 2014

# Overview

# Inter-DC WANs

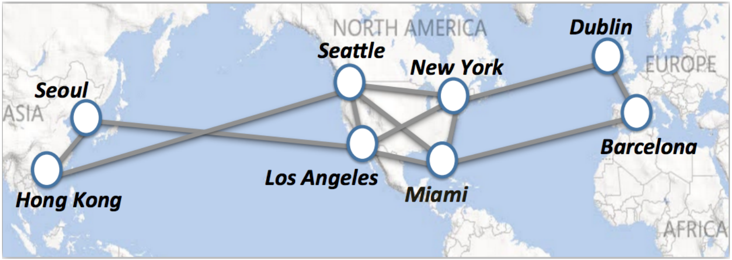



Figure: [1]

[1] Achieving High Utilization with Software-Driven WAN, SIGCOMM'13

# Inter-DC WANs

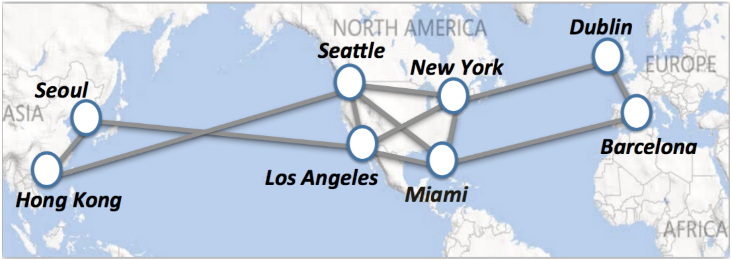



Figure: [1]

Large transfers with preassigned *deadlines over Inter-DC WANs.*

*Examples:*

*moving new search indexes from one DC to another...*

*collected datasets on one DC for later analysis on another Big Data DC...*

[1] Achieving High Utilization with Software-Driven WAN, SIGCOMM '13

# Challenges

- Long-running transfers are often time-critical.
- The network should at all time be able to serve high-priority traffic and long-running traffics.

# The Calendaring Problem

- No delays and zero loss for high-priority traffic.
- When not limited by network capacity, long-running requests are fully met before the deadline.
- When demands exceed network capacity, continue to offer guarantees such as maximizing the minimal fraction of transfers that finish before deadline (**fairness**), or maximizing a specified total utility function.

# Related Work

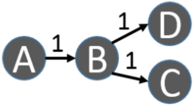
- B4: Experience with a Globally-Deployed Software Defined WAN (Google, SIGCOMM '13)
- Achieving High Utilization with Software-Driven WAN (Microsoft, SIGCOMM '13)

They operate at high utilizations, but these schemes only plan for a single future timestep. Any transfers that last longer receive no guarantees.

# Related Work

NetStitcher uses information about future bandwidth availability to move data between DCs at low costs; it does not support deadlines for requests.

# Example

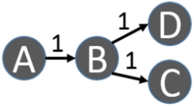



$a$: arrival, $D$: demand, $d$: deadline

| Request | $a$ | $D$ | $d$ |
|---|---|---|---|
| $R_1$: A → B | 1 | 4 | 6 |
| $R_2$: A → C | 1 | 2 | 8 |
| $R_3$: A → D | 1 | 2 | 7 |
| $R_4$: B → C | 4 | 5 | 9 |

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| **Offline** | [2,5] | 1,8 | [6,7] | [4,7], 9 |
| **EDF** | [1,4] | [7,8] | [5,6] | [4,6][9,10] |
| **Greedy** | [1,8] | [1,6] | [1,6] | [4,9] |
| **Online** | [1,4] | [7,8] | [5,6] | [4,6][9,10] |
| **Tempus** | [1,6] | [1,8] | [1,7] | [4,9] |

# Example




$a$: arrival, $D$: demand, $d$: deadline

| Request | $a$ | $D$ | $d$ |
|---|---|---|---|
| $R_1$: A → B | 1 | 4 | 6 |
| $R_2$: A → C | 1 | 2 | 8 |
| $R_3$: A → D | 1 | 2 | 7 |
| $R_4$: B → C | 4 | 5 | 9 |

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| **Offline** | [2,5] | 1,8 | [6,7] | [4,7], 9 |
| **EDF** | [1,4] | [7,8] | [5,6] | [4,6][9,10] |
| **Greedy** | [1,8] | [1,6] | [1,6] | [4,9] |
| **Online** | [1,4] | [7,8] | [5,6] | [4,6][9,10] |
| **Tempus** | [1,6] | [1,8] | [1,7] | [4,9] |

- **EDF** picks requests strictly on deadline order, preventing it from finding beneficial schedules that simultaneously schedule requests on different parts of the network. (*e.g.*, schedule $R_2$ early so that $R_1$ and $R_4$ can run simultaneously).
- **Greedy** (*e.g.*, SWAN) is enable to plan into the future, *i.e.*, fair share at each timestep does not translate into meeting deadlines.
- **Online** plans into the future for all the requests that it is **currently aware of**.

# Guildlines for Online Temporal Planning

- The offline calendaring problem can be formulated as a LP.
- Planning in the midst of dynamically evolving conditions.
- Promises: do not renege on promises
- Conservation of computation.

# Problem Definition

| | |
|---|---|
| $G = (V, E)$ | $\lvert V \rvert = n$, $\lvert E \rvert = m$ |
| $c_e$ | edge capacity |
| request | $(a_i, b_i, d_i, D_i, s_i, t_i, \mathcal{P}_i)$ |
| $a_i$ | the aware time by TEMPUS of the request |
| $b_i$ | begin time for scheduling |
| $d_i$ | deadline for scheduling |
| $D_i$ | demand of the request |
| $s_i$ | source node of the request |
| $t_i$ | target node of the request |
| $\mathcal{P}_i$ | admissible paths from s to t of the request |
| $f_{i,p,t}$ | flow allocated for request $i$ on path $p \in \mathcal{P}_i$ in time $t$ |
| $\alpha_i$ | the fraction of request satisfied for request $i$ |
| $\max \min_i \alpha_i$ | maximize the smallest fraction $\alpha_i$ (fairness) |
| $\delta = avg_i \alpha_i$ | secondary utility function |

# Offline Case

$$h_{e,t}(f) = \sum_{i:b_i \leq t \leq d_i} \sum_{p \in \mathcal{P}_i : e \in p} f_{i,p,t} \qquad \text{(capacity usage)}$$

$$\varphi_i(f) = \sum_{t=b_i}^{d_i} \sum_{p \in \mathcal{P}_i} f_{i,p,t} \qquad \text{(satisfied demand)}$$

$$g(f) = \sum_{i}^{k} \sum_{p \in \mathcal{P}_i} \sum_{t=b_i}^{d_i} u_{i,p,t} \cdot f_{i,p,t} \qquad \text{(utility)}$$

# Offline Case (Impractical)

formulate into an LP:

$$
\begin{aligned}
LP(\alpha, U) = \{ & h_{e,t}(f) \leq c_e && \text{(capacity)} \\
& \varphi_i(f) \leq D_i && \text{(demand)} \\
& \varphi_i(f) \geq \alpha D_i && \text{(fairness)} \\
& g(f) \geq U && \text{(utility)} \\
& f \geq 0 \}
\end{aligned}
$$

# Online (Challenges)

- Computationally expensive
- can lead to a substantially worse solution
- Running the offline LP at each timestep treats **requests that arrive later** unfairly (smaller $\alpha$ for later requests)

# Online (TEMPUS's Approach)

- TEMPUS systematically *under-allocates* future edges
- TEMPUS renege upon promises, but it *rarely* happens
- For faster computation, TEMPUS employs **sliding window** approach, i.e., long requests are broken into smaller requests with their demand scaled accordingly.

# Online (TEMPUS's Approach)

TEMPUS systematically under-allocates future edges. When planning at time $\tau$, the fraction of edge capacity that is available for allocation at time $t > \tau$ is denoted by $\beta_{e,t,\tau}$.
TEMPUS chooses $\beta$ to be a function that decreases with $t - \tau$.
Intuitively, the further an edge is into the future, the less its capacity can be used up during the current timestep.

# Packing-Covering Solver

Given a variable vector $x$, a *packing* matrix $P$ and a *covering* matrix $C$, a mixed packing-covering problem finds a feasible solution for these inequalities:

$$\{Px \leq 1, Cx \geq 1, x \leq 0\}$$

# Packing-Covering Solver

Given a variable vector $x$, a *packing* matrix $P$ and a *covering* matrix $C$, a mixed packing-covering problem finds a feasible solution for these inequalities:

$$\{Px \leq 1, Cx \geq 1, x \leq 0\}$$

Let $x = 0$ be the starting zero solution; all packing constraints are satisfied, but none of the covering constraints are met. Suppose there exists an increment $\Delta x$ that satisfies:

$$\max P(x + \Delta x) - \max Px \leq \min C(x + \Delta x) - \min Cx$$

which ensures that the smallest covering constraint improves by more than the largest packing constraint. Incrementing by such $\Delta x$ will yield a feasible solution.

# Young's Method

Young's method[2], first smoothes min and max:

$$\min_{1 \le i \le n} x_i \ge l\min(x) \triangleq -\ln(\sum_{i=1}^{n} e^{-x_i})$$

$$\max_{1 \le i \le n} x_i \le l\max(x) \triangleq \ln(\sum_{i=1}^{n} e^{x_i})$$

second, Young restricts the increment per iteration to a small amount and to one variable $x_i$ in $x$. Then the condition becomes:

$$\frac{\partial l\max(Px)}{\partial x_i} \le \frac{\partial l\min(Cx)}{\partial x_i}$$

[2]N.E. Young. Sequential and parallel algorithms for mixed packing and covering, FOCS '01

# Online (TEMPUS's Approach)

Observe that

- edge capacity translates to a packing constraint,
- request satisfaction translates to a covering constraint,
- the variable vector $x$ contains flow allocation.
- the old solution from previous timestep is a feasible searching point for the next timestep

# Online (TEMPUS's Approach)

Observe that

- edge capacity translates to a packing constraint,
- request satisfaction translates to a covering constraint,
- the variable vector $x$ contains flow allocation.
- the old solution from previous timestep is a feasible searching point for the next timestep
  - none of the packing constraints are violated (new edges have flow 0 and every old edge has strictly large capacity to allocate)
  - the old covering constraints are unchanged whereas the new covering constraints are unmet (flow for new requests is 0)
  - none of the work done by previous timesteps, in terms of allocating flow, needs to be redone (speeds-up the computation)

# Online (TEMPUS's Approach)

Observe that $LP(\alpha, U)$ is exactly in the form of packing-covering problem, after we normalize the constraints so that the right hand side is all 1s.

- Start with $U = 0$ and conduct a search to find the largest $\alpha$ for which $LP(\alpha, 0)$ is feasible.
- The resulting $\alpha$ is then kept fixed and we search for the largest $U$ for which $LP(\alpha, U)$ is feasible.
- Use Young's method to determine whether a flow vector $f$ exists that satisfies $LP(\alpha, U)$ given some values of $\alpha$ and $U$.

# Feasibility Check of $LP(\alpha, U)$

TEMPUS uses internal variables, each corresponding to a constraint of $LP(\alpha, U)$ to search check for feasibility ($\varepsilon$ is an accuracy parameter and $N$ is a scaling factor):

$$y_{e,t} \triangleq e^{\frac{N}{\varepsilon \cdot c_e} \cdot h_{e,t}(f)} \qquad \text{(capacity)}$$

$$q_i \triangleq e^{\frac{N}{\varepsilon \cdot D_i} \cdot \varphi_i(f)} \qquad \text{(demand-packing)}$$

$$z_i \triangleq e^{-\frac{N}{\varepsilon \cdot \alpha \cdot D_i} \cdot \varphi_i(f)} \qquad \text{(fairness)}$$

$$r \triangleq e^{-\frac{N}{\varepsilon \cdot U} \cdot g(f)} \qquad \text{(utility)}$$

The running time for the feasibility check is $O(\varepsilon^{-2}(mT + k)\ln(mTk))$. In TEMPUS, $\varepsilon = 0.1$.

# Feasibility Check of $LP(\alpha, U)$

1. $y_{e,t} \leftarrow 1,\ q_i \leftarrow 1,\ z_i \leftarrow 1,\ r \leftarrow 1,\ f \leftarrow 0$
2. Find $1 \leq i^* \leq k,\ b_{i^*} \leq t^* \leq d_{i^*},\ p^* \in \mathcal{P}_{i^*}$, s.t.:

$$\frac{\partial l \max(Px)}{\partial x_i} \leq \frac{\partial l \min(Cx)}{\partial x_i}$$

3. $y \leftarrow \varepsilon \cdot \min\left(D_{i^*}, \min_{e \in p^*} c_e\right)$
4. Update $y_{e,t},\ q_i,\ z_i,\ r,\ f \ldots$ (details omitted)

# TEMPUS's workflow

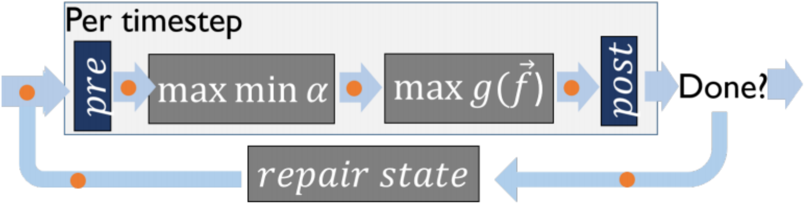



**Figure 3: TEMPUS's workflow**

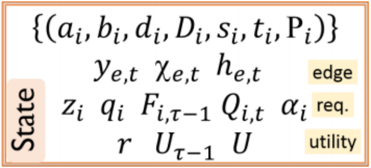



**Figure 4: State kept by TEMPUS; at time-step $\tau$, for currently active requests $i \in R(\tau)$ and future time-steps $t \geq \tau$.**

# Experiment

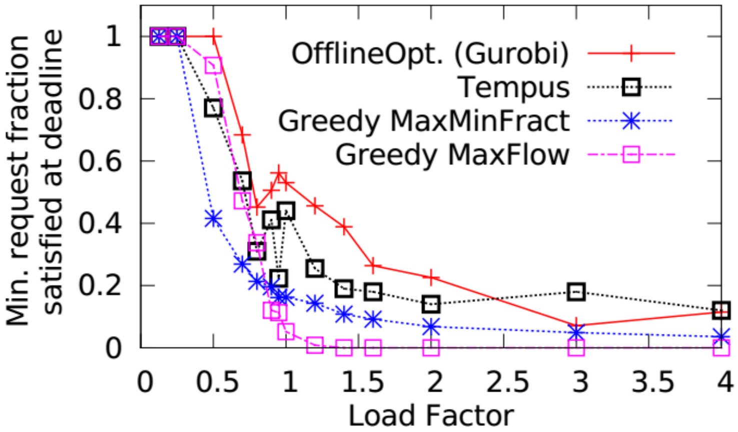



**Figure 6: Minimum request fraction satisfied post-facto.**

# Experiment

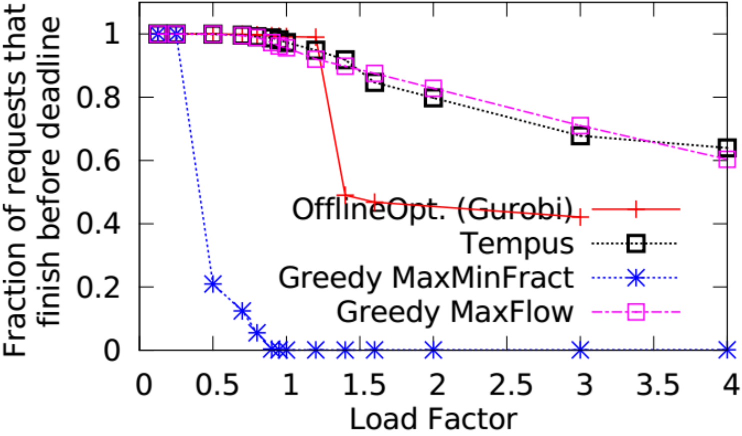



**Figure 7: Fraction of transfers that finish before deadline.**

# Experiment

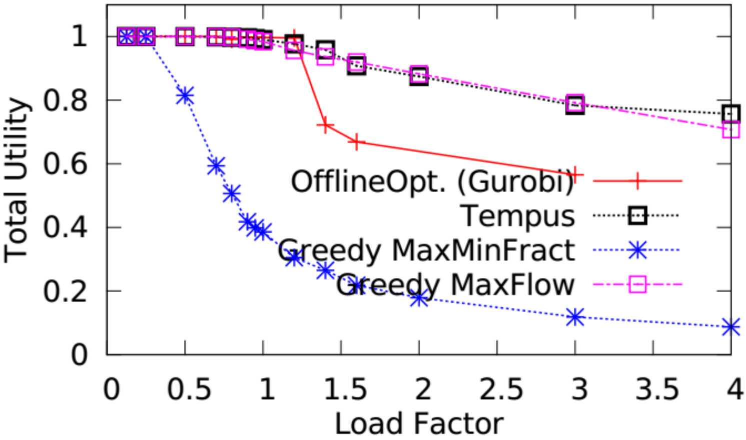



**Figure 8: Total utility.**

# Comparison with Yu's work

- Minor difference in application scenario (TEMPUS doesn't consider a store-and-forward architecture)
- Theoretical breakthrough (applying Young's method online, parallelizing Young's Algorithm) v.s. online heuristic
- Production WAN experiment v.s. small cluster experiment

# Thank You

Q&A